Searching

The Search Problem

- Given an array A storing n numbers, and a target number v, locate the position in A (if it exists) where A[i] = v
- Example
  - Input: A = {3,8,2,15,99,52}, v = 99
  - Output: position 4
- Notes
  - Array positions (indexes) go from 0..n-1
  - When the target does not exist in the array, return an undefined position, such as -1

Linear Search

Linear Search

- Start from the leftmost element of array A and one by one compare v with each element of A
- If v matches with an element, return the index.
- If v doesn’t match with any of elements, return -1.

Algorithm 1: Linear Search

Algorithm *LinearSearch(A,v):*

Input: *An array A of n numbers, search target v*
Output: *position i where A[i]=v, -1 if not found*

```
for i ← 0 to n - 1 do
    if A[i] = v then
        return i
return -1
```

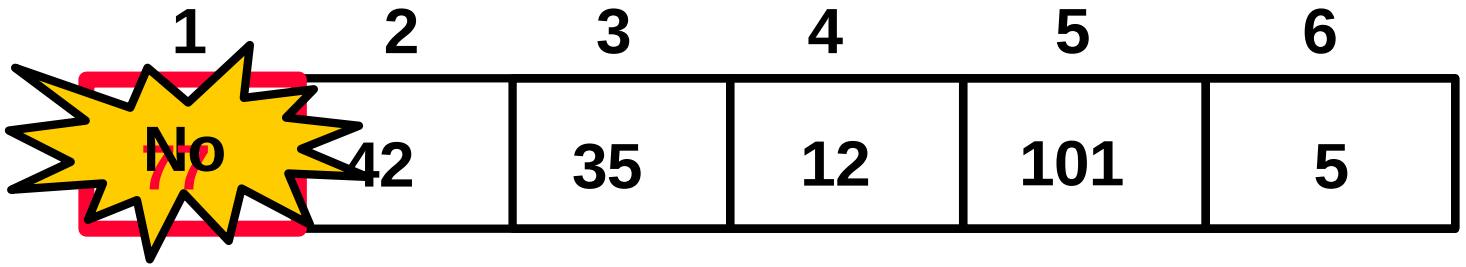
Linear Search: example

- Start from the leftmost element of array A and one by one compare v (101) with each element of A
- If v matches with an element, return the index.
- If v doesn't match with any of elements, return -1.

1	2	3	4	5	6
77	42	35	12	101	5

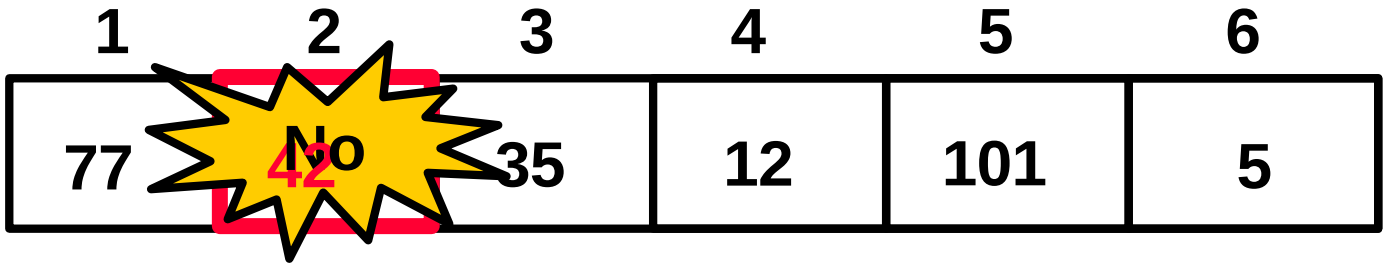
Linear Search: example

- Start from the leftmost element of array A and one by one compare v (101) with each element of A
- If v matches with an element, return the index.
- If v doesn't match with any of elements, return -1.



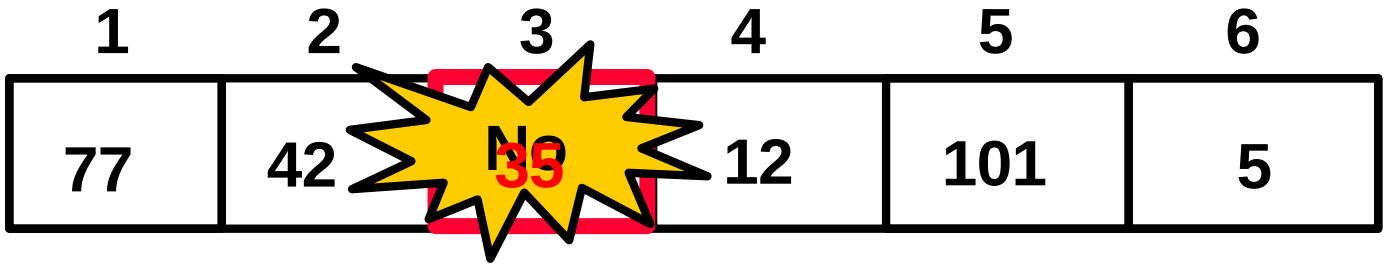
Linear Search

- Start from the leftmost element of array A and one by one compare v (101) with each element of A
- If v matches with an element, return the index.
- If v doesn't match with any of elements, return -1.



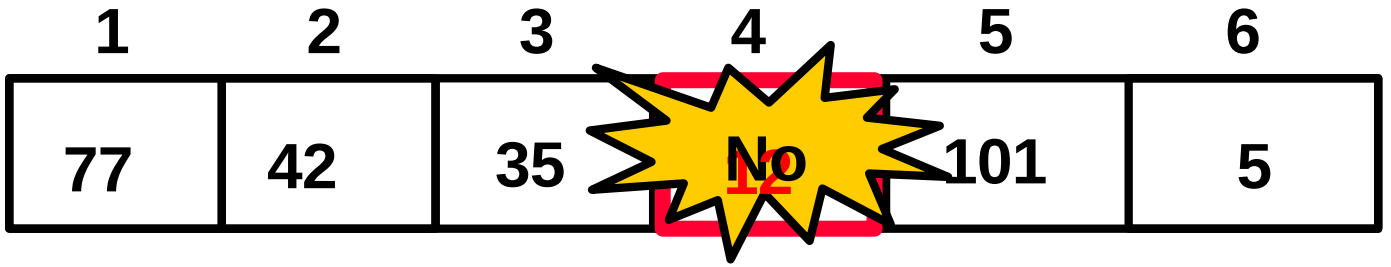
Linear Search

- Start from the leftmost element of array A and one by one compare v (101) with each element of A
- If v matches with an element, return the index.
- If v doesn't match with any of elements, return -1.



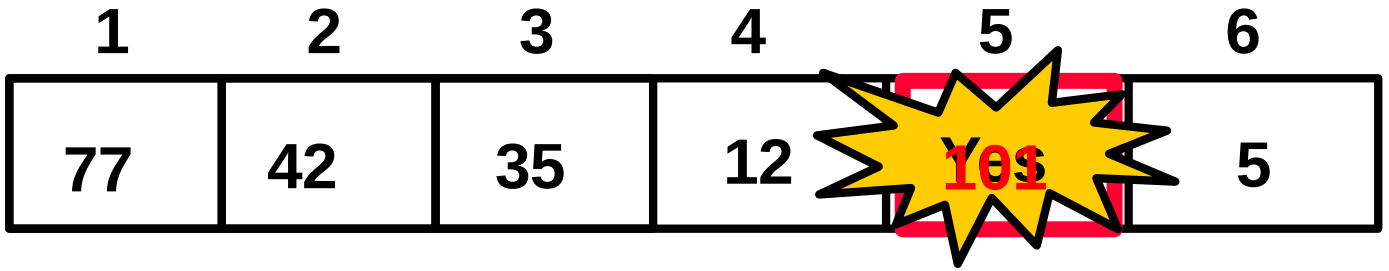
Linear Search

- Start from the leftmost element of array A and one by one compare v (101) with each element of A
- If v matches with an element, return the index.
- If v doesn't match with any of elements, return -1.



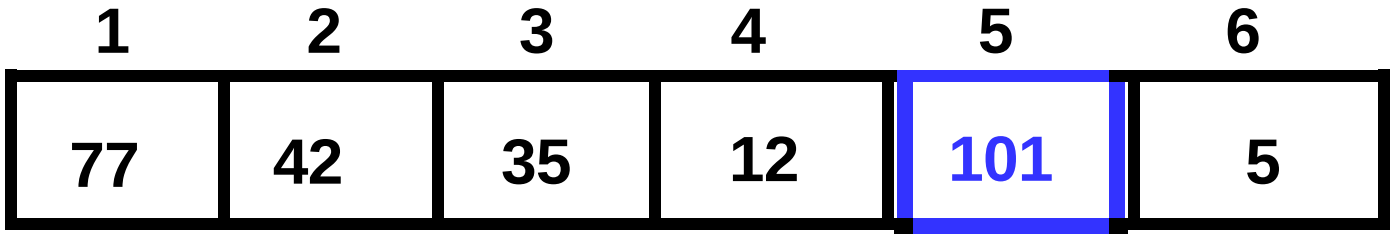
Linear Search

- Start from the leftmost element of array A and one by one compare v (101) with each element of A
- If v matches with an element, return the index.
- If v doesn't match with any of elements, return -1.



Linear Search

- Start from the leftmost element of array A and one by one compare v (101) with each element of A
- If v matches with an element, return the index.
- If v doesn't match with any of elements, return -1.



return the index 5

Linear Search time complexity

- Worst-case: O(n)
  - If v does not exist or if v is the last element in the array
- Best-case: O(1)
  - When the target element v is the first element in the array
- Average-case: O(n)
  - e.g., if the target element is somewhere in the middle of the array, the for-loop will iterate n/2 times. Still O(n)

Binary Search

Searching in sorted arrays

- Suppose the array is arranged in increasing or non-decreasing order
- Linear search on a sorted array still yields the same analysis
  - O(n) worst-case time complexity
- Can exploit sorted structure by performing **binary search**
- Strategy: inspect middle of the search list so that half of the list is discarded at every step
- **Binary Search Algorithm** is a searching algorithm used in a sorted array by **repeatedly dividing the search interval in half**. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to

O(log N).

Algorithm 2: Binary Search

Algorithm *BinarySearch(A,v):*
Input: *A* ***SORTED*** *array A of n numbers, search target v*
Output: *position i where A[i]=v, -1 if not found*

```
low ← 0, high ← n-1
while low <= high do
   mid ← (low+high)/2
   if A[mid]= v then
      return mid
   else if A[mid] < v then
      low ← mid+1
   else
      high ← mid-1
```

جامعة شط العرب كلية
العلوم
قسم علوم الحاسبات

المرحلة الثالثة
مادة هياكل بيانات2
أستاذ المادة: م.م مصطفى صبحي سلمان

return -1

Binary Search time complexity

- Time complexity analysis: how many iterations of the while loop?
  (assume worst-case)

- Observe values for low and high
  - Note that before loop entry,

    size of search space = n = high-low+1.
  - On each suceeding iteration, search space is cut in half
  - Loop terminates when search space collapses to 0 or 1

Binary Search time complexity

- search space size — iteration number
  n — 1
  n/2 — 2
  n/4 — 3
  … — …
  1 — x
- x = number of iterations
- Observe $2^x = n$, $\log 2^x = \log n$,
  x = log n iterations carried out
- Binary Search worst-case time complexity:
  O(log n)

Binary Search time complexity

- Worst-case: O(log n)
  - If v does not exist
- Best-case: O(1)
  - When the target element v happens to be in the middle of the array
- Average-case: O(log n)
  - Technically, some statistical analysis needed here (beyond the scope of this course)

Binary Search (version 2)

- Can use recursion instead of iteration

- *BinSearch(A,v):*
  return BinSearchHelper(A,0,n-1,v)

- *BinSearchHelper(A,low,high,v):*
  if low > high then
  return -1
  mid ← (low+high)/2
  if A[mid] = v then
  return mid
  else if A[mid] < v then
  return BinSearchHelper(A,mid+1,high,v)
  else
  return BinSearchHelper(A,low,mid-1,v)

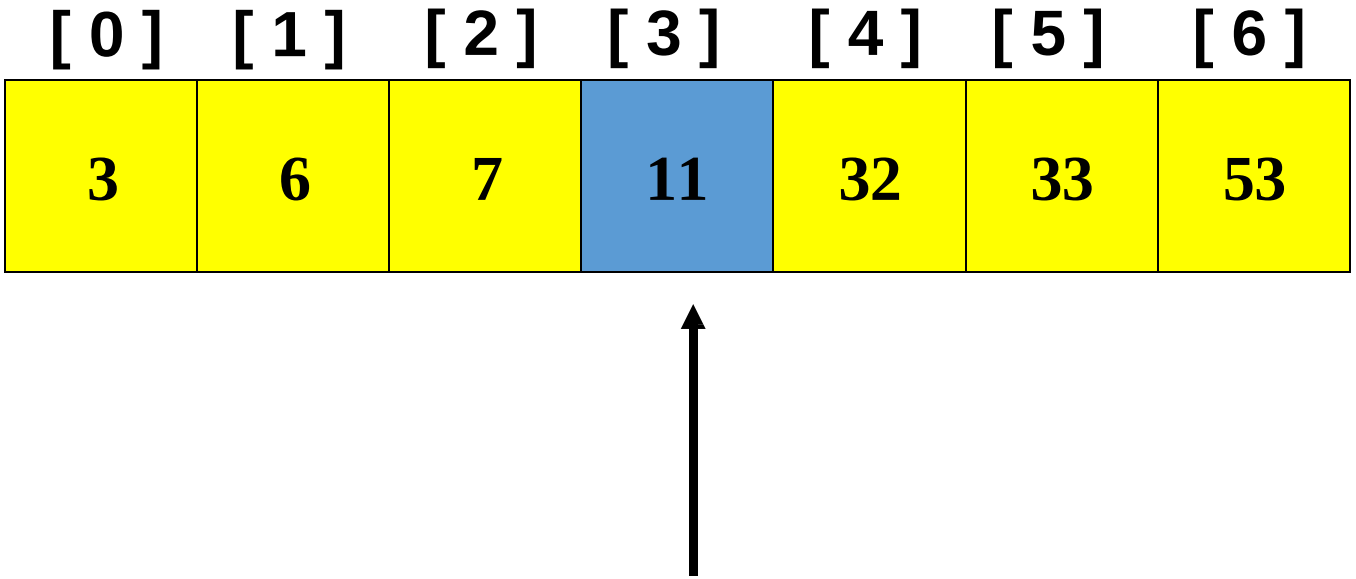
Binary Search: example

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

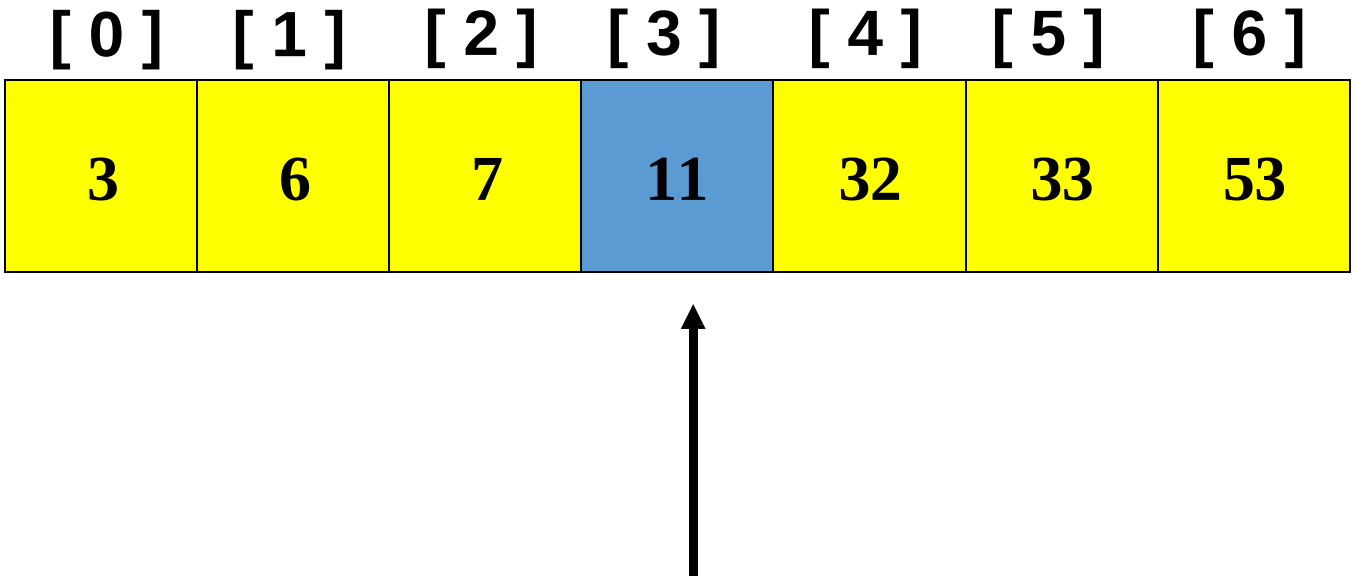
Binary Search: example

Example: sorted array of integer keys. Target=7.



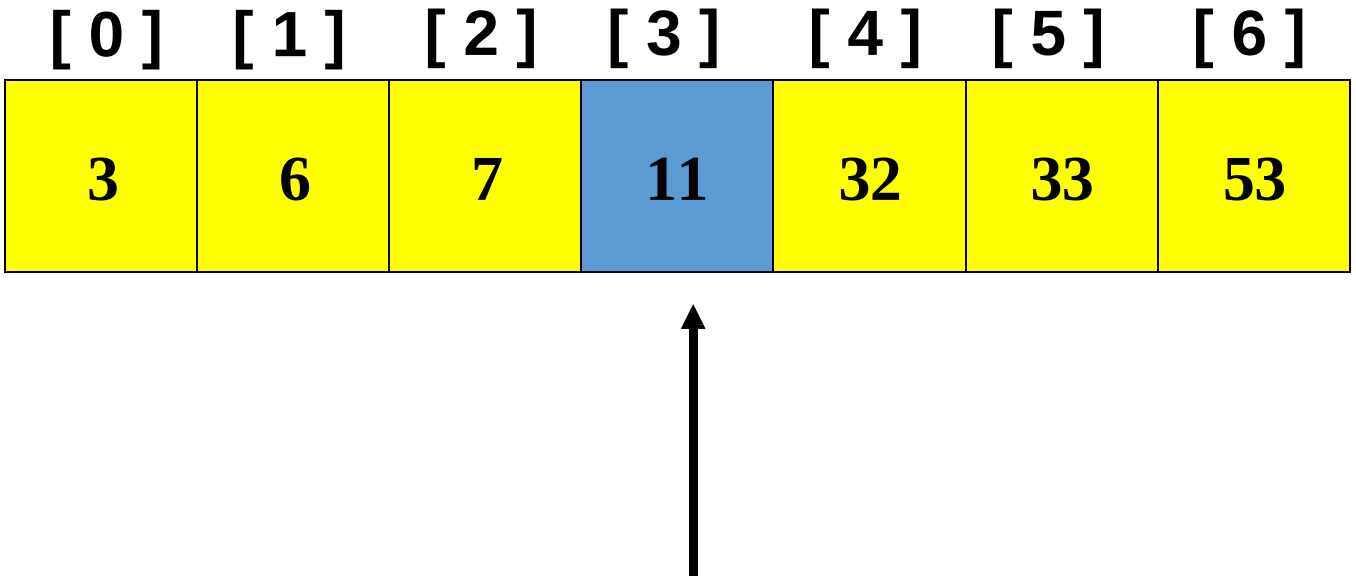
Binary Search: example

Example: sorted array of integer keys. Target=7.



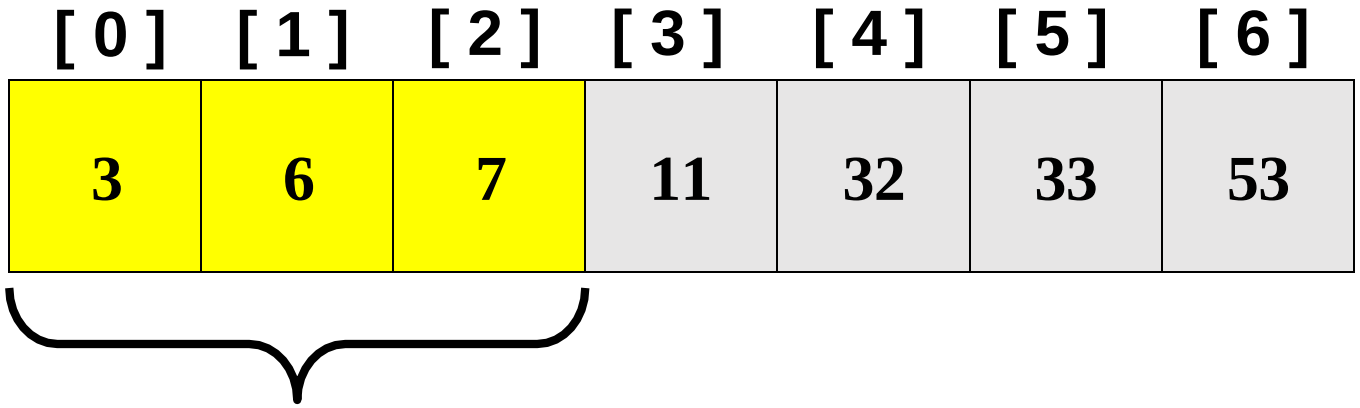
Binary Search: example

Example: sorted array of integer keys. Target=7.



Binary Search: example

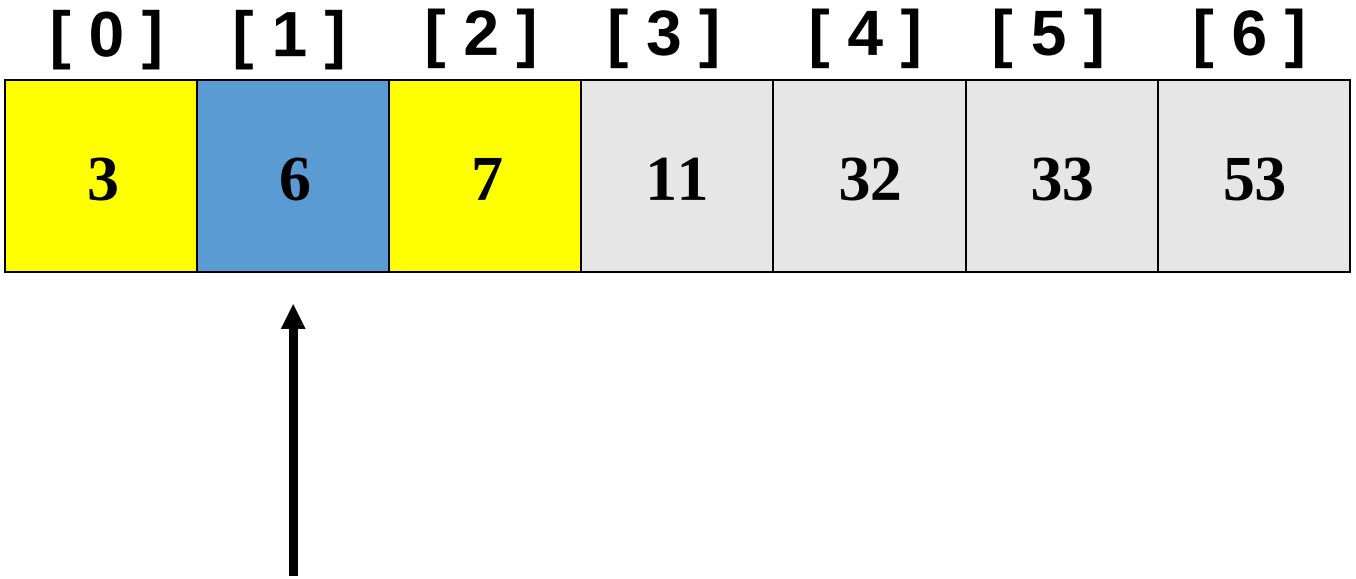
Example: sorted array of integer keys. Target=7.



Search for the target in the area before midpoint.

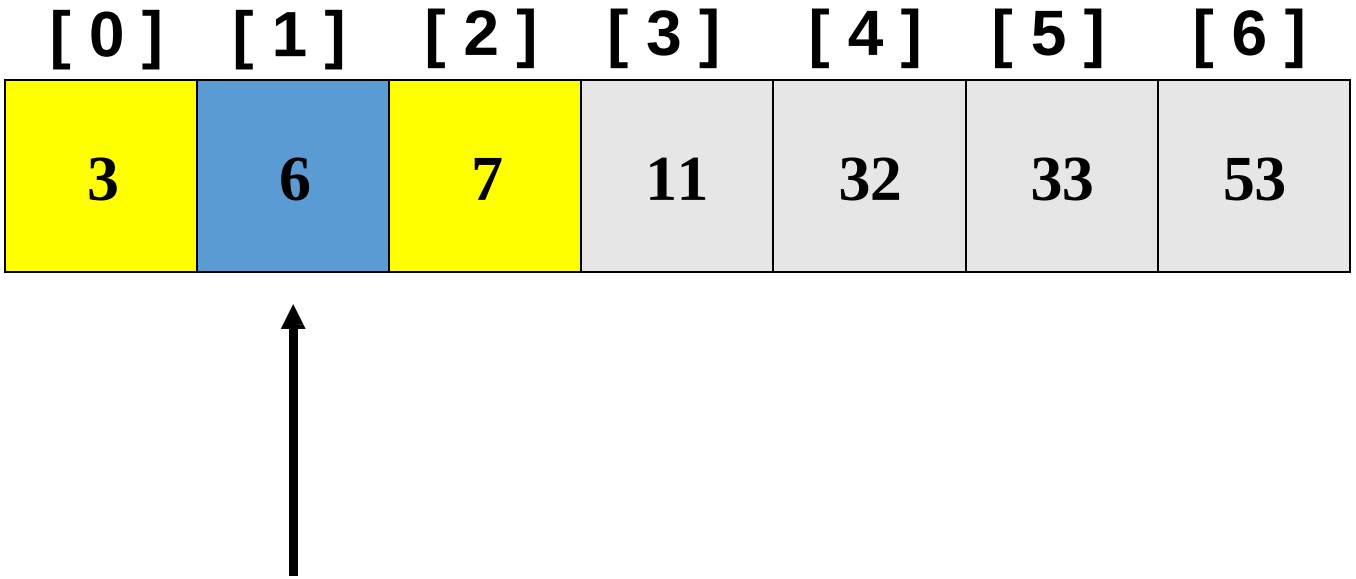
Binary Search: example

Example: sorted array of integer keys. Target=7.



Binary Search: example

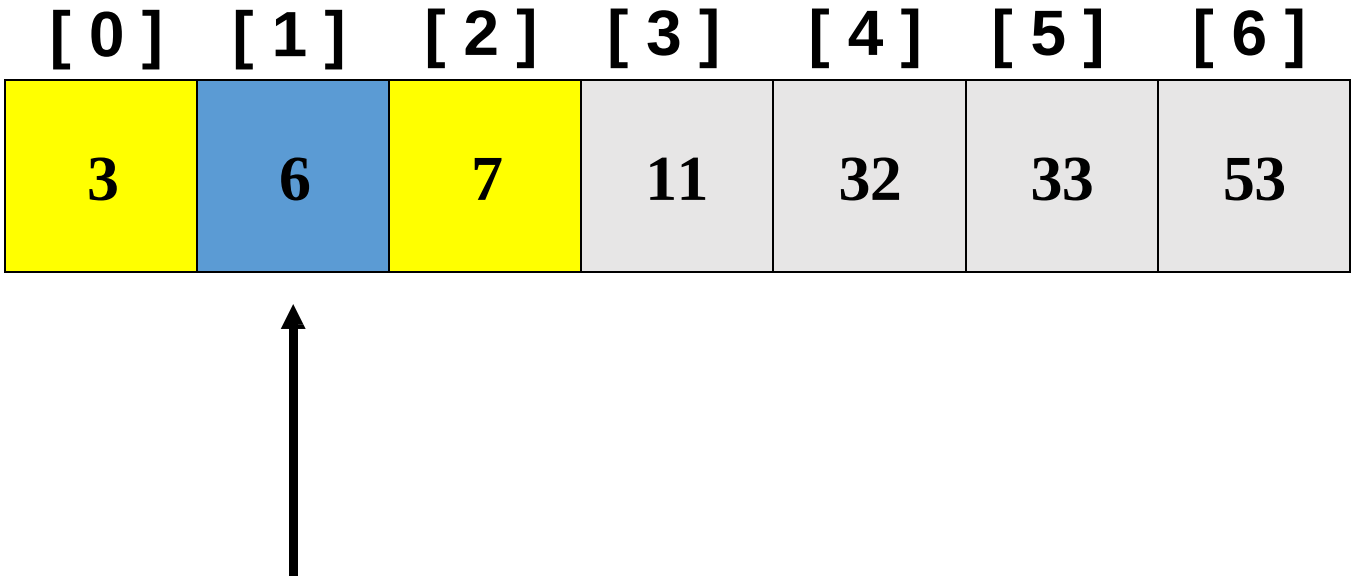
Example: sorted array of integer keys. Target=7.



Target = key of midpoint? NO.

Binary Search: example

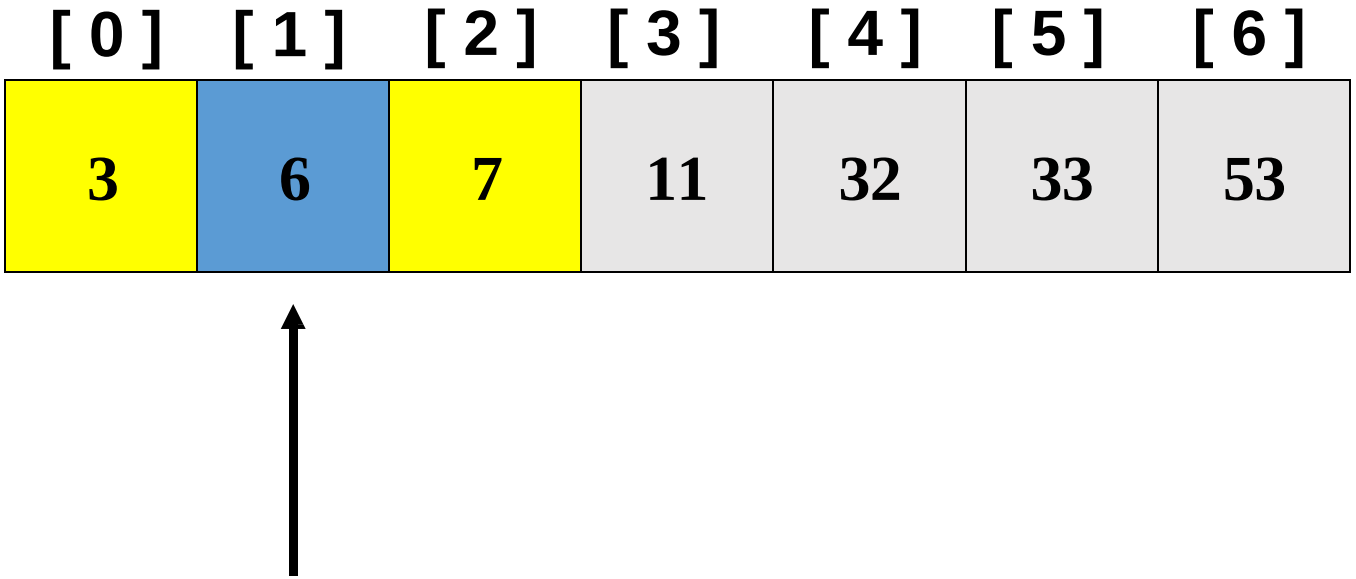
Example: sorted array of integer keys. Target=7.



Target < key of midpoint? NO.

Binary Search: example

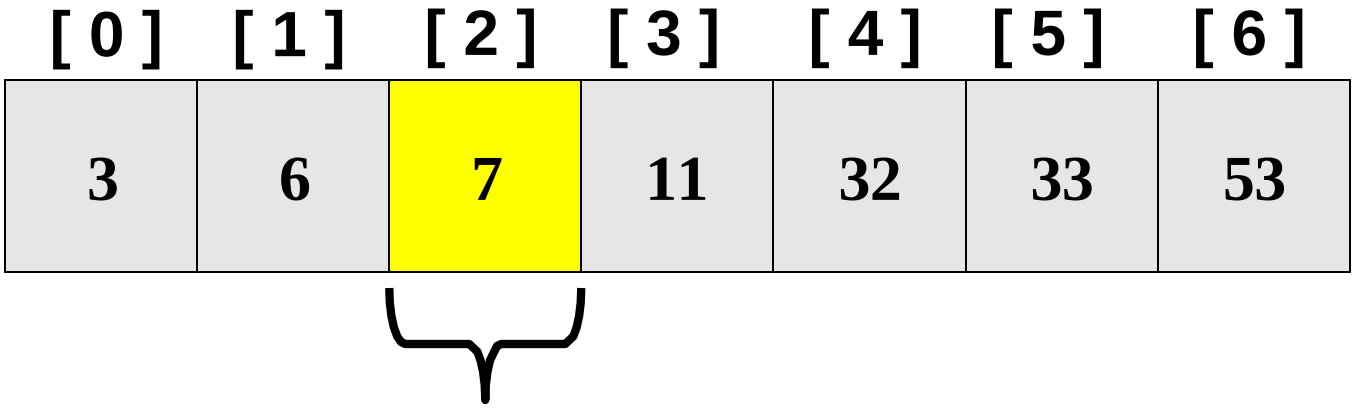
Example: sorted array of integer keys. Target=7.



Target > key of midpoint? YES.

Binary Search: example

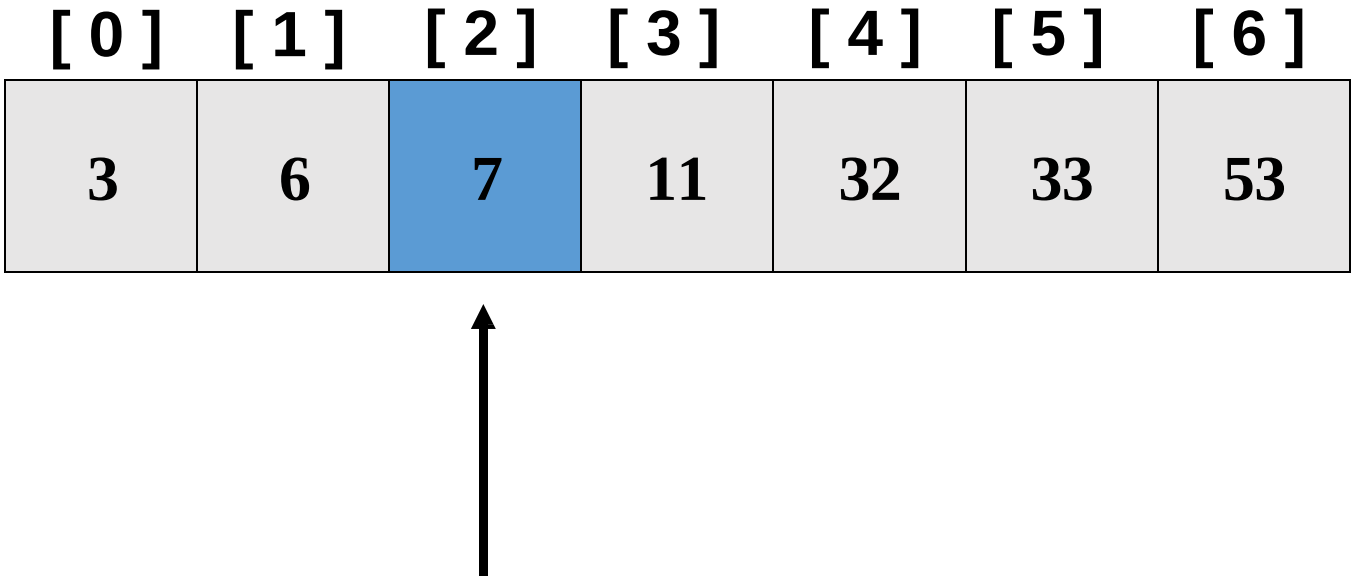
Example: sorted array of integer keys. Target=7.



Search for the target in the area after midpoint.

Binary Search: example

Example: sorted array of integer keys. Target=7.



Find approximate midpoint.
Is target = midpoint key? YES.

Summary

- Linear search on an array takes O(n) time in the worst-case
- If the array is sorted, can perform binary search in O(log n) time
  - Iterative and recursive versions
- Both algorithms run in O(1) time in the best-case
  - Case where the first element inspected is the target element

جامعة شط العرب كلية
العلوم
قسم علوم الحاسبات

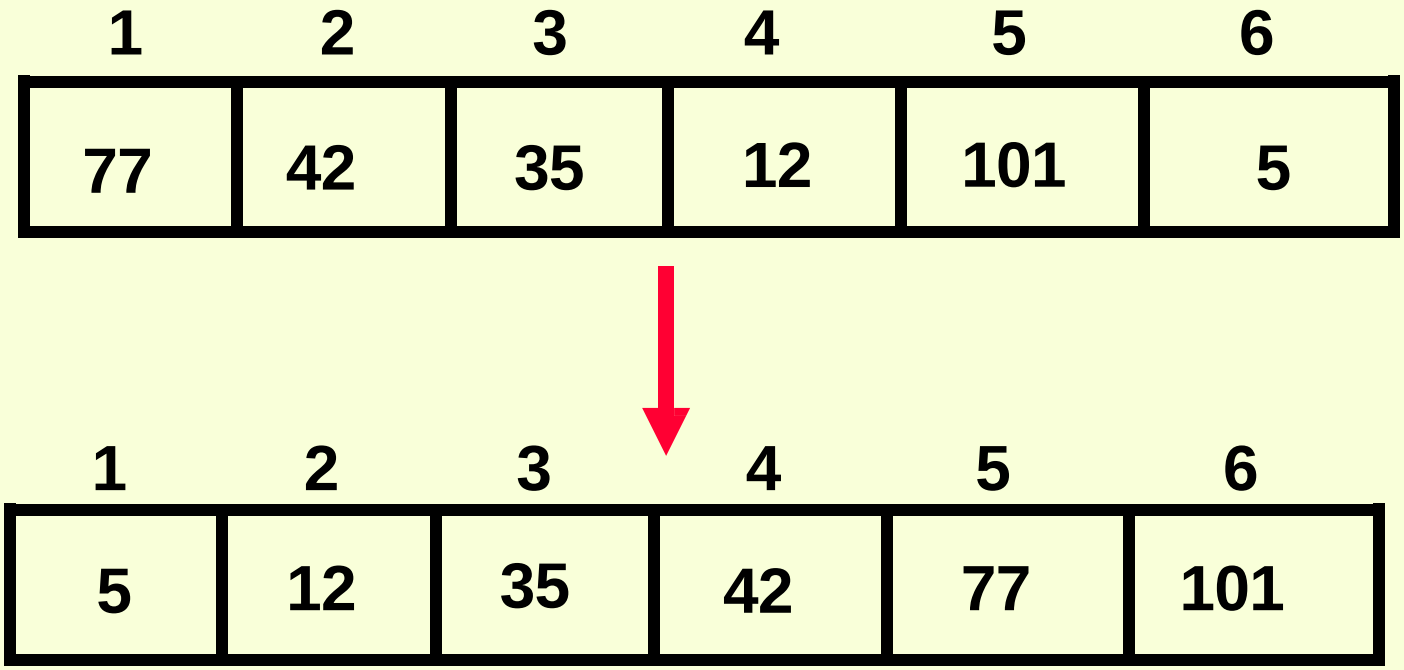
المرحلة الثالثة
مادة هياكل بيانات2
أستاذ المادة: م.م مصطفى صبحي سلمان

Lecture 1

Bubble Sort

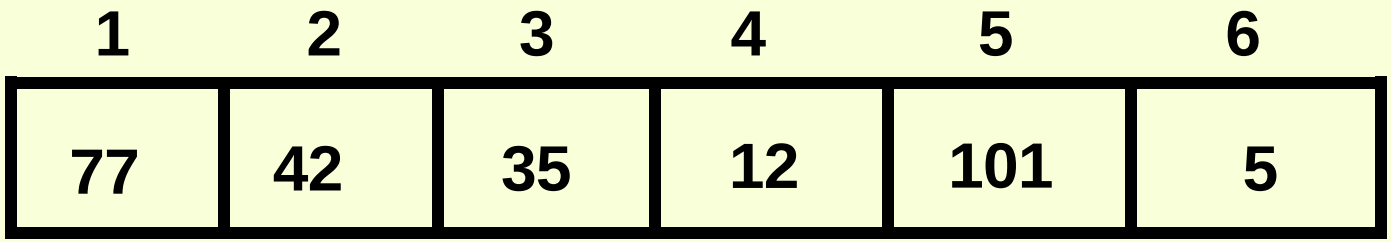
Sorting

- **Sorting takes an unordered collection and makes it an ordered one.**



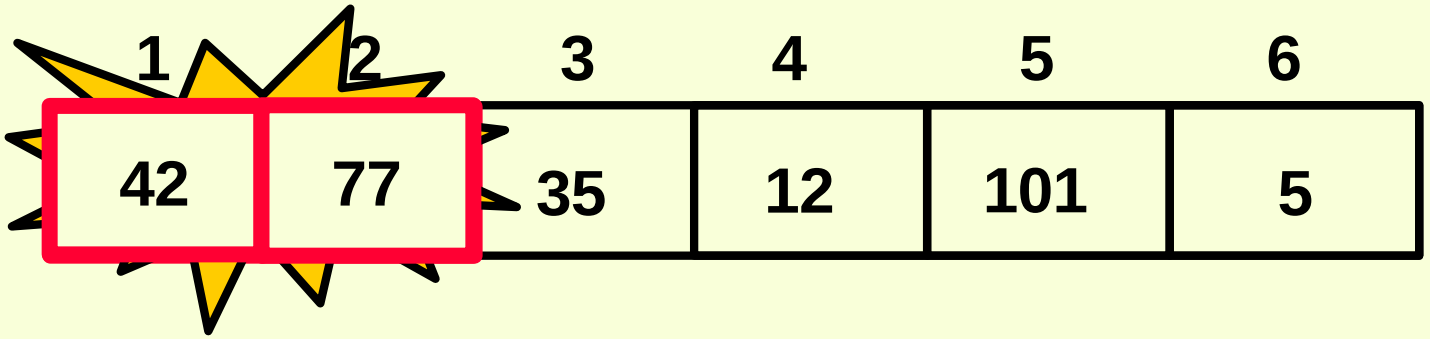
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" the largest value to the end using pair-wise comparisons and swapping



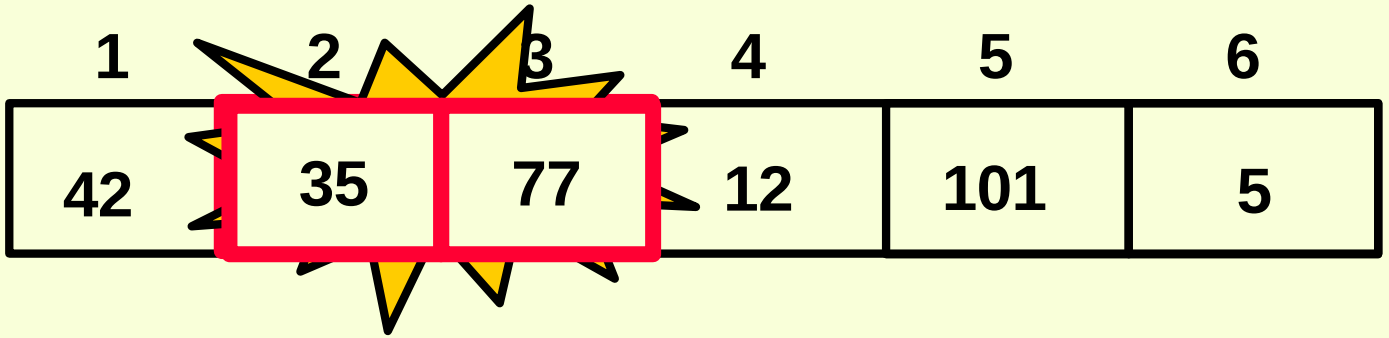
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



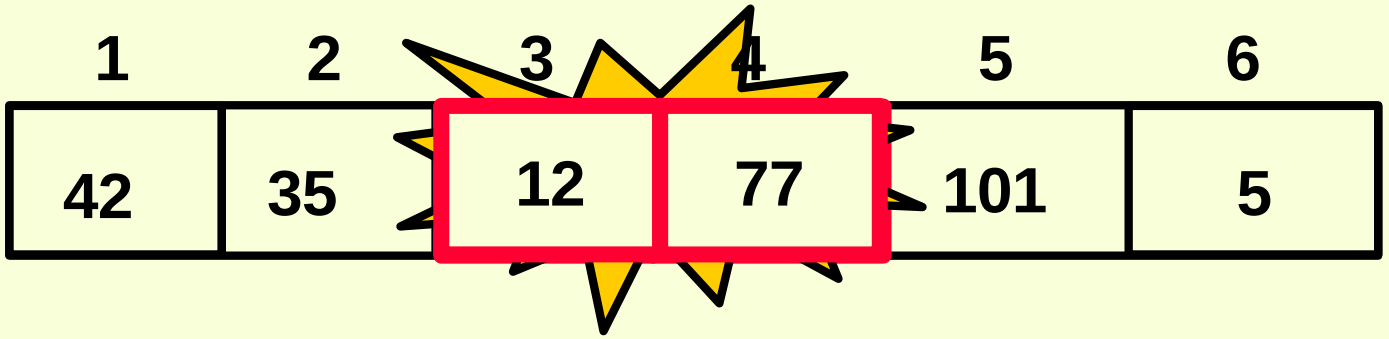
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



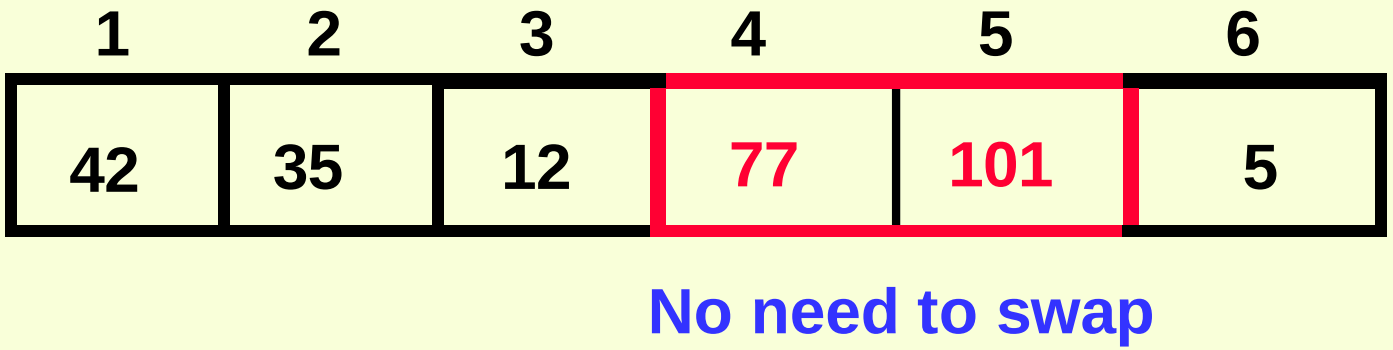
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



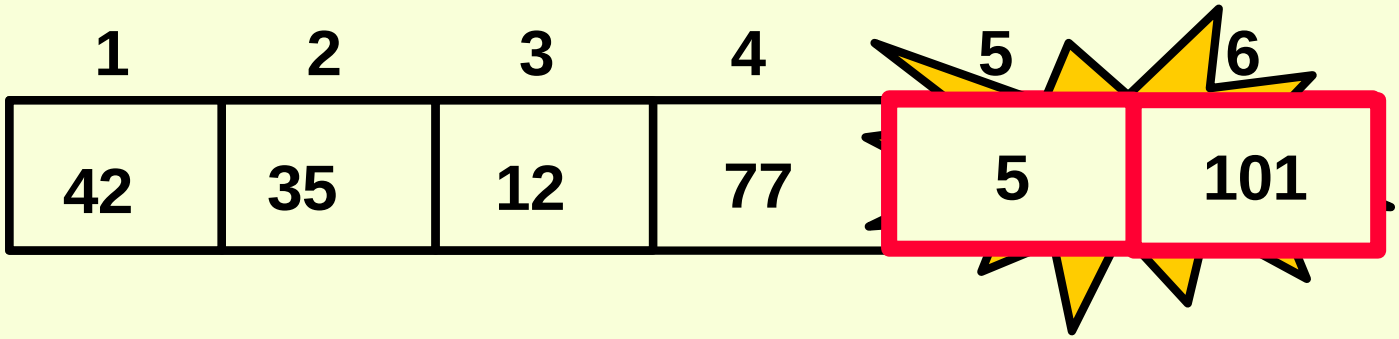
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



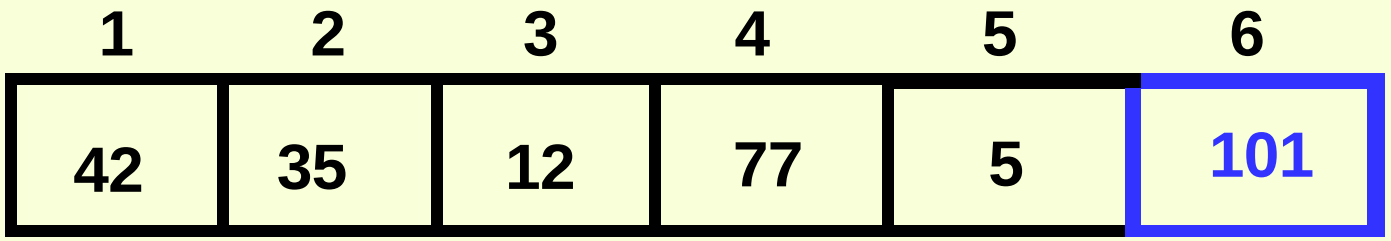
"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



"Bubbling Up" the Largest Element

- Traverse a collection of elements
  - Move from the front to the end
  - “Bubble” the largest value to the end using pair-wise comparisons and swapping



Largest value correctly placed

The “Bubble Up” Algorithm

```
index <- 1
last_compare_at <- n - 1

loop
  exitif(index > last_compare_at)
  if(A[index] > A[index + 1]) then
    Swap(A[index], A[index + 1])
  endif
  index <- index + 1
endloop
```

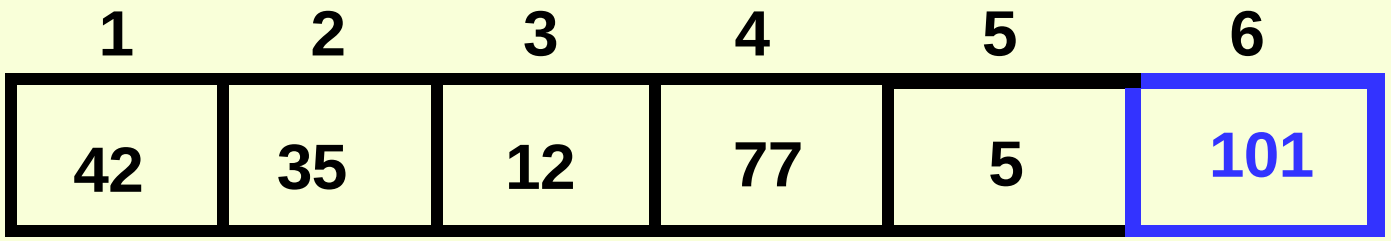
LB

No, Swap isn't built in.

```
Procedure Swap(a, b isoftype in/out Num)
   t isoftype Num
   t <- a
   a <- b
   b <- t
endprocedure // Swap
```

Items of Interest

- Notice that only the largest value is correctly placed
- All other values are still out of order
- So we need to repeat this process

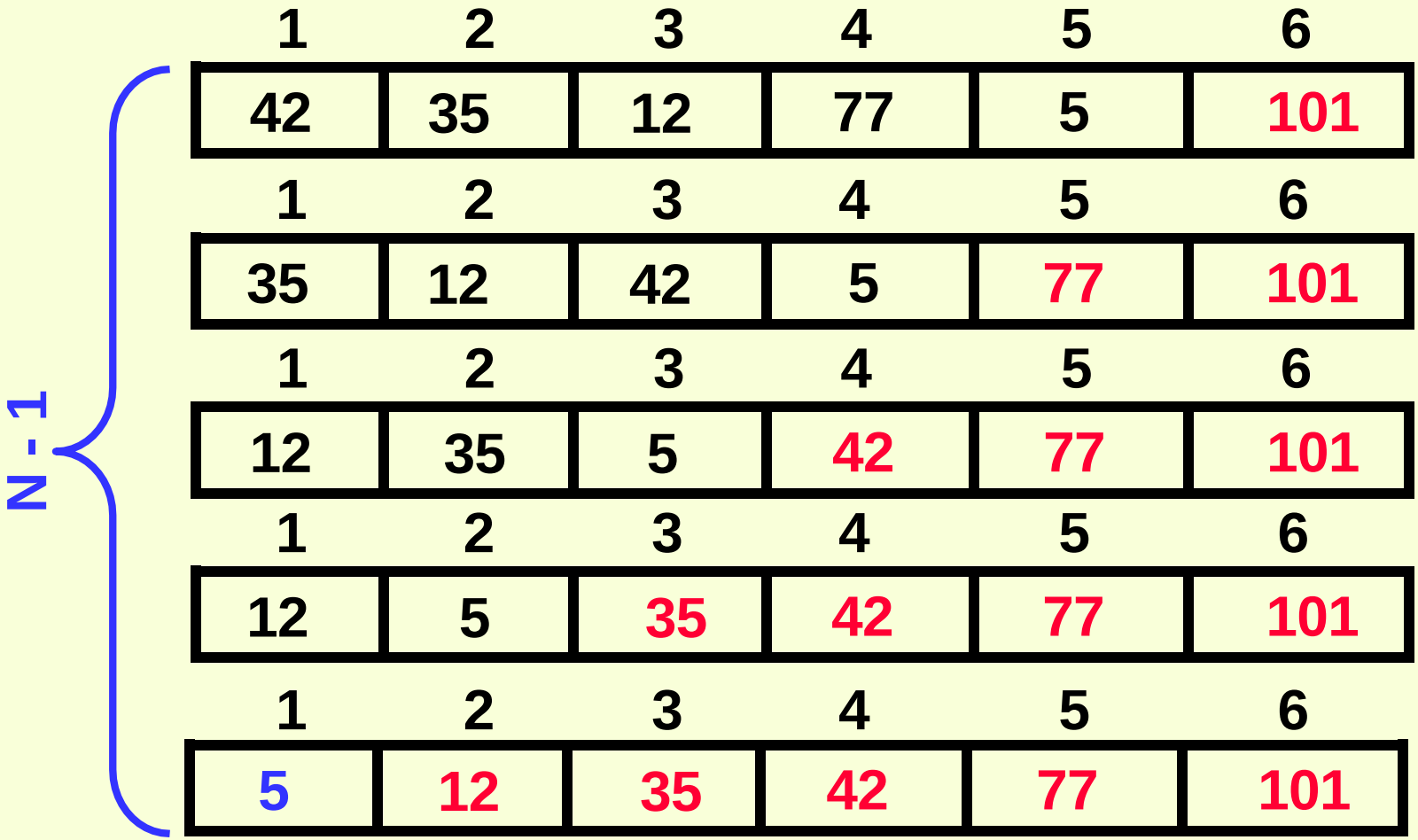


Largest value correctly placed

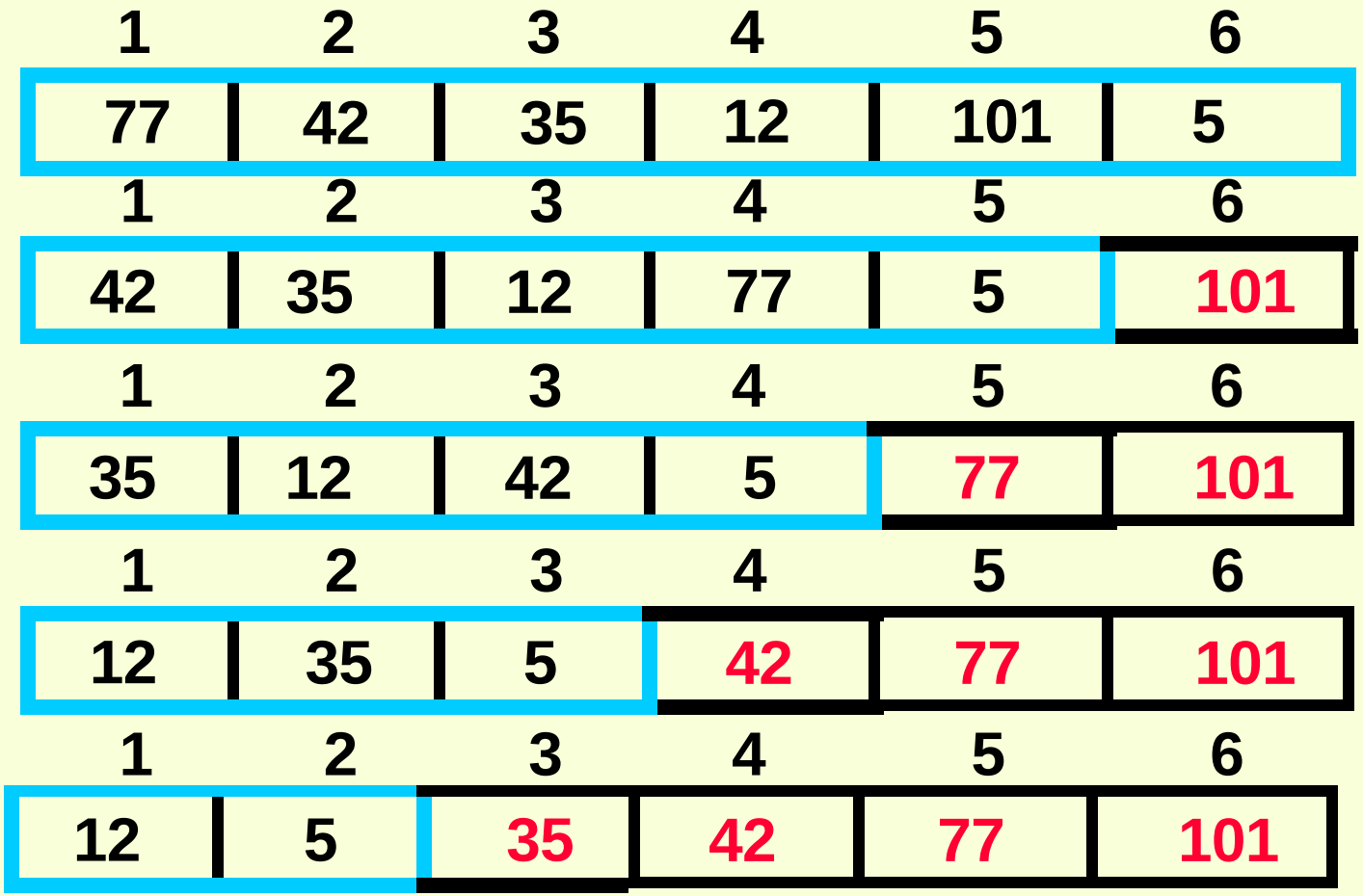
Repeat “Bubble Up” How Many Times?

- If we have N elements…
- And if each time we bubble an element, we place it in its correct location…
- Then we repeat the “bubble up” process N – 1 times.
- This guarantees we’ll correctly place all N elements.

“Bubbling” All the Elements



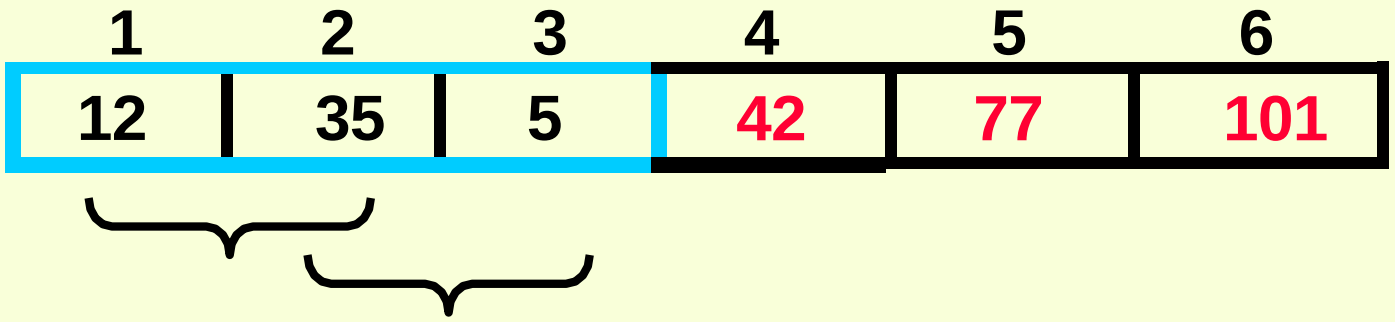
Reducing the Number of Comparisons



Reducing the Number of Comparisons

- **On the Nth "bubble up", we only need to do MAX-N comparisons.**

- **For example:**
  - **This is the 4th "bubble up"**
  - **MAX is 6**
  - **Thus we have 2 comparisons to do**



Putting It All Together

```
N is … // Size of Array

Arr_Type definesa Array[1..N] of Num

Procedure Swap(n1, n2 isoftype in/out Num)
  temp isoftype Num
  temp <- n1
  n1   <- n2
  n2   <- temp
endprocedure // Swap
```

```
procedure Bubblesort(A isoftype in/out Arr_Type)
  to_do, index isoftype Num
  to_do <- N - 1

  loop
    exitif(to_do = 0)
    index <- 1
    loop
      exitif(index > to_do)
      if(A[index] > A[index + 1]) then
        Swap(A[index], A[index + 1])
      endif
      index <- index + 1
    endloop
    to_do <- to_do - 1
  endloop
endprocedure // Bubblesort
```

Already Sorted Collections?

- What if the collection was already sorted?
- What if only a few elements were out of place and after a couple of “bubble ups,” the collection was sorted?

- We want to be able to detect this and “stop early”!

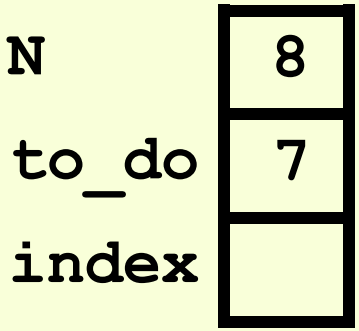
1	2	3	4	5	6
5	12	35	42	77	101

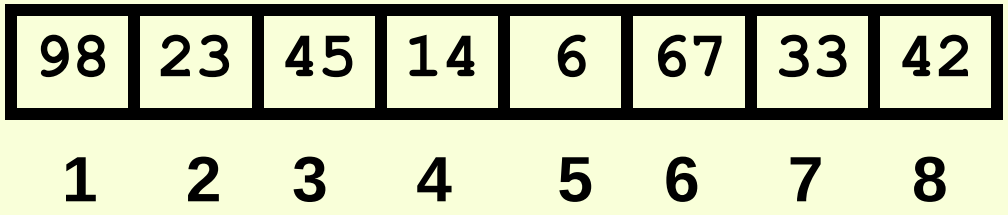
Using a Boolean “Flag”

- We can use a boolean variable to determine if any swapping occurred during the “bubble up.”
- If no swapping occurred, then we know that the collection is already sorted!
- This boolean “flag” needs to be reset after each “bubble up.”

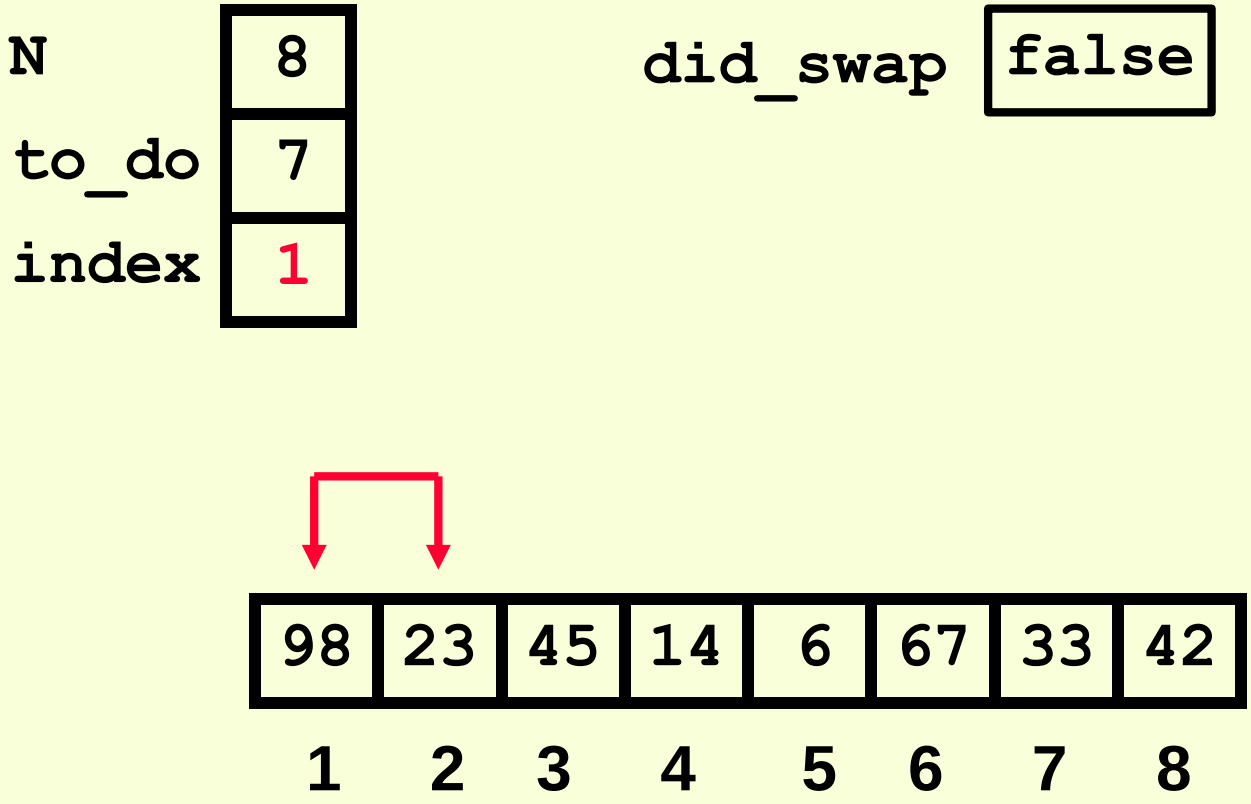
```
did_swap isoftype Boolean
did_swap <- true

loop
  exitif ((to_do = 0) OR NOT(did_swap))
  index <- 1
  did_swap <- false
  loop
    exitif(index > to_do)
    if(A[index] > A[index + 1]) then
      Swap(A[index], A[index + 1])
      did_swap <- true
    endif
    index <- index + 1
  endloop
  to_do <- to_do - 1
endloop
```

An Animated Example

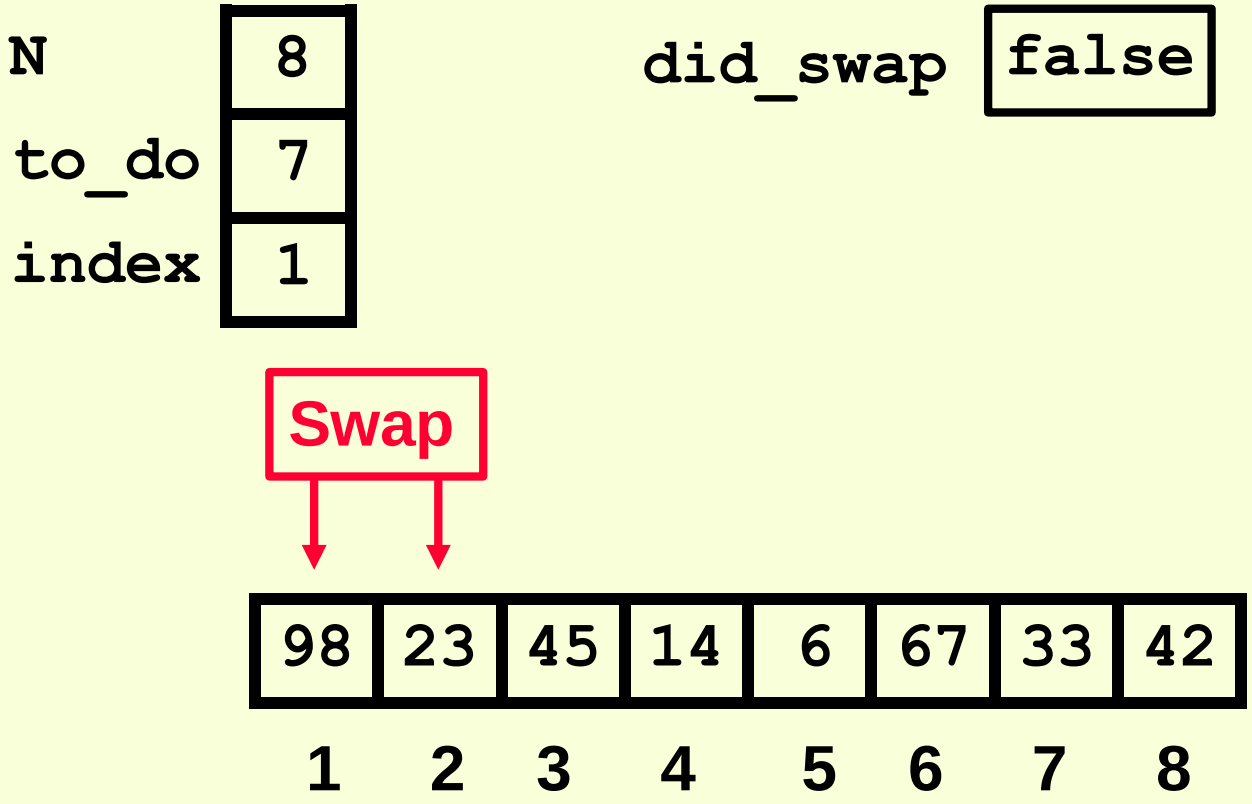




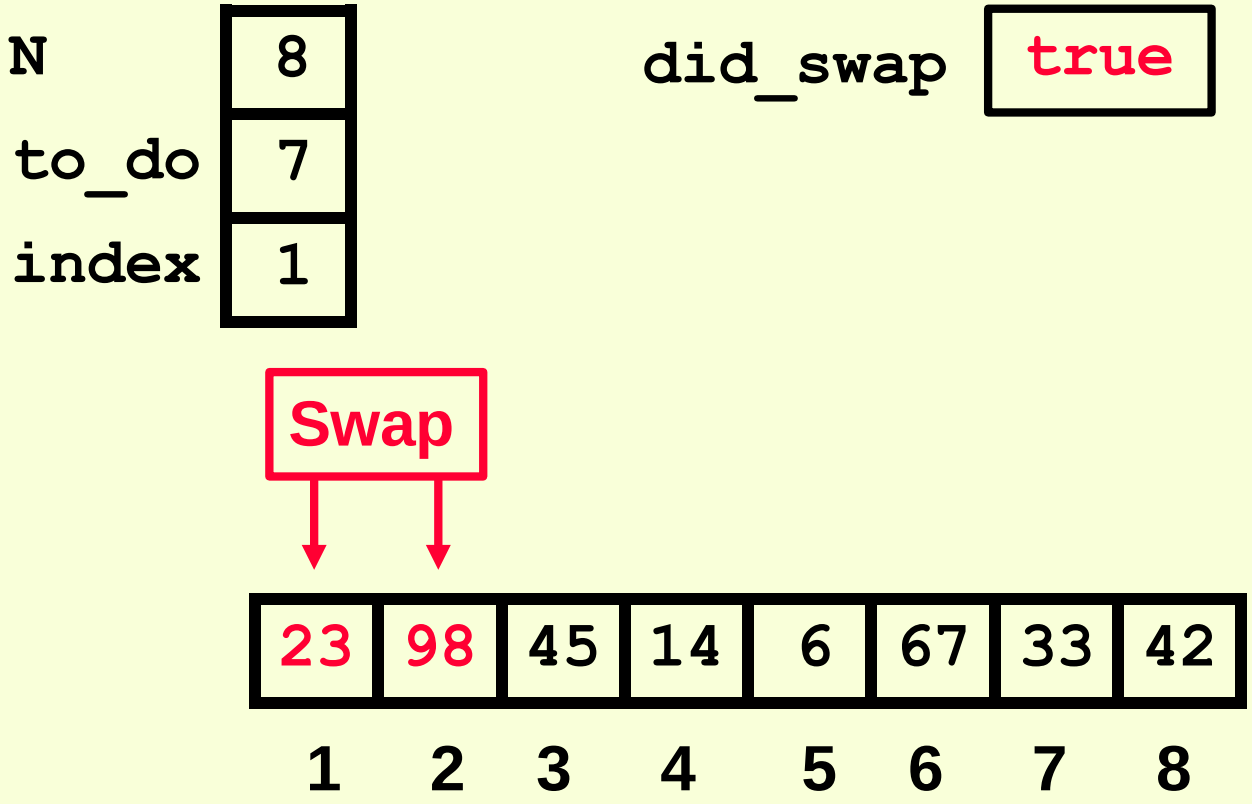
An Animated Example



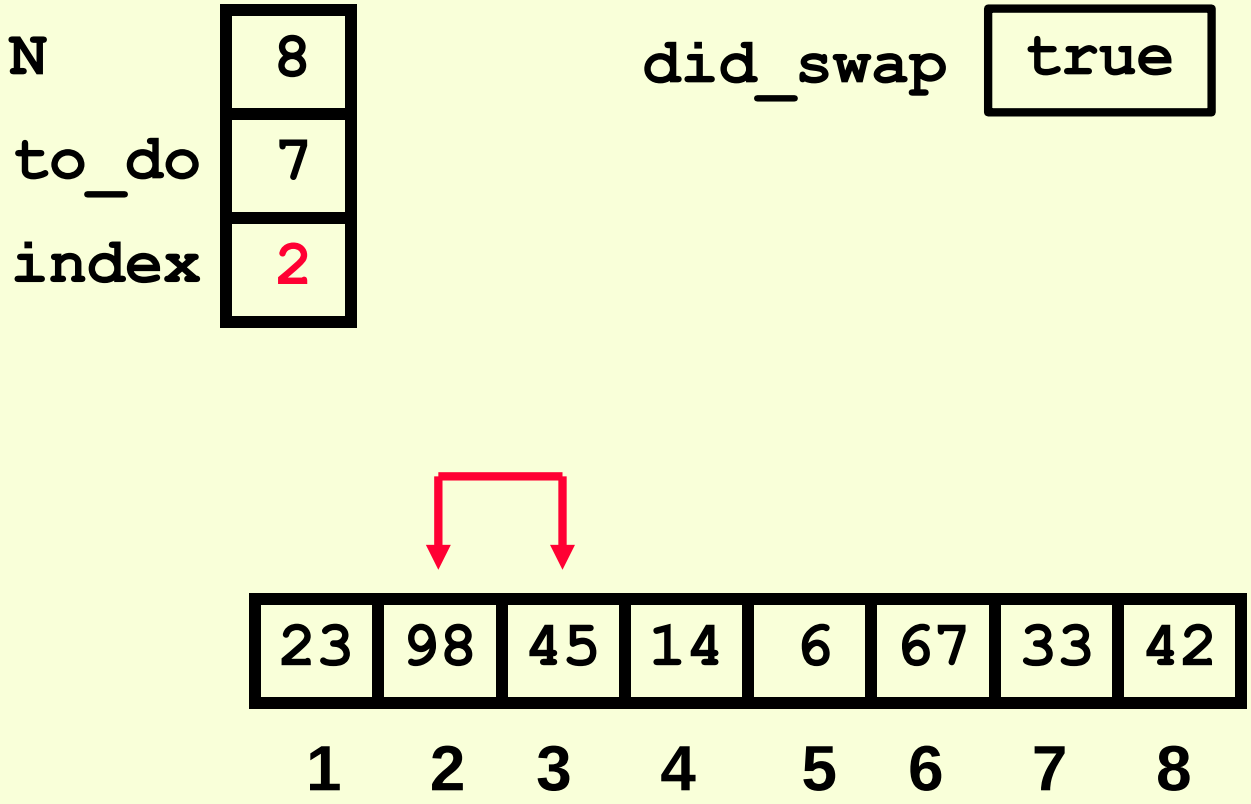
An Animated Example



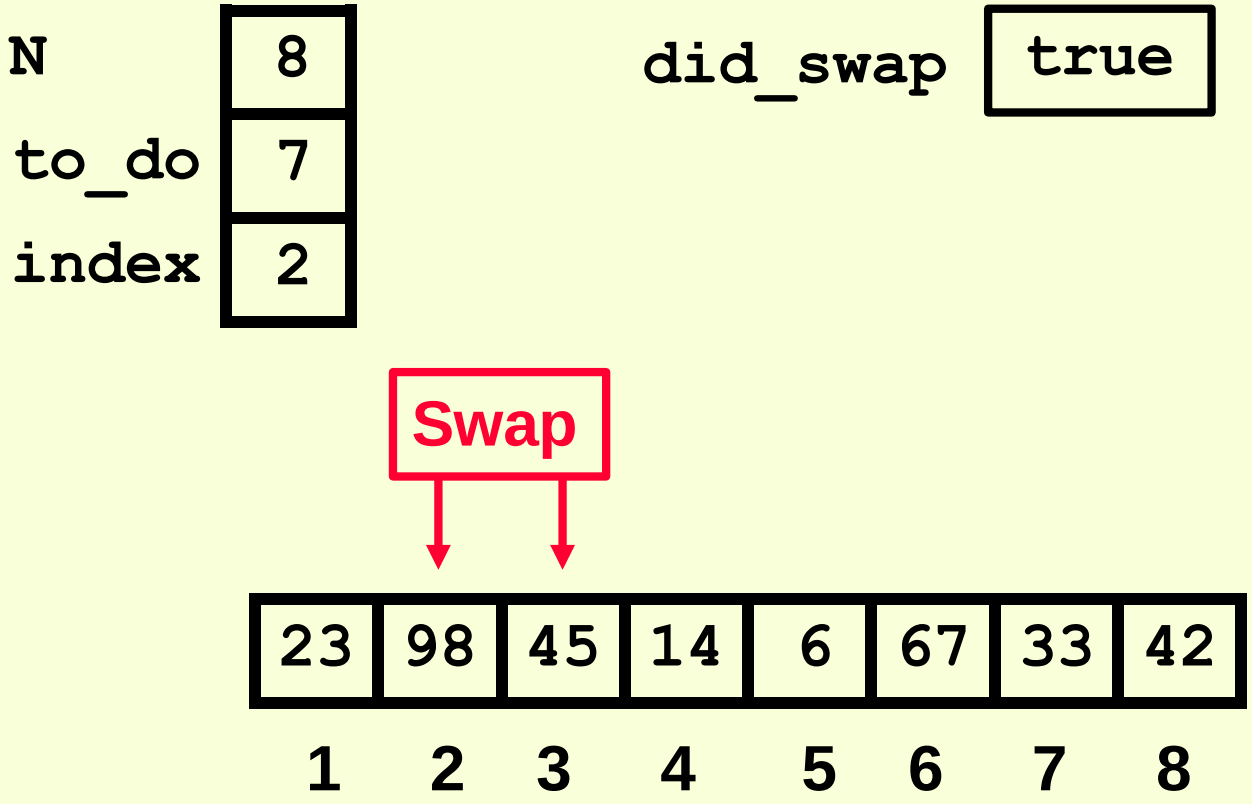
An Animated Example



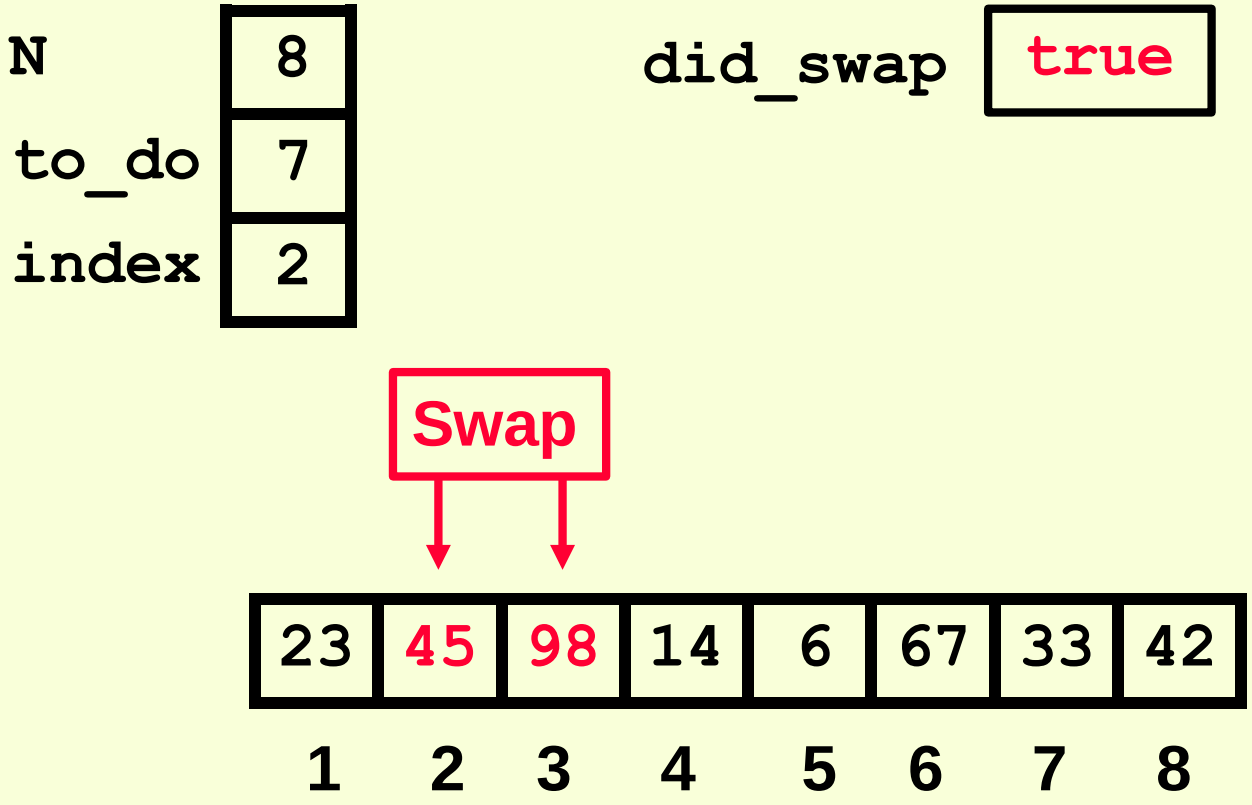
An Animated Example



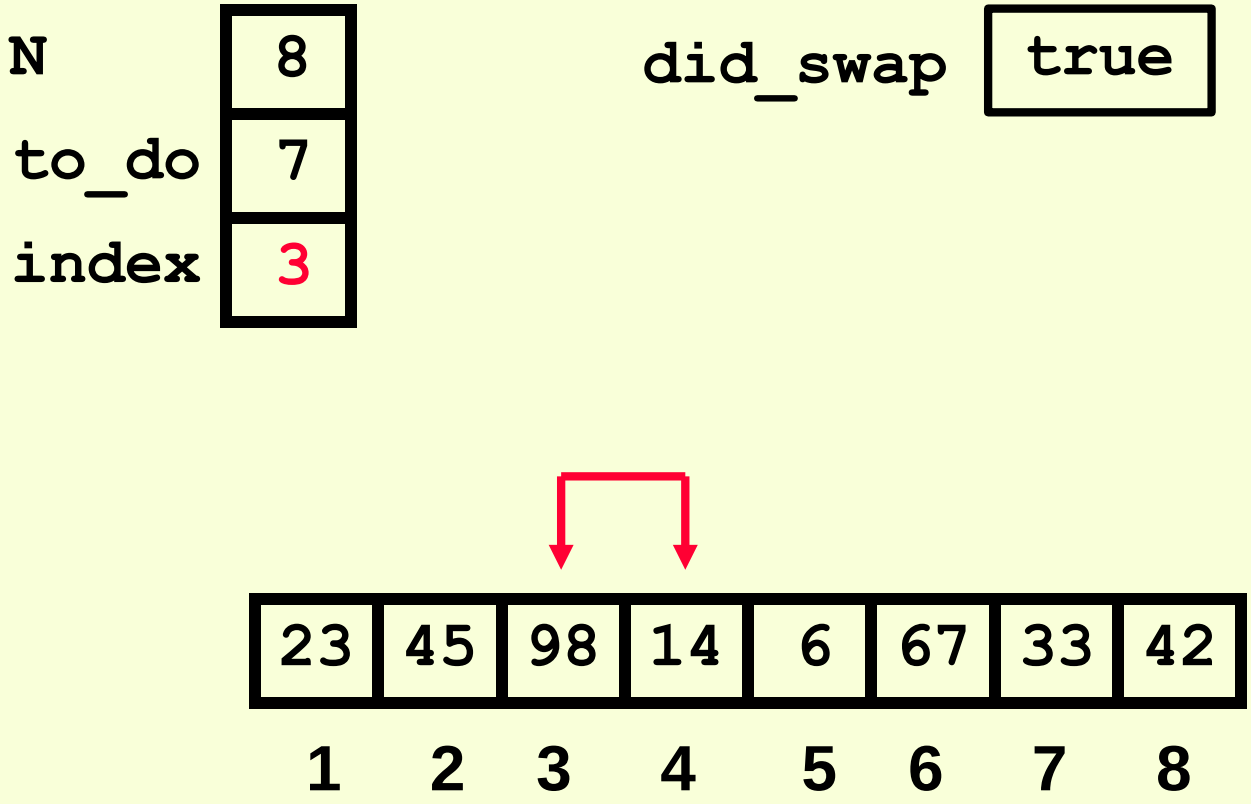
An Animated Example



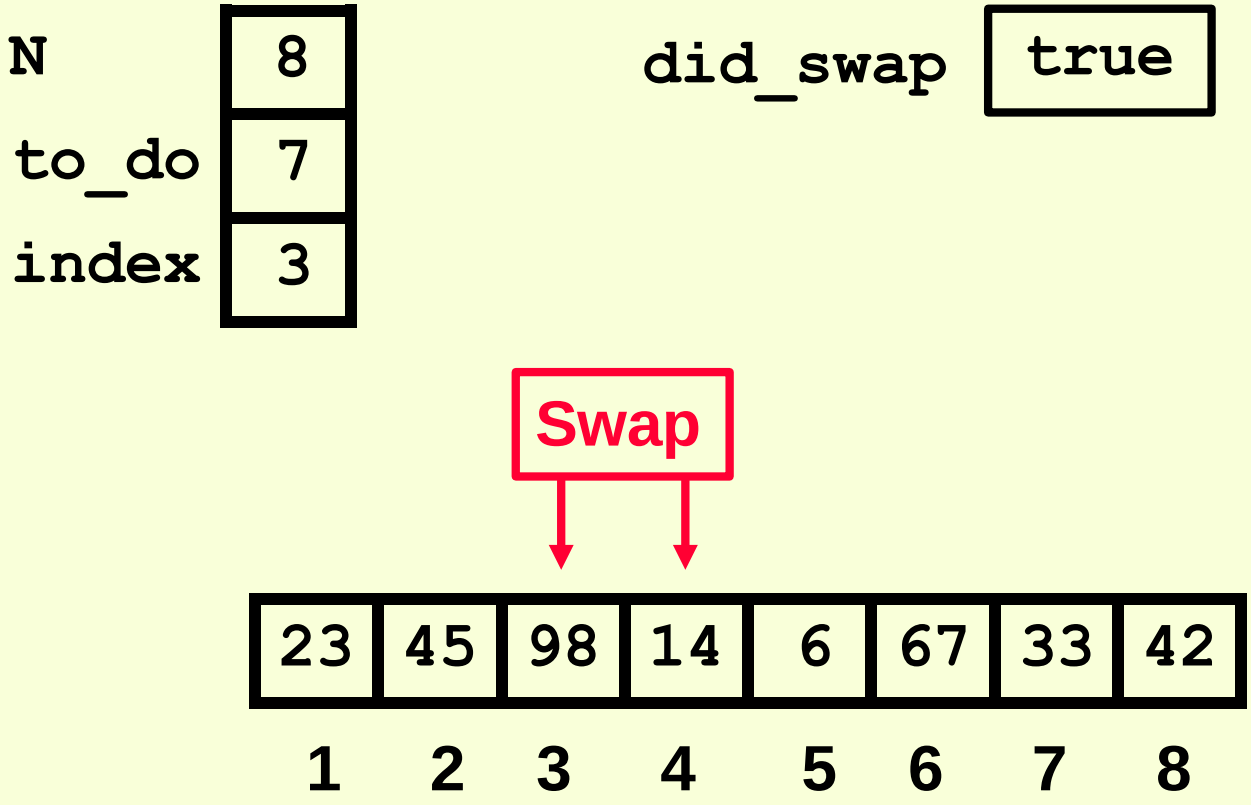
An Animated Example



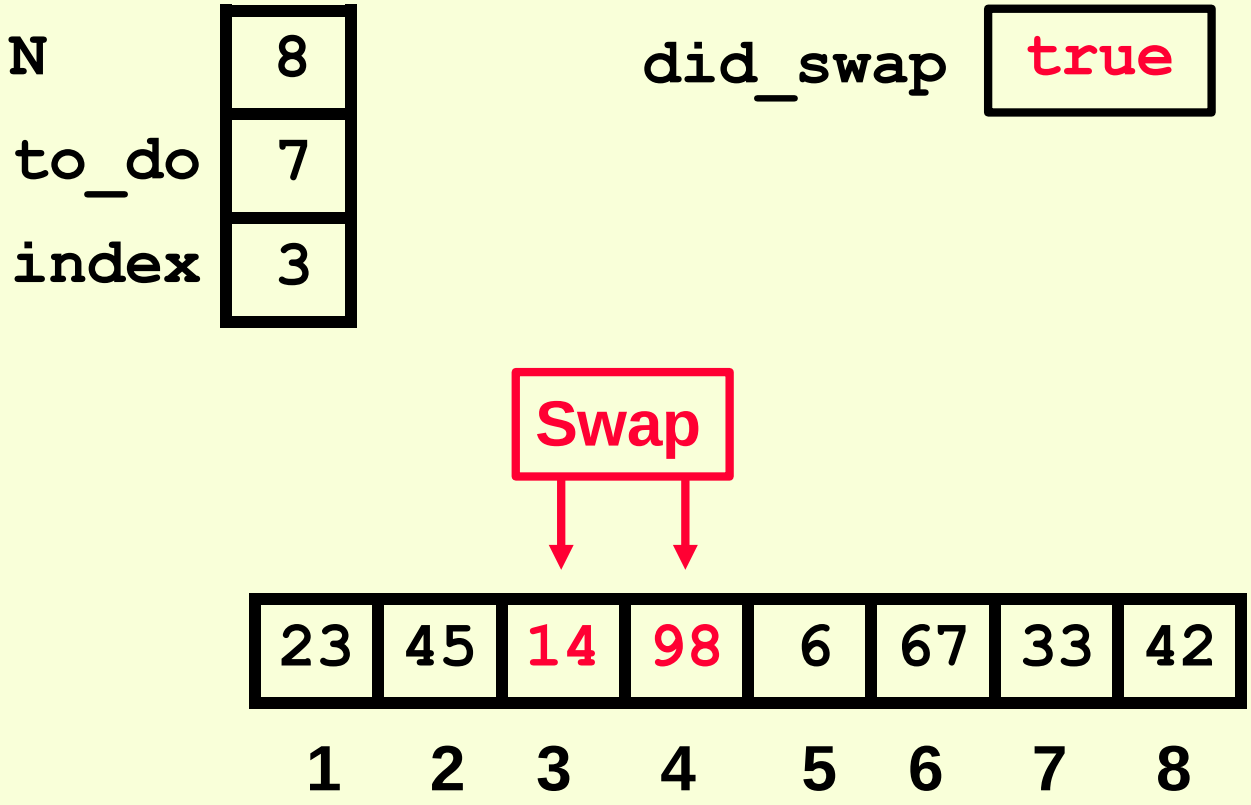
An Animated Example



An Animated Example



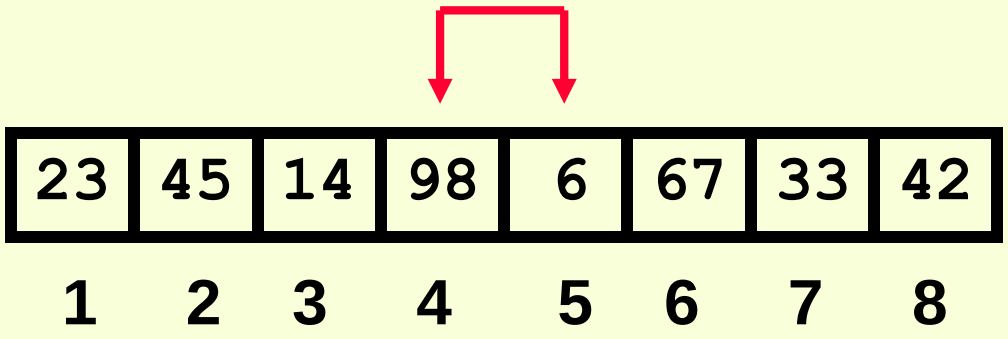
An Animated Example



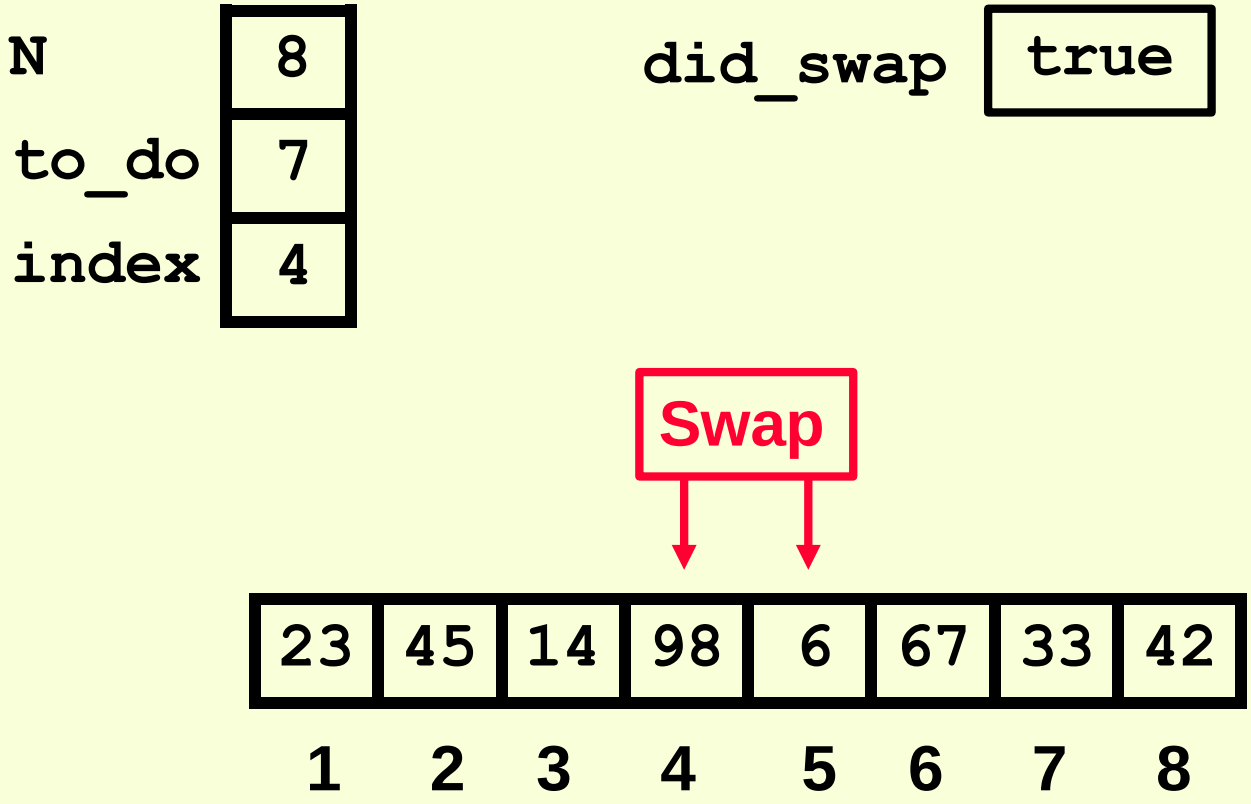
An Animated Example

N	8
to_do	7
index	4

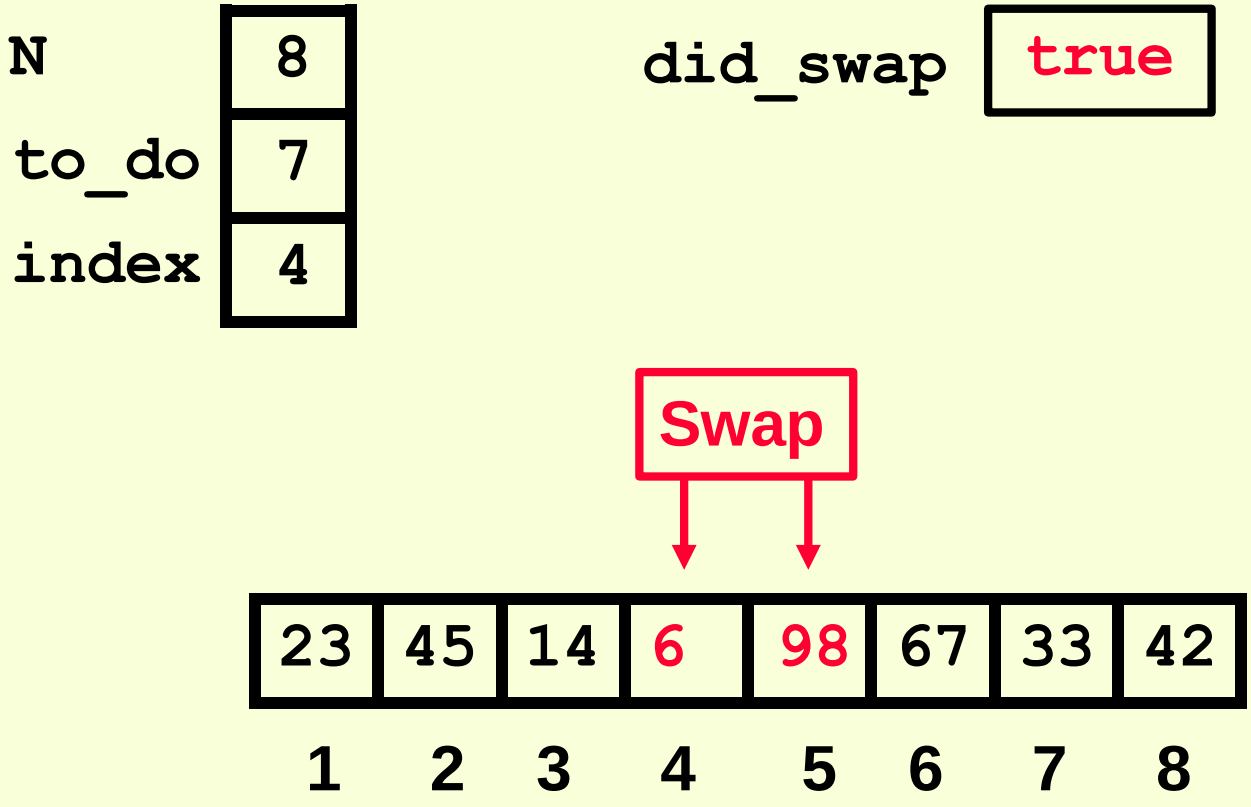
did_swap	true



An Animated Example



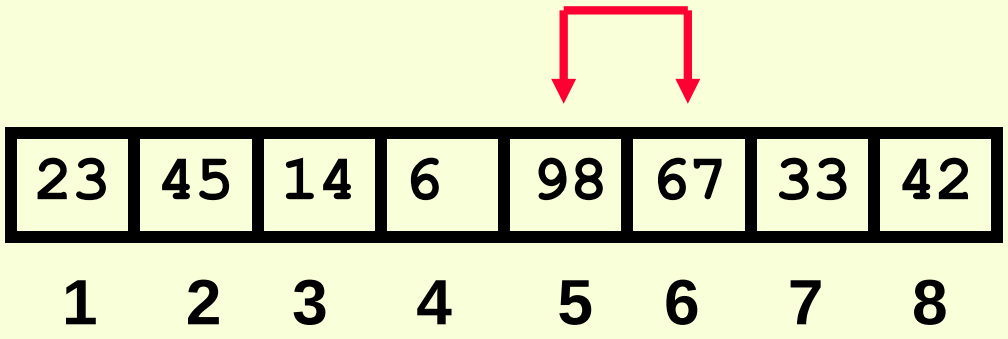
An Animated Example



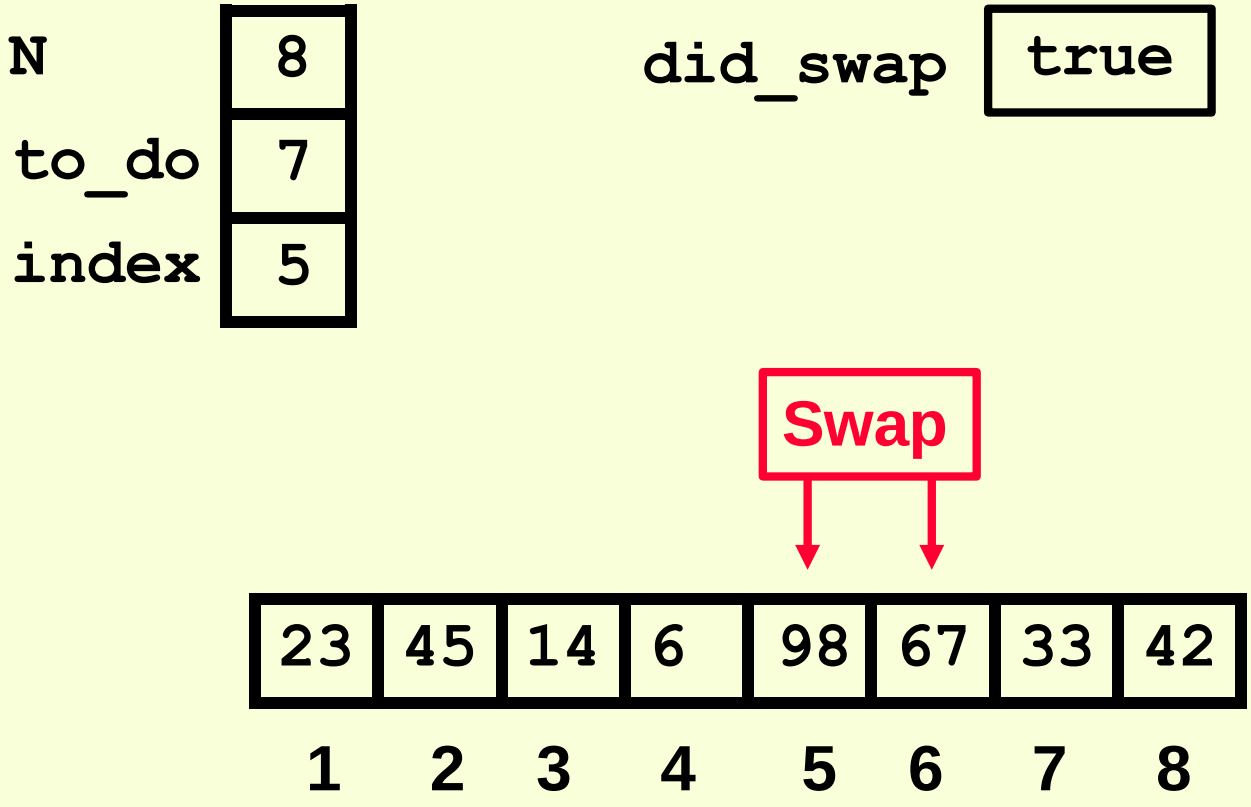
An Animated Example

N	8
to_do	7
index	5

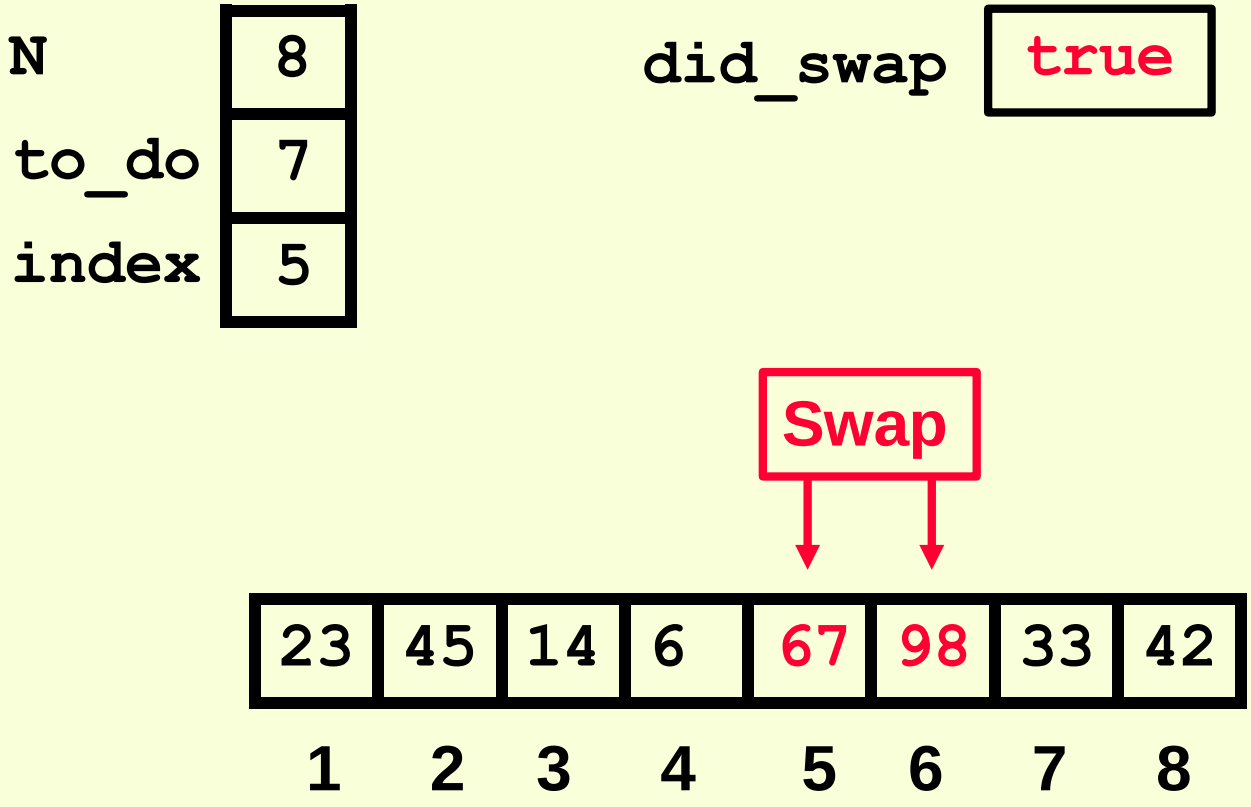
did_swap	true



An Animated Example



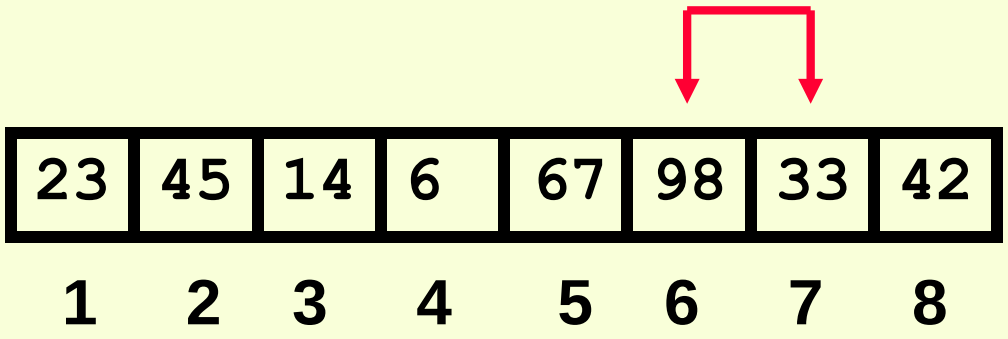
An Animated Example



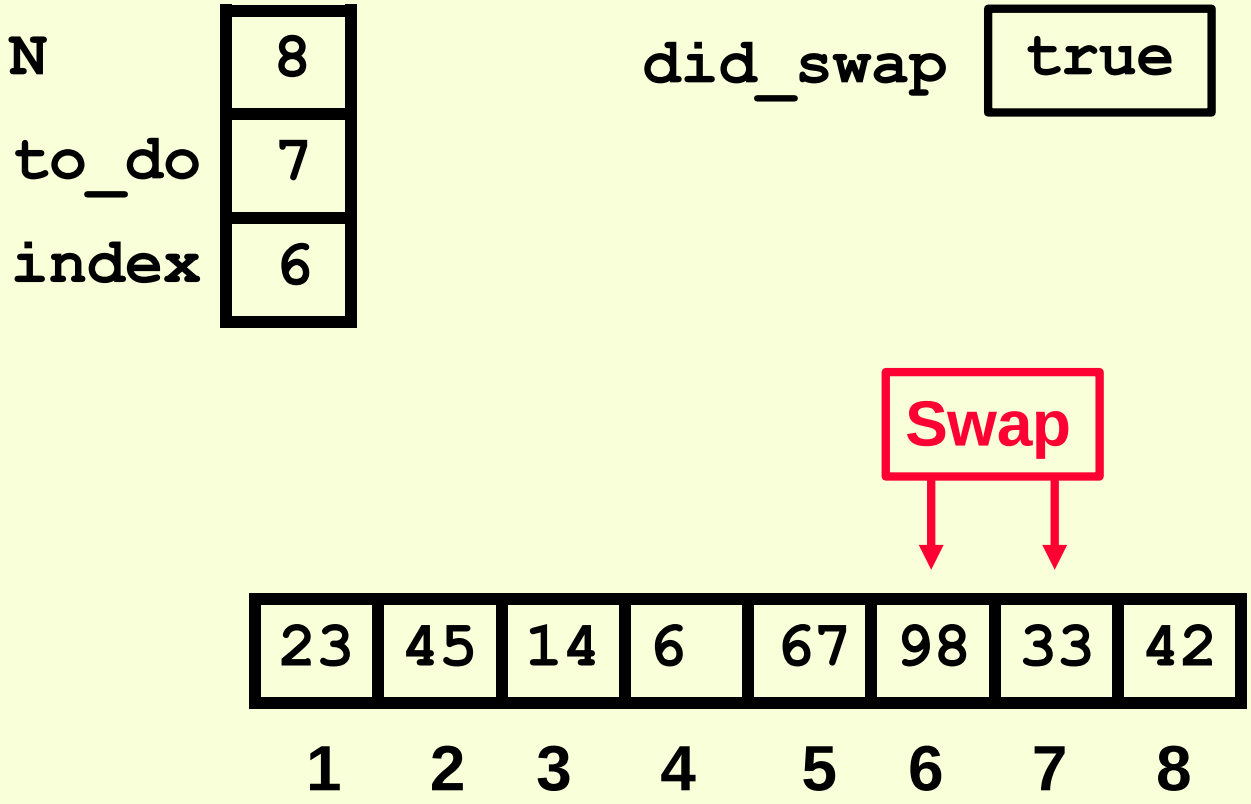
An Animated Example

N	8
to_do	7
index	6

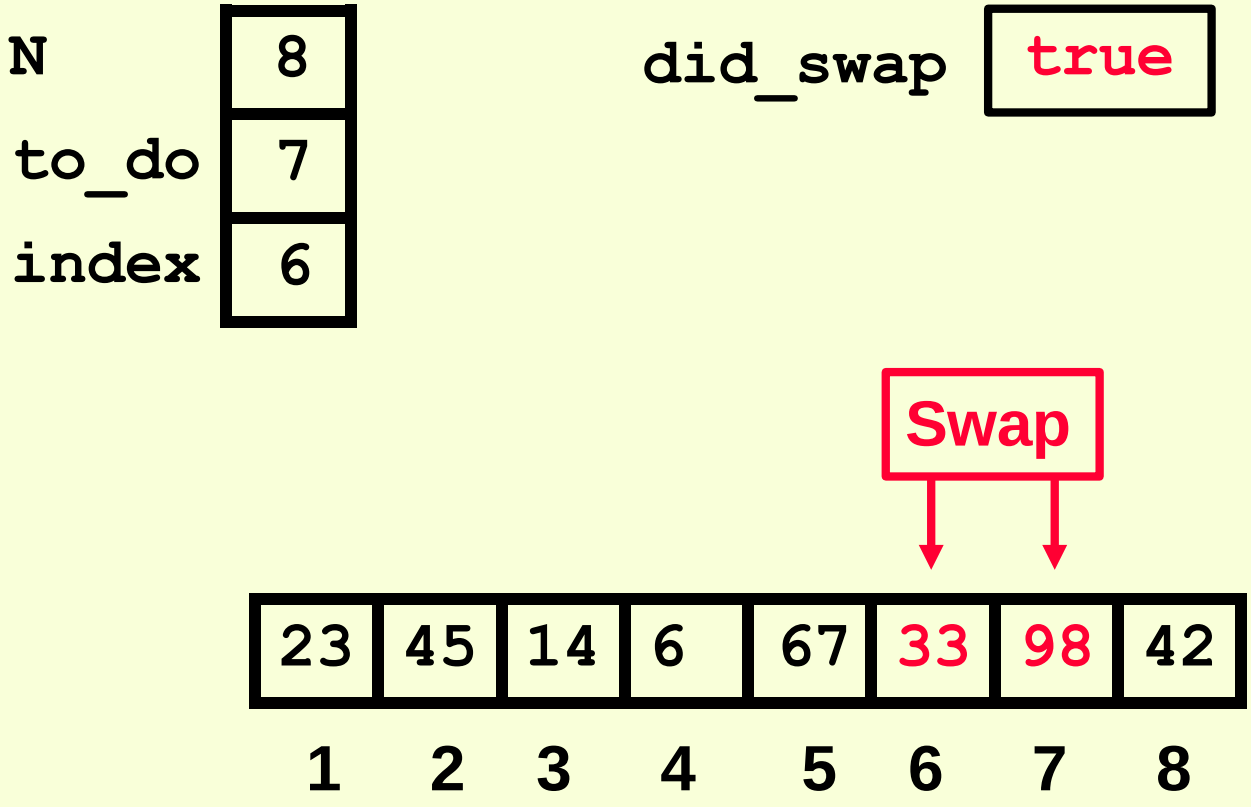
did_swap	true



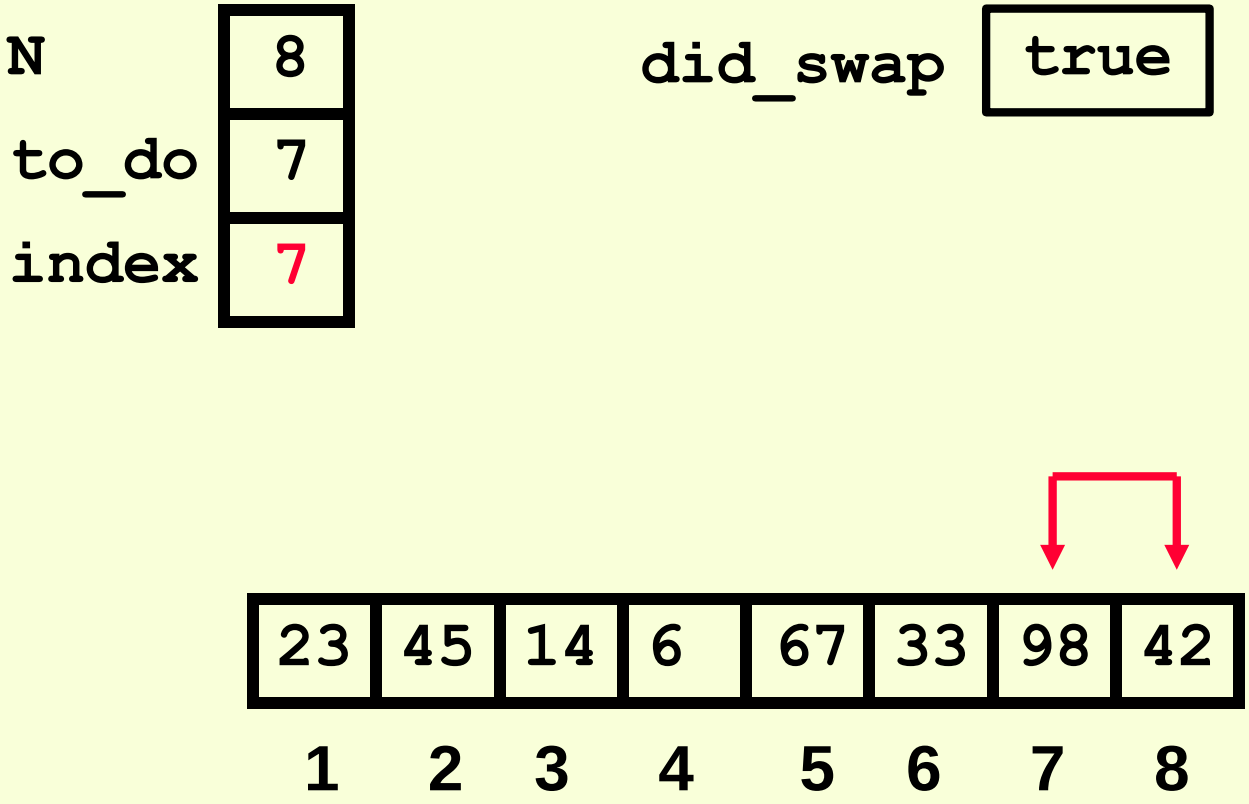
An Animated Example



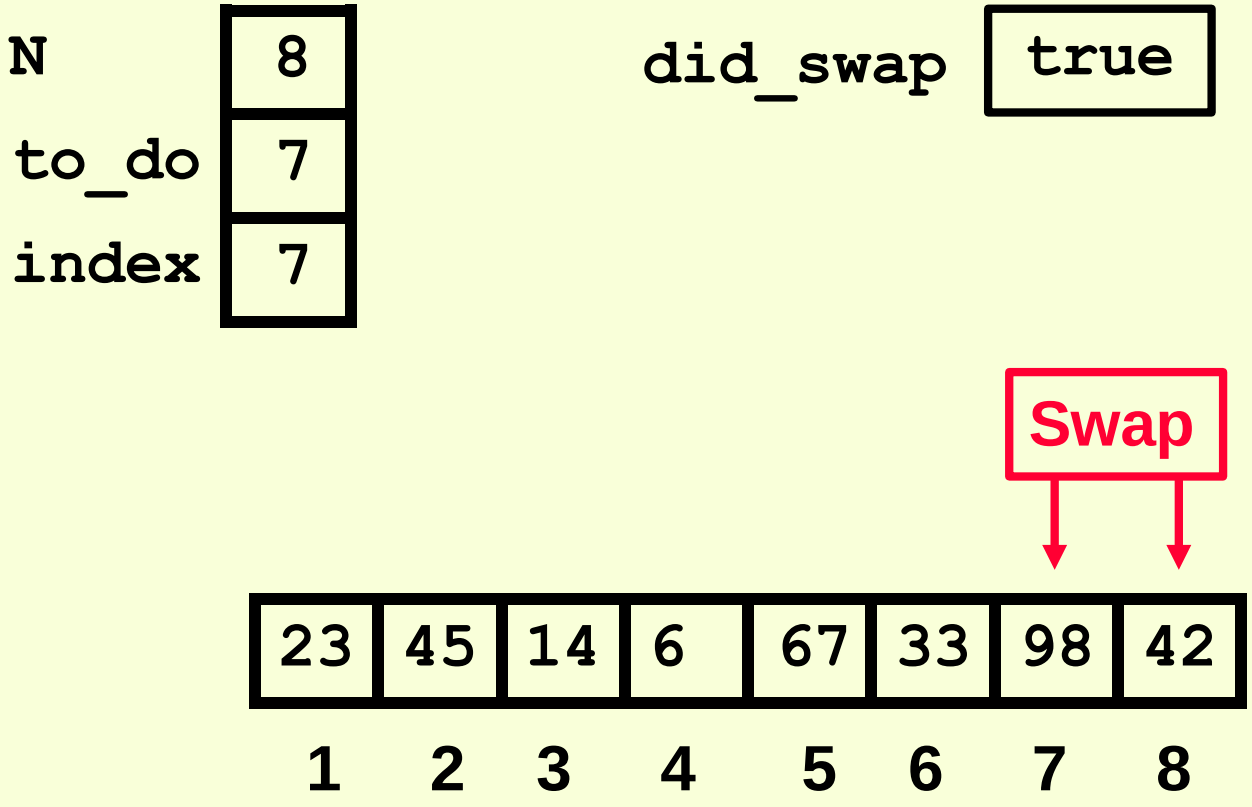
An Animated Example



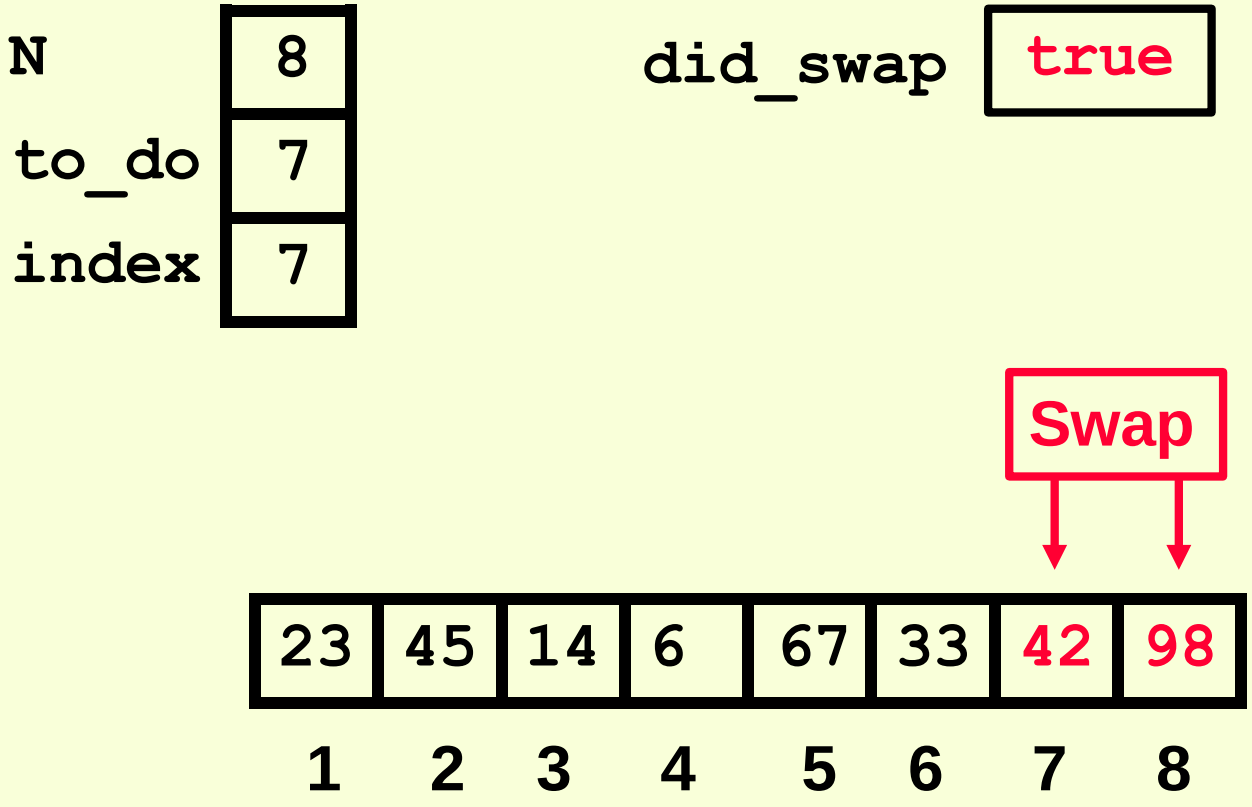
An Animated Example



An Animated Example



An Animated Example

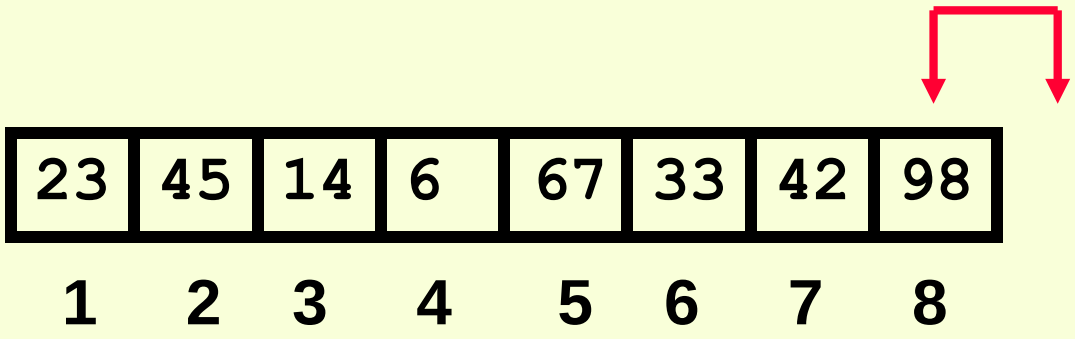


After First Pass of Outer Loop

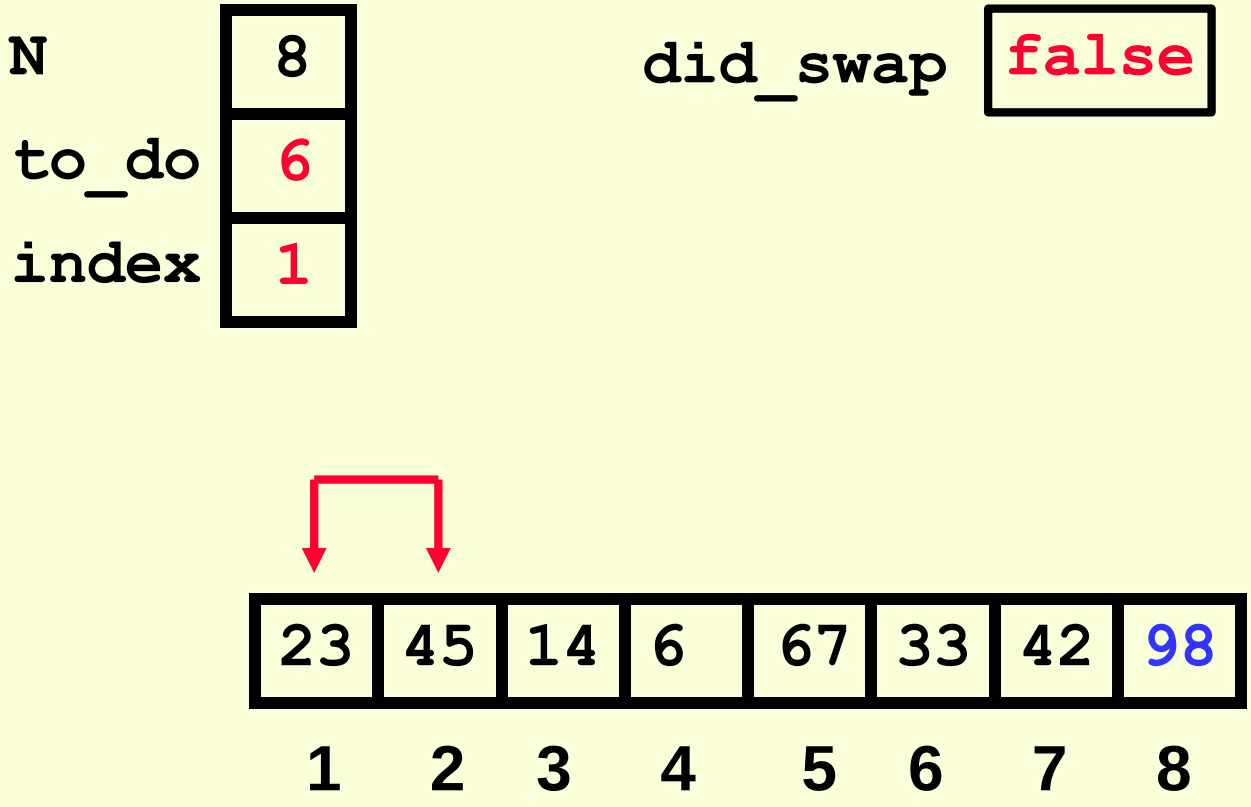
N	8
to_do	7
index	8

did_swap | true

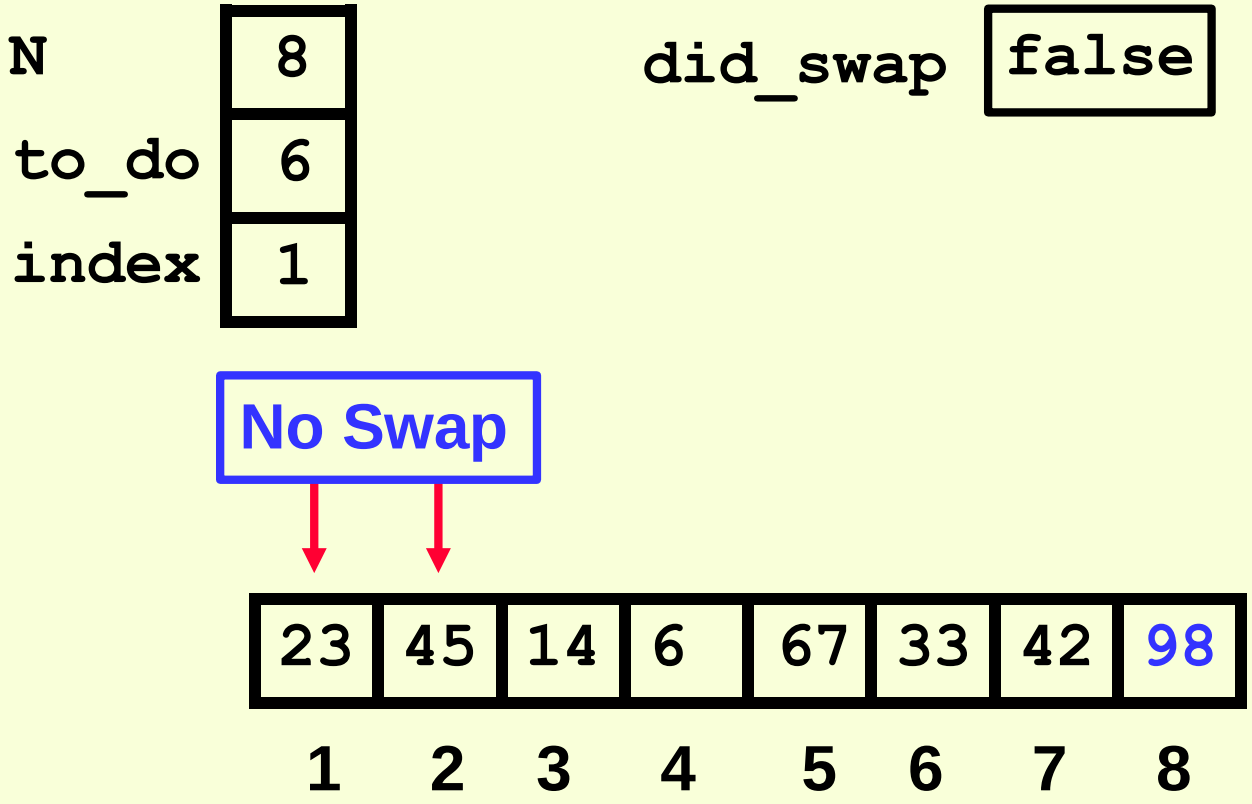
Finished first “Bubble Up”



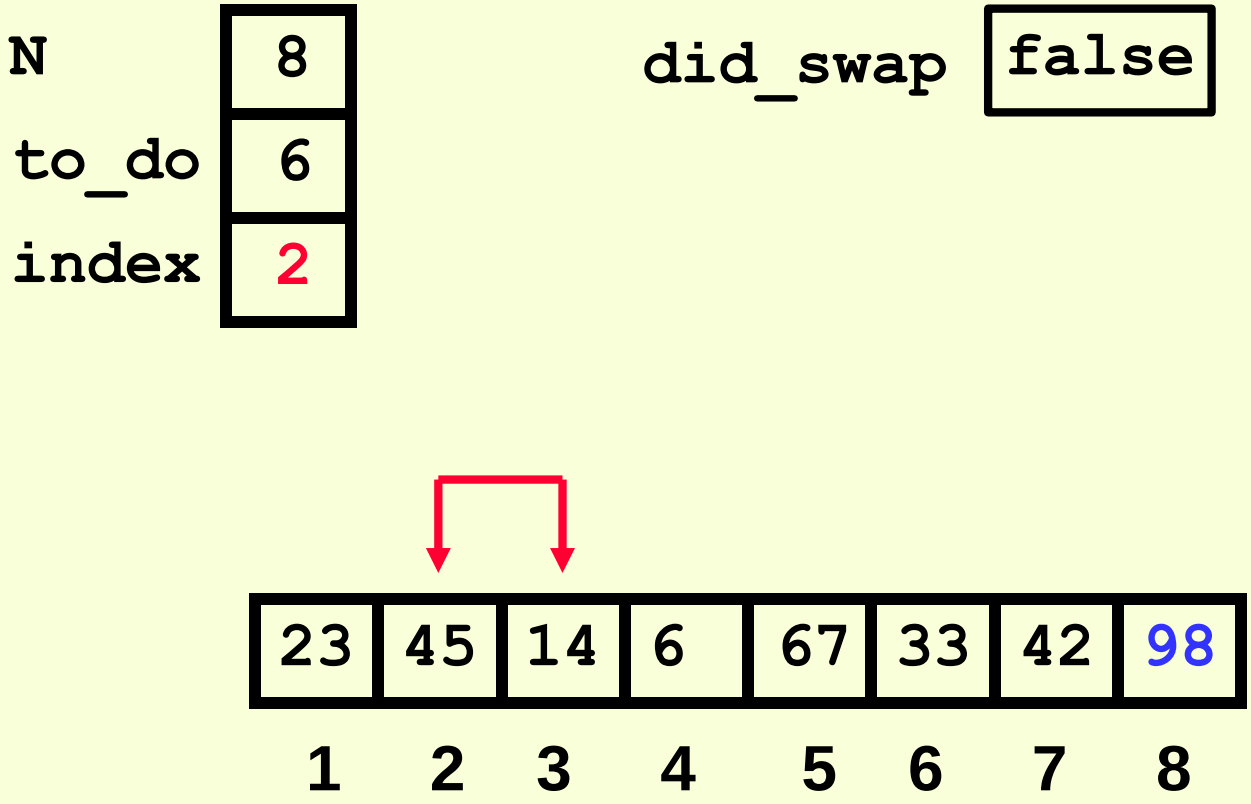
The Second “Bubble Up”



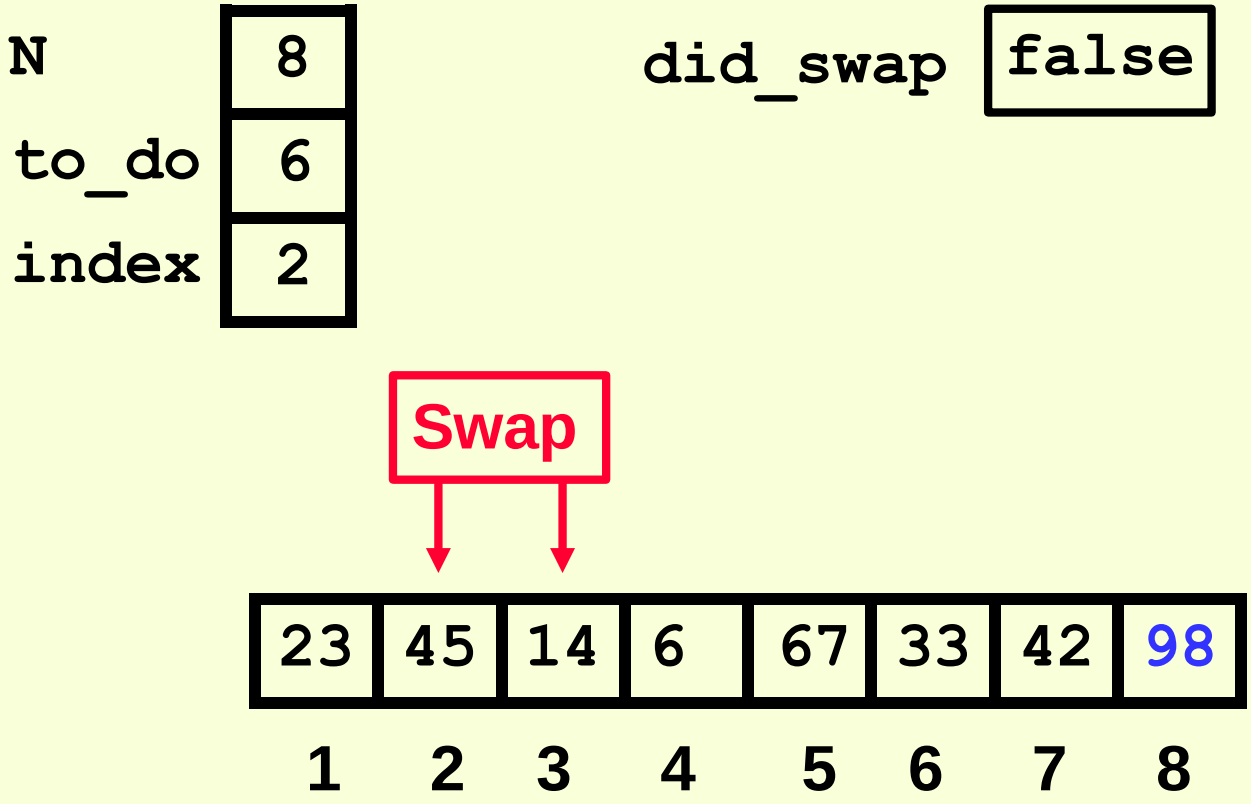
The Second “Bubble Up”



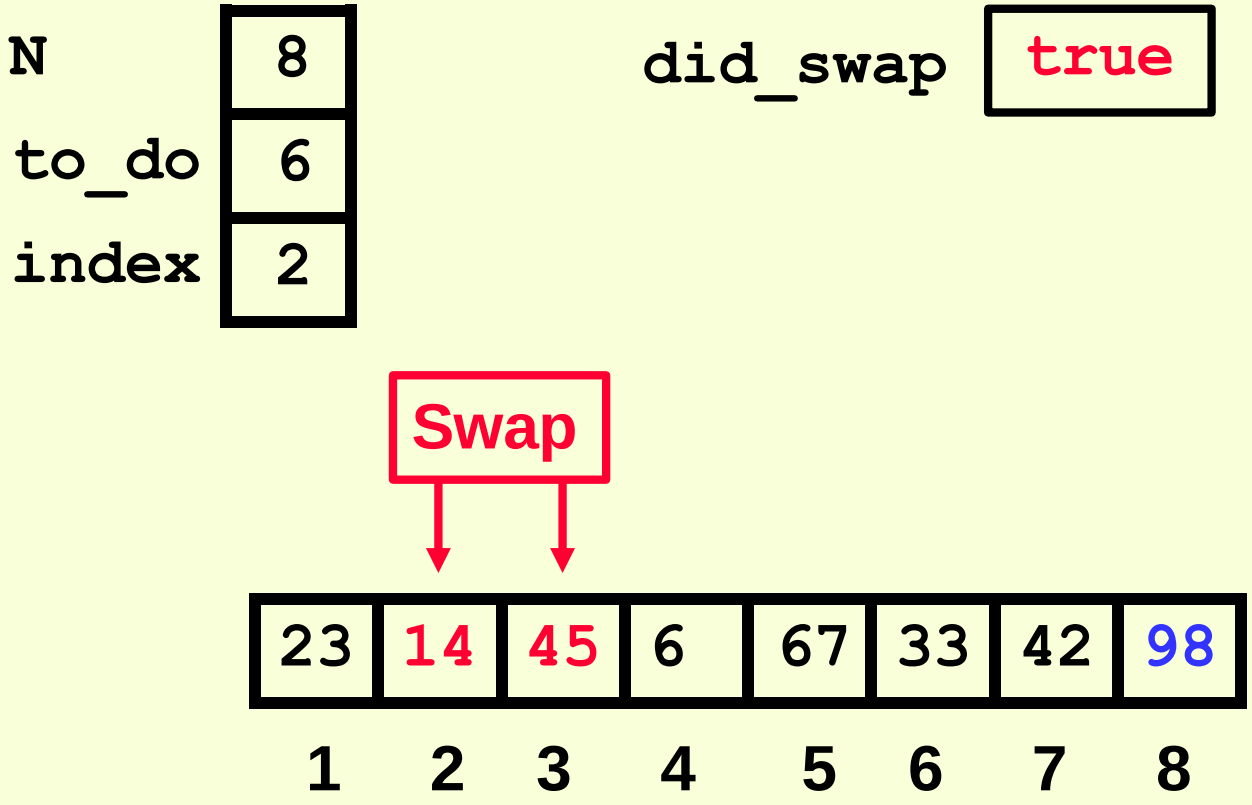
The Second “Bubble Up”



The Second “Bubble Up”



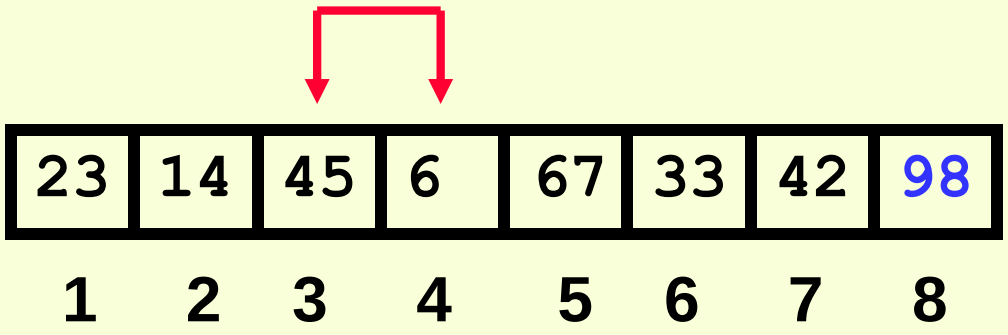
The Second “Bubble Up”



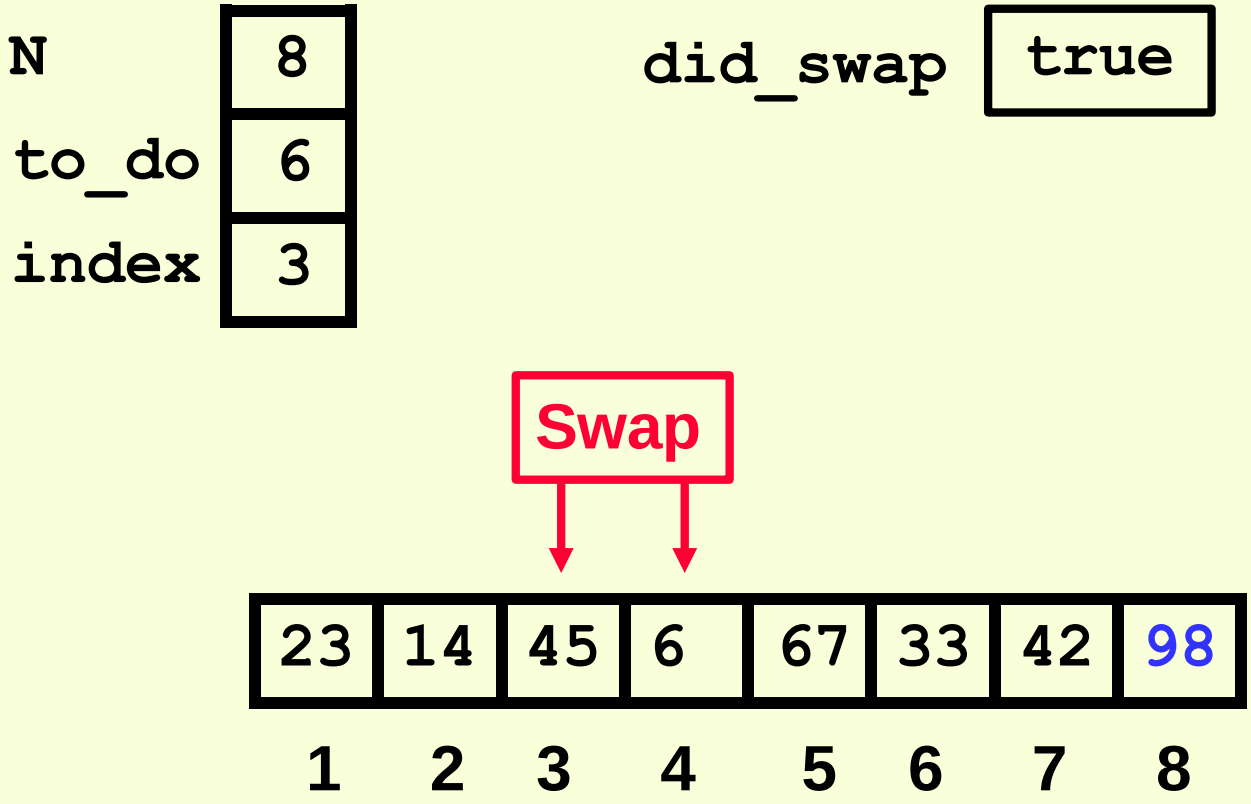
The Second “Bubble Up”

N	8
to_do	6
index	3

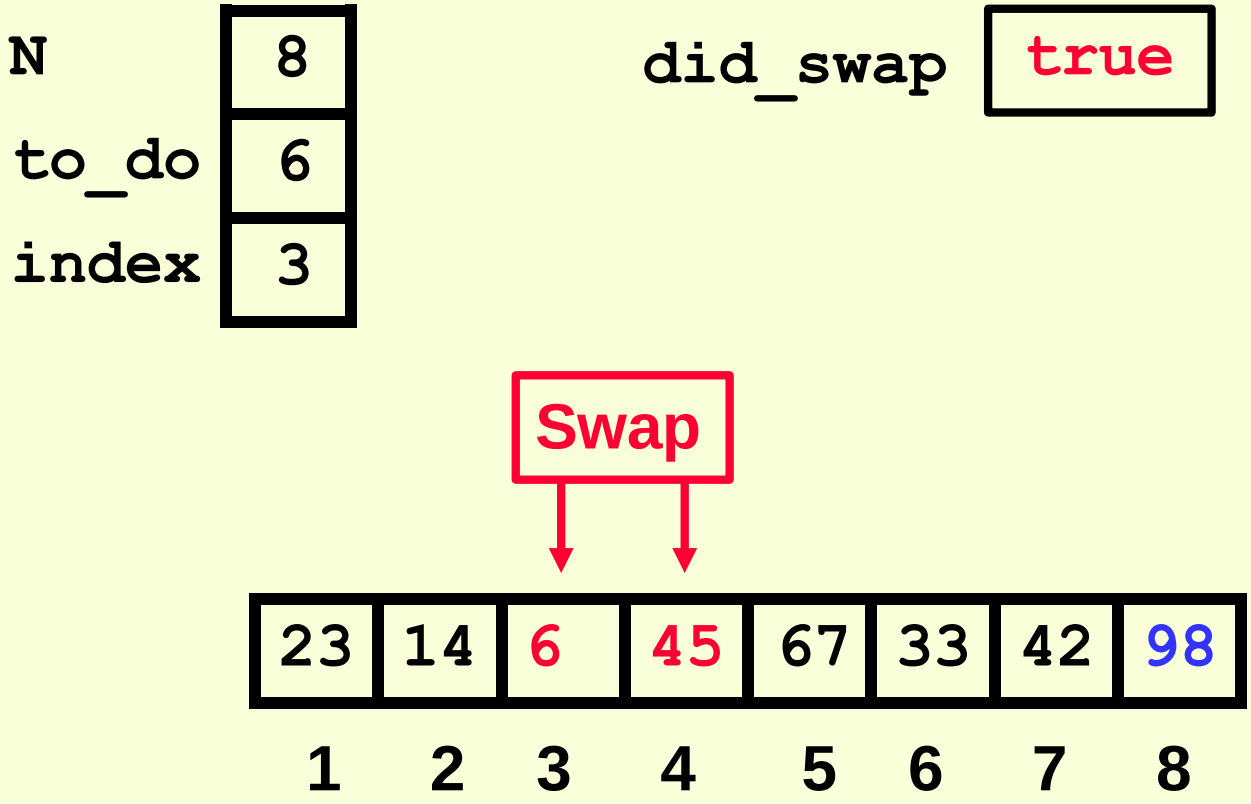
did_swap	true



The Second “Bubble Up”



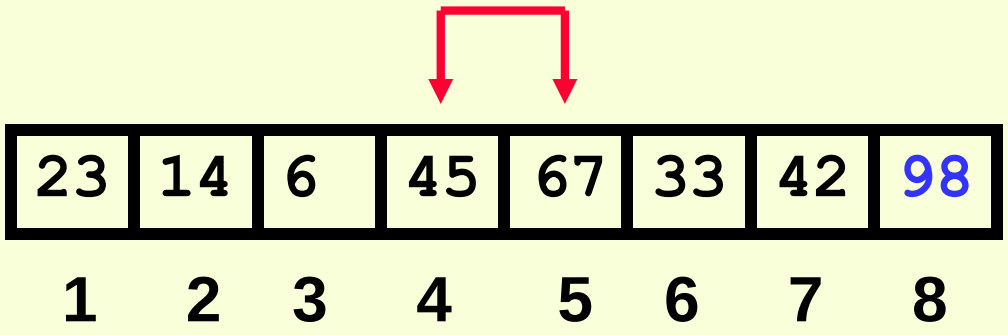
The Second “Bubble Up”



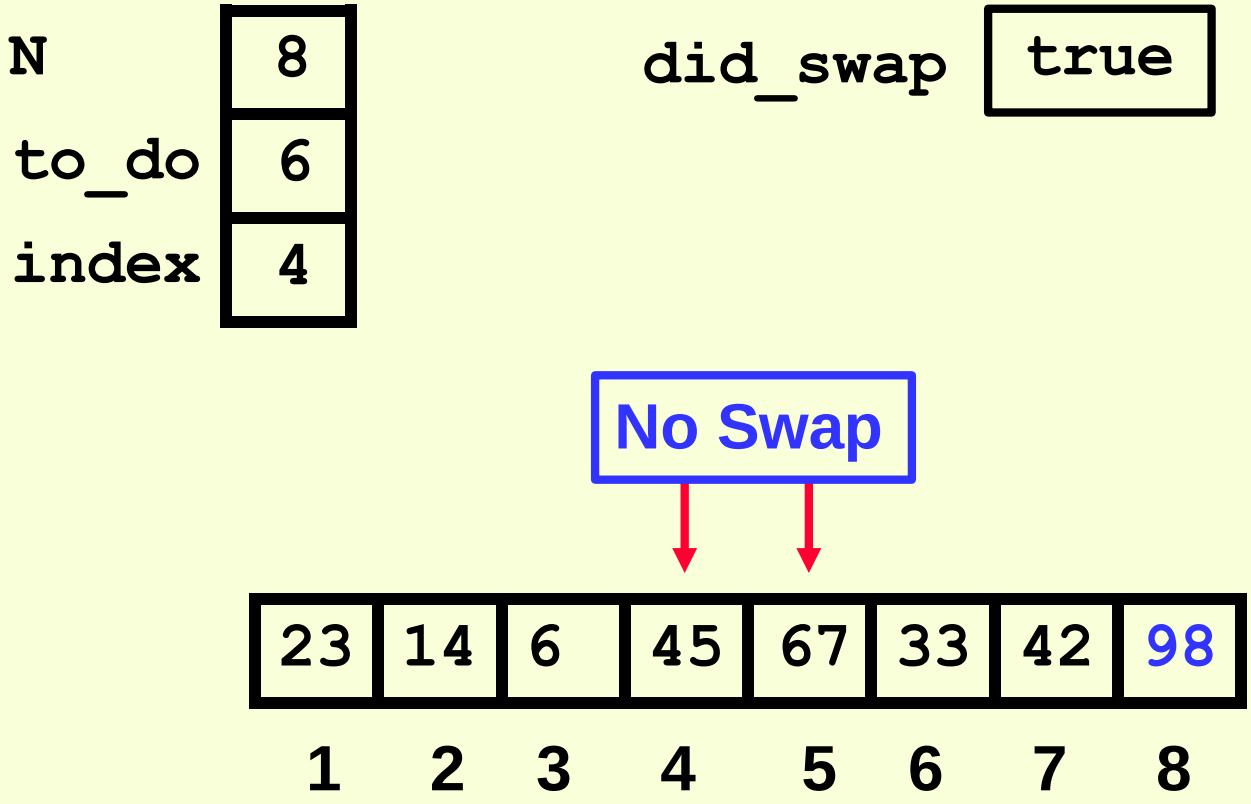
The Second “Bubble Up”

N	8
to_do	6
index	4

did_swap	true



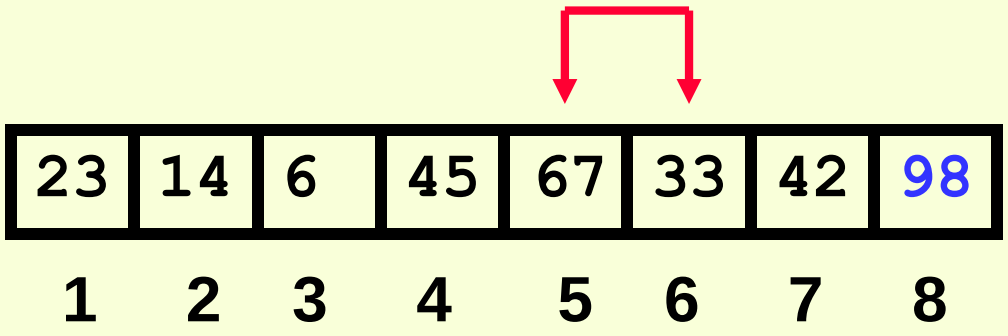
The Second “Bubble Up”



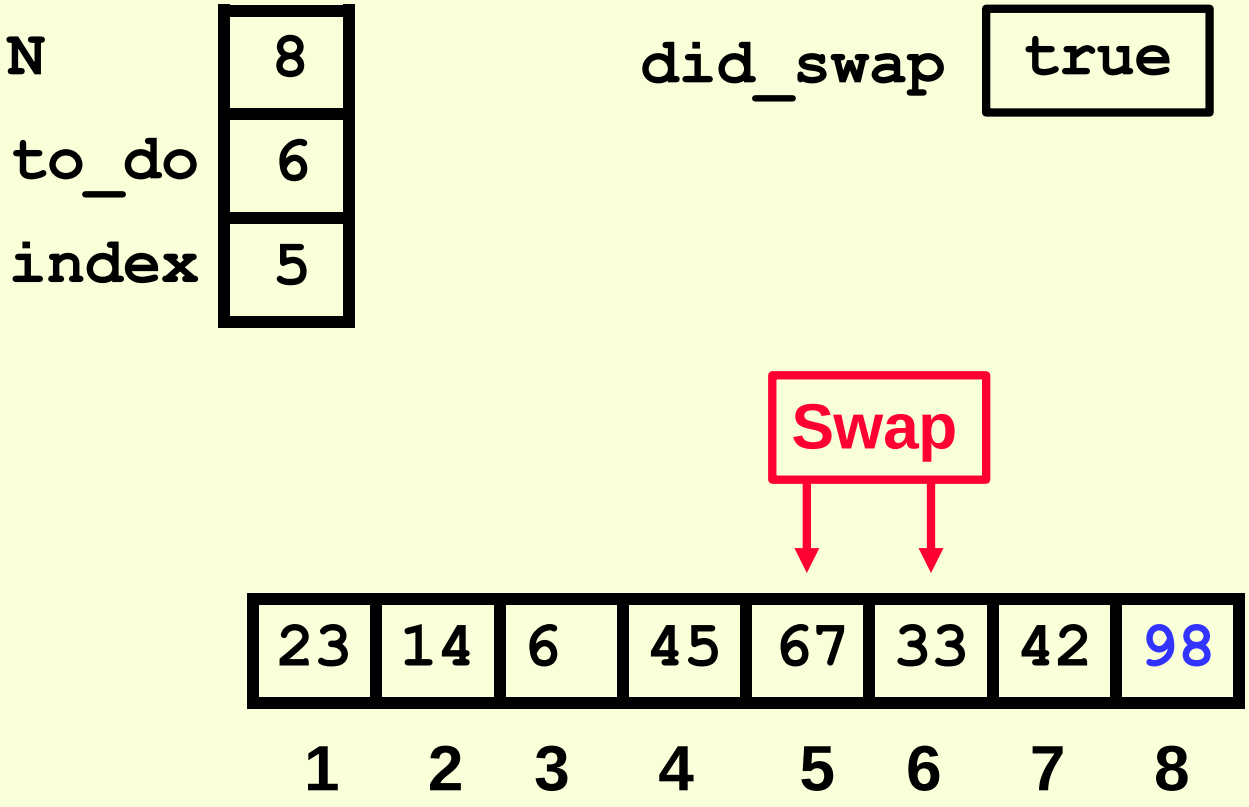
The Second “Bubble Up”

N	8
to_do	6
index	5

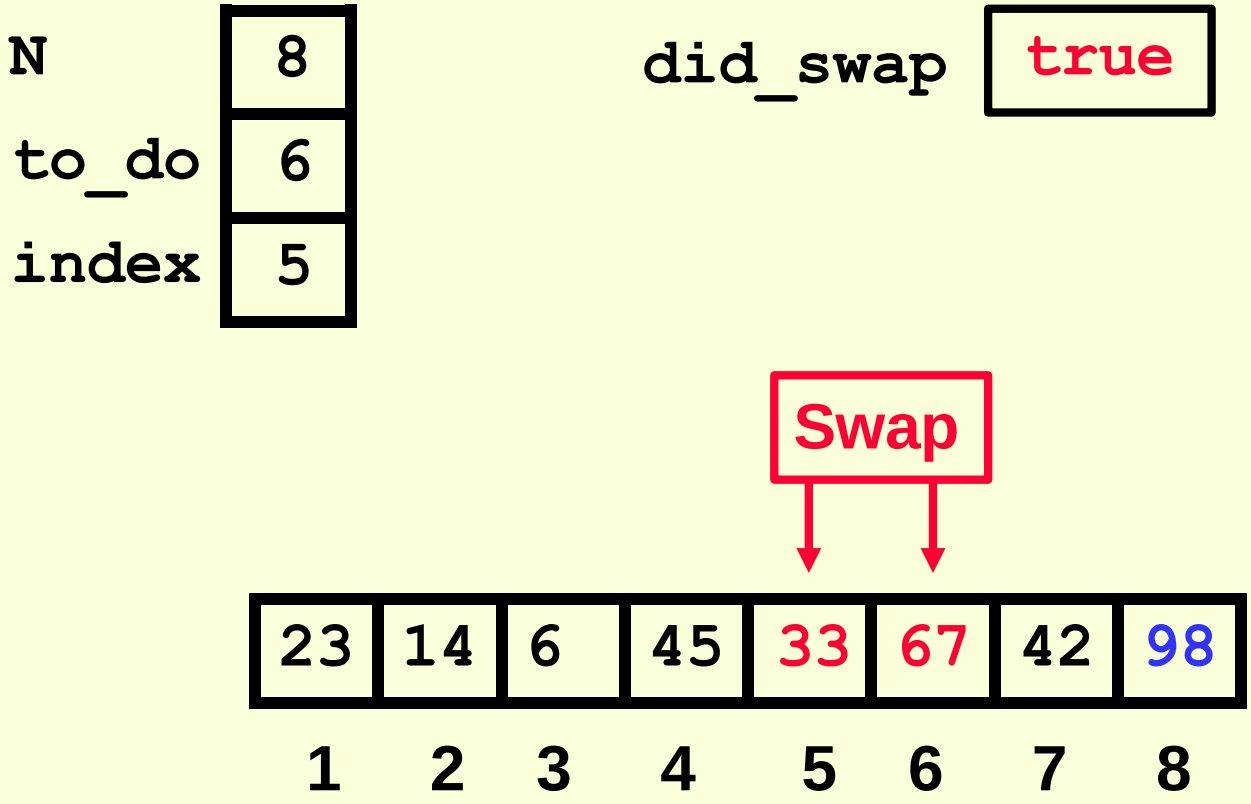
did_swap	true



The Second “Bubble Up”



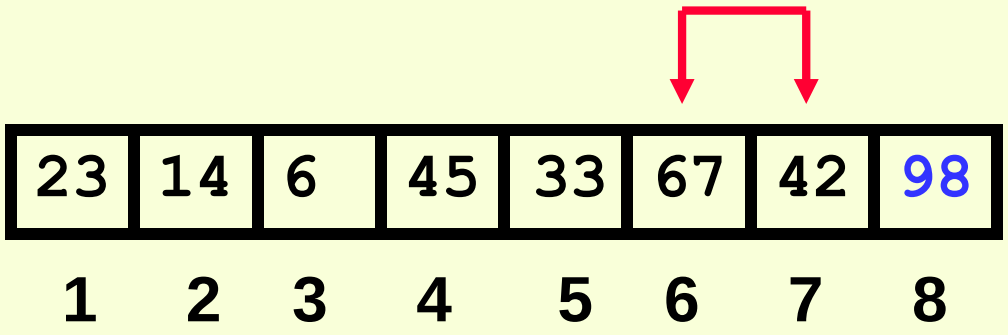
The Second “Bubble Up”



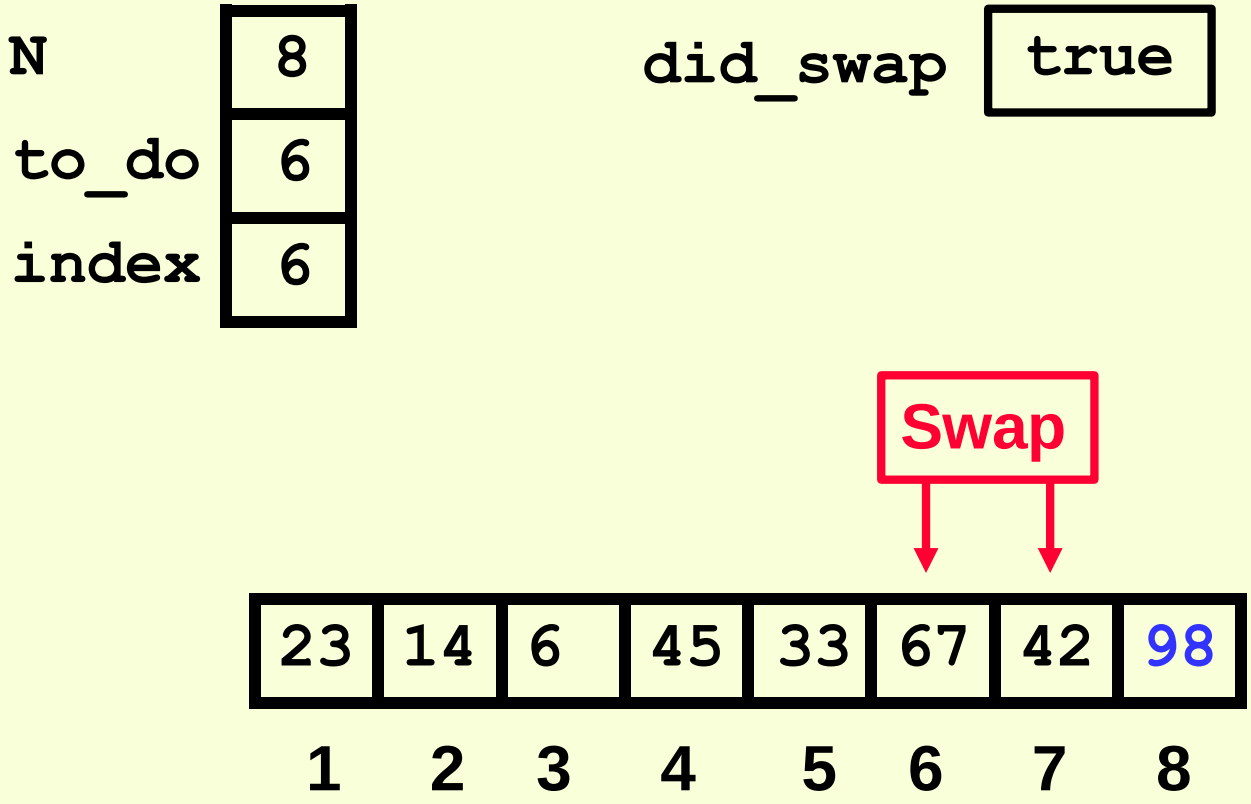
The Second “Bubble Up”

N	8
to_do	6
index	6

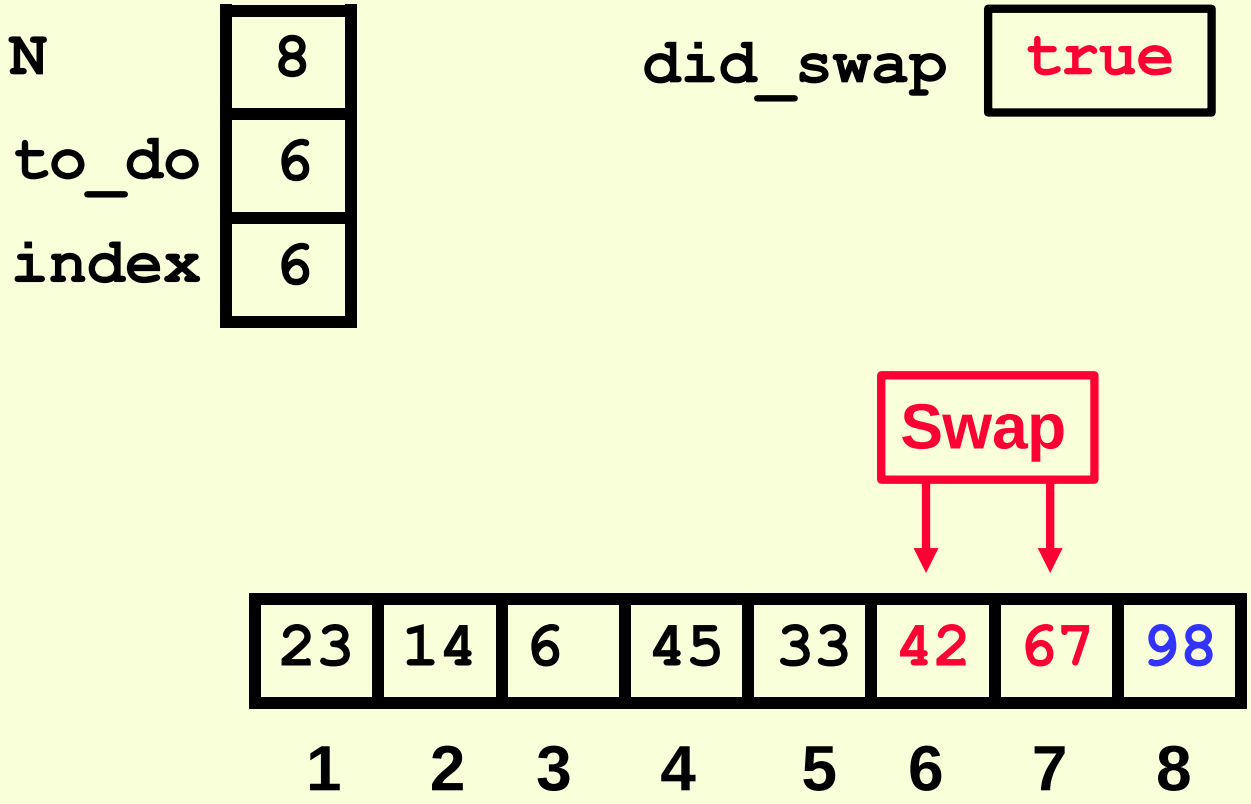
did_swap	true



The Second “Bubble Up”



The Second “Bubble Up”

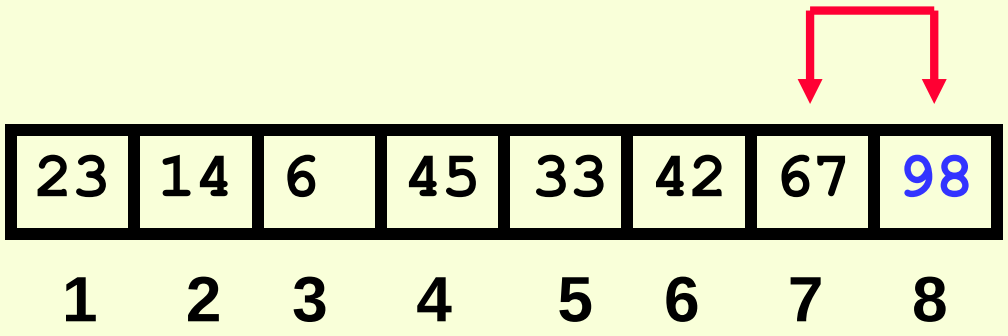


After Second Pass of Outer Loop

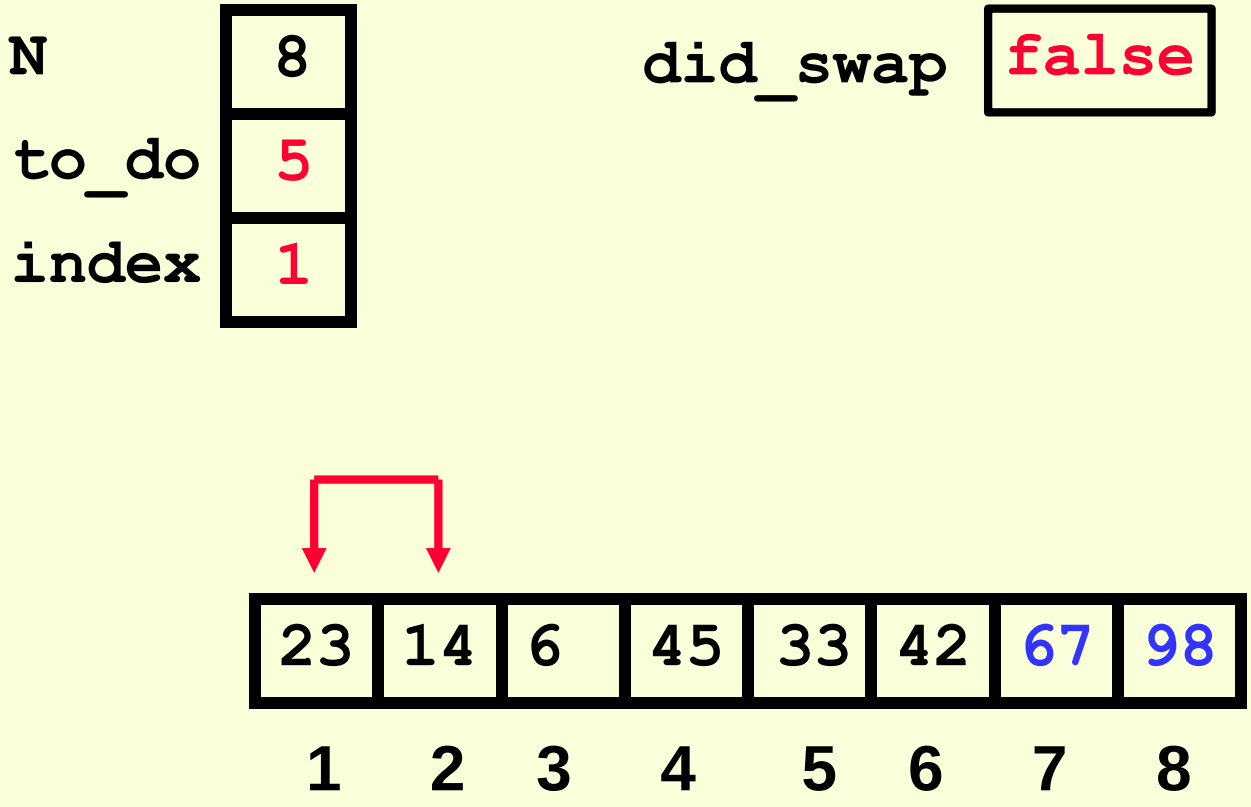
N	8
to_do	6
index	7

did_swap	true

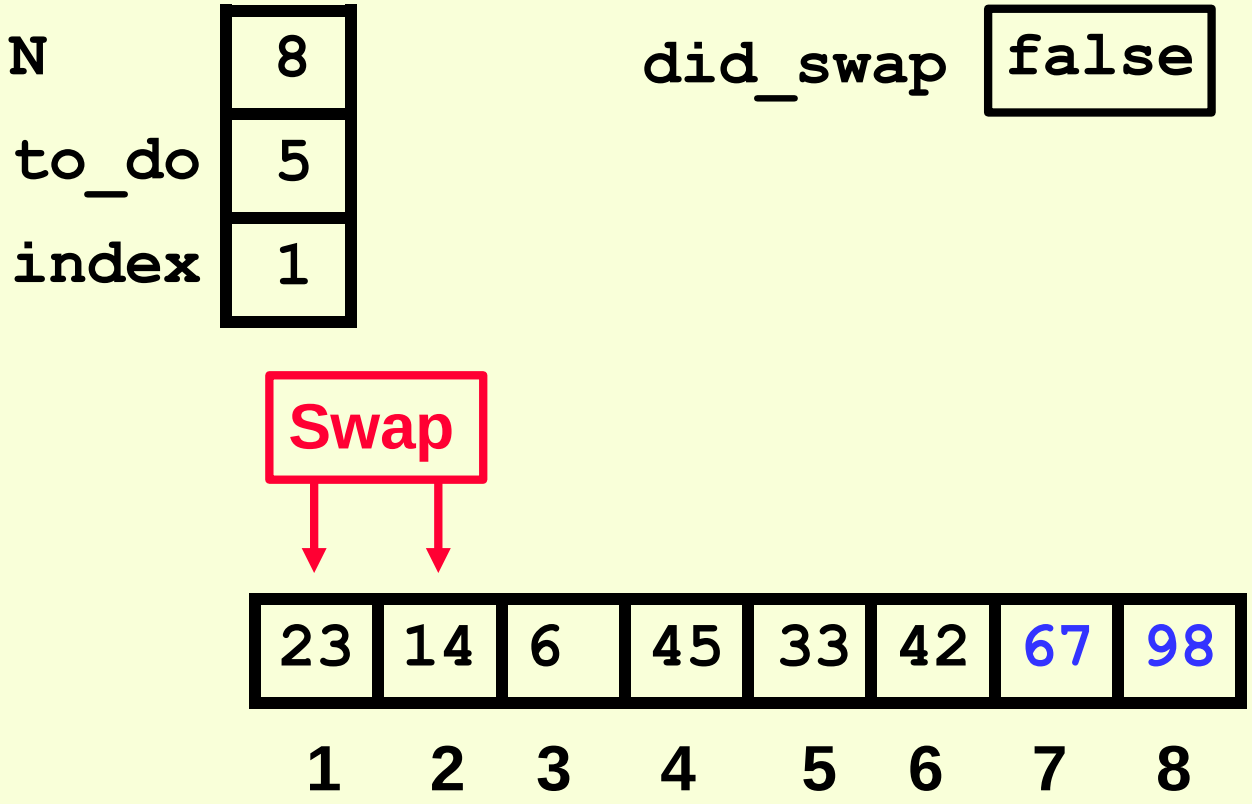
Finished second “Bubble Up”



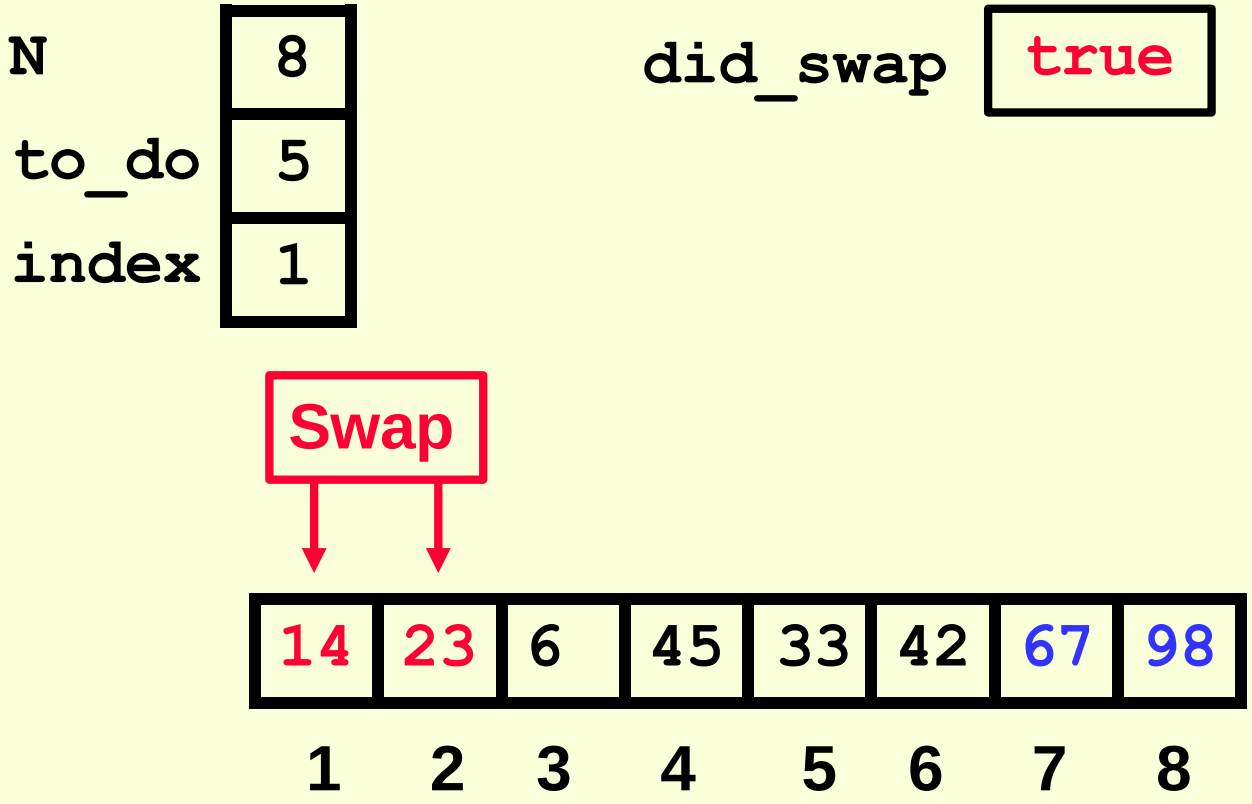
The Third “Bubble Up”



The Third “Bubble Up”



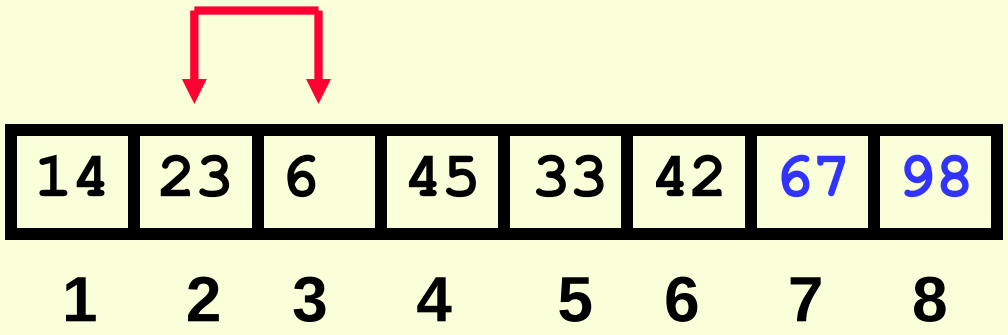
The Third “Bubble Up”



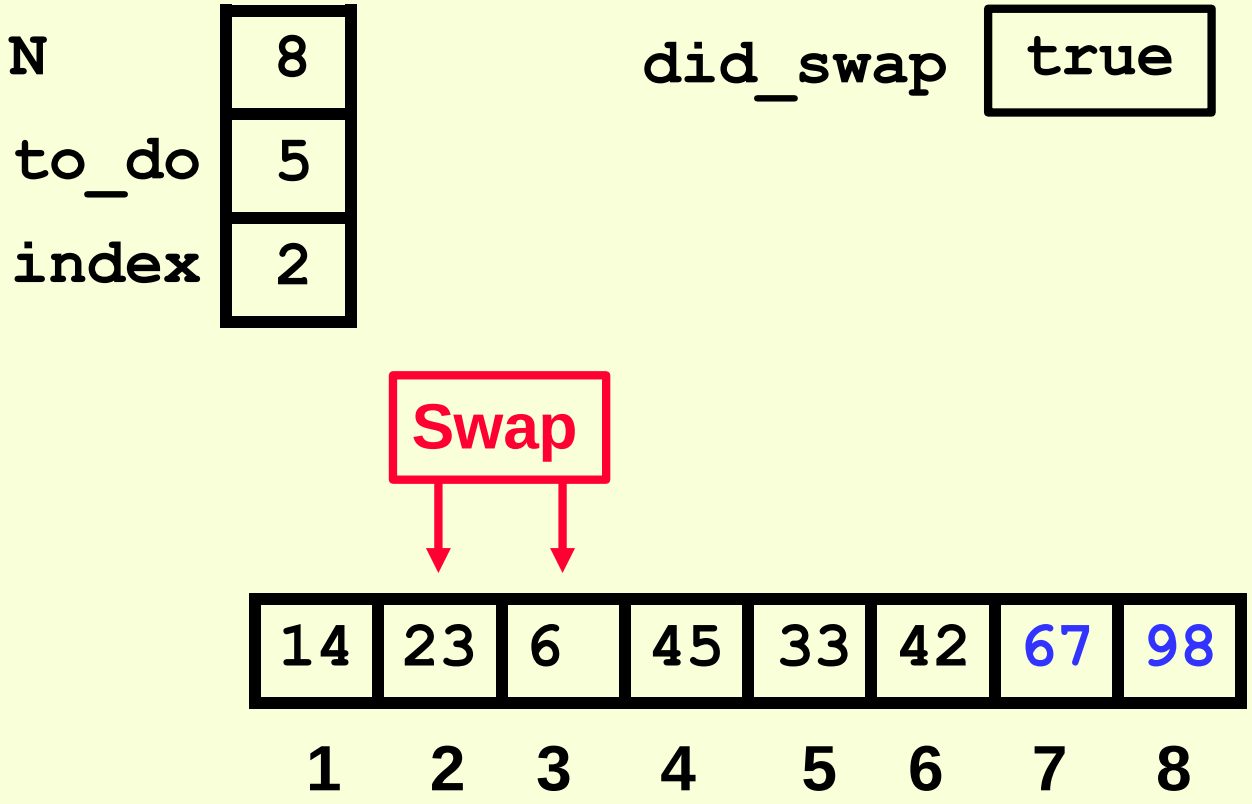
The Third “Bubble Up”

N	8
to_do	5
index	2

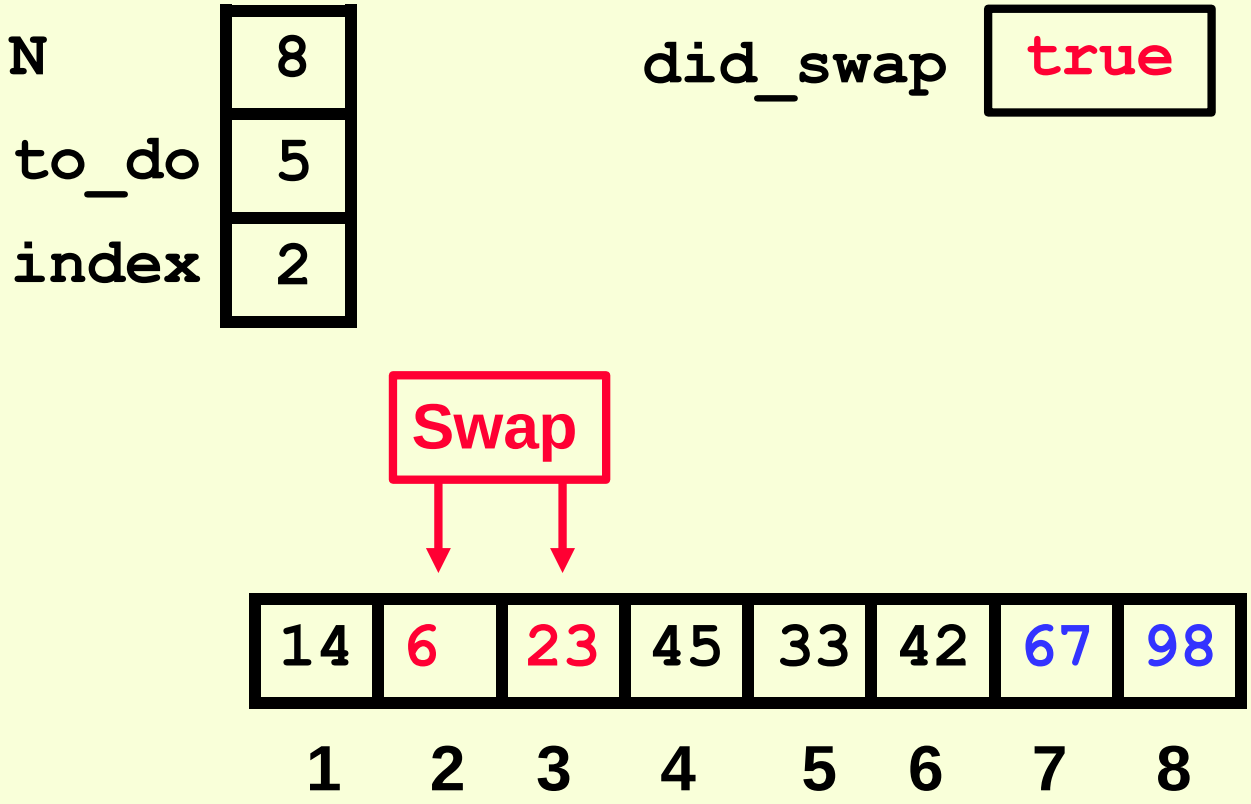
did_swap	true



The Third “Bubble Up”



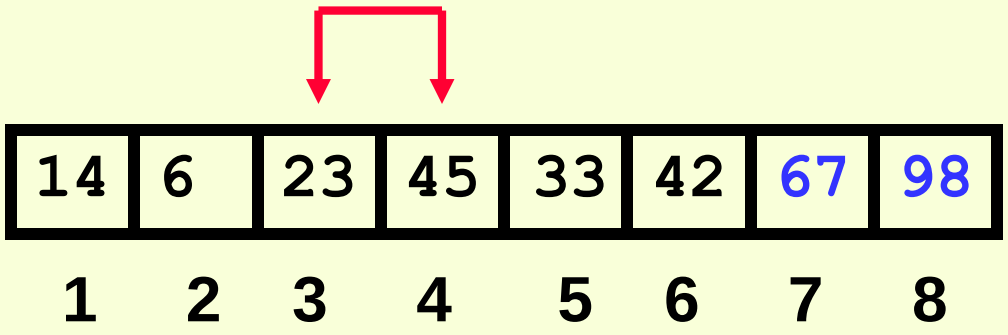
The Third “Bubble Up”



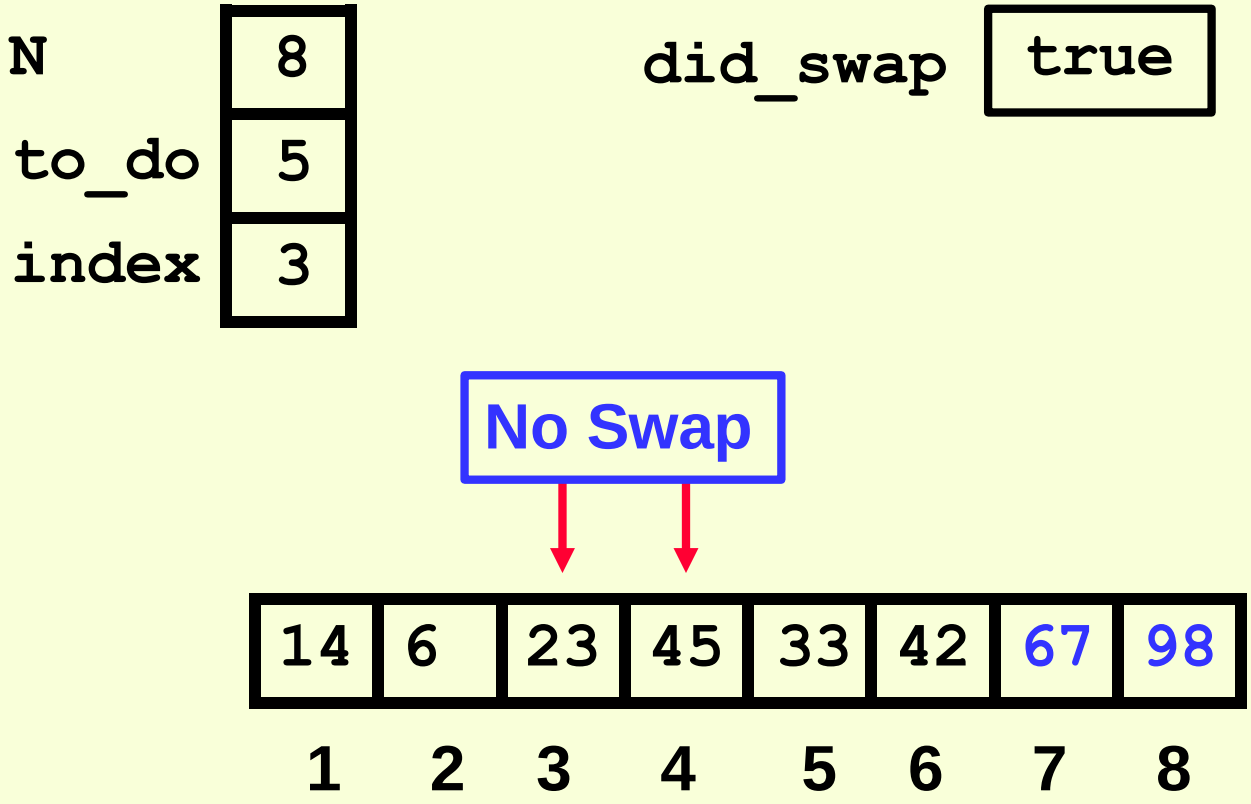
The Third “Bubble Up”

N	8
to_do	5
index	3

did_swap	true



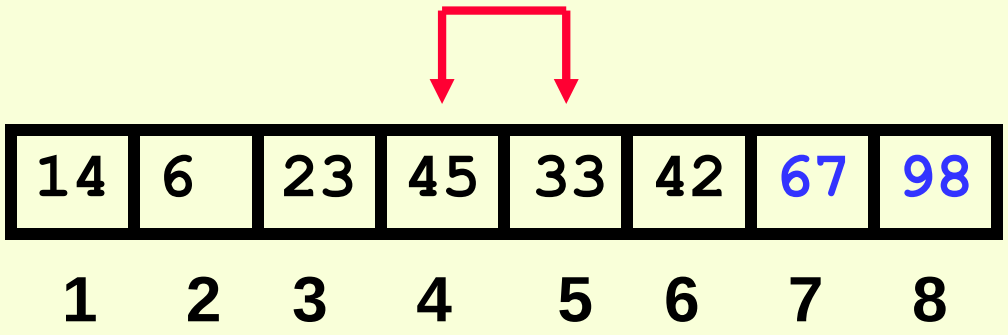
The Third “Bubble Up”



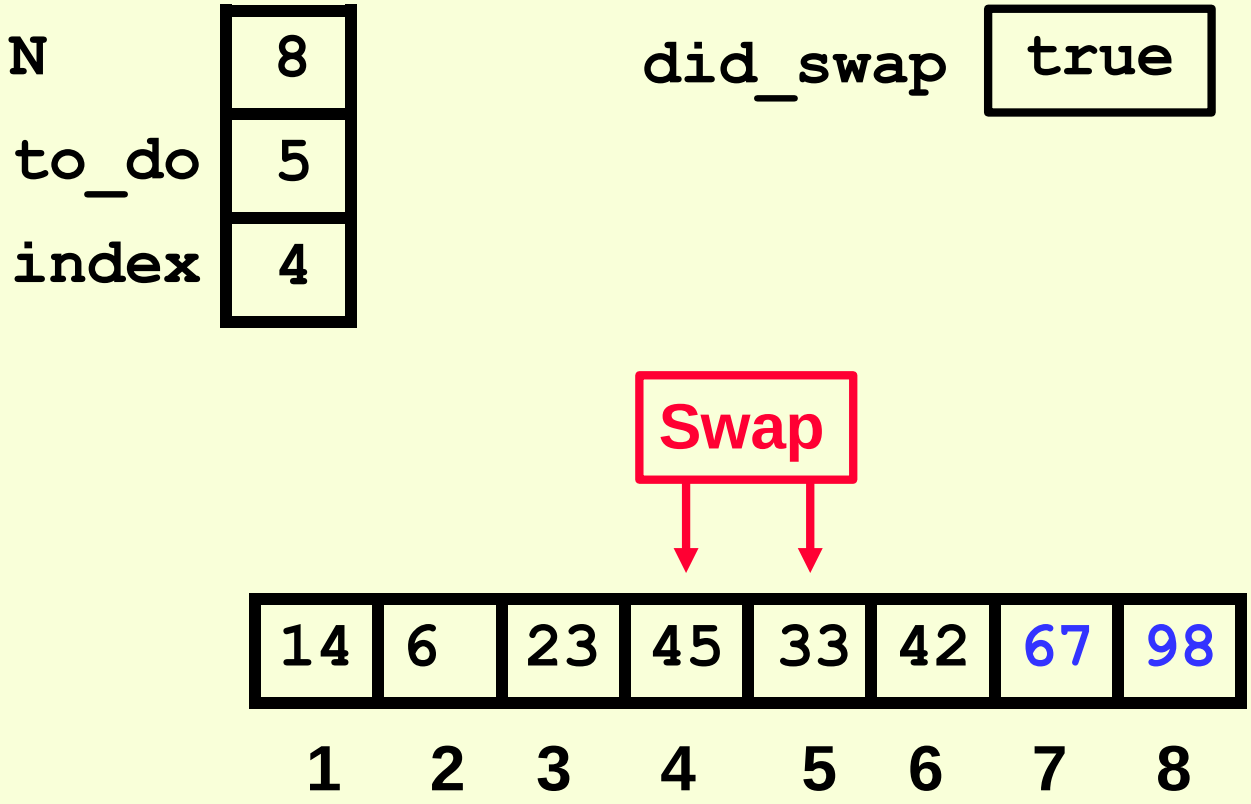
The Third “Bubble Up”

N	8
to_do	5
index	4

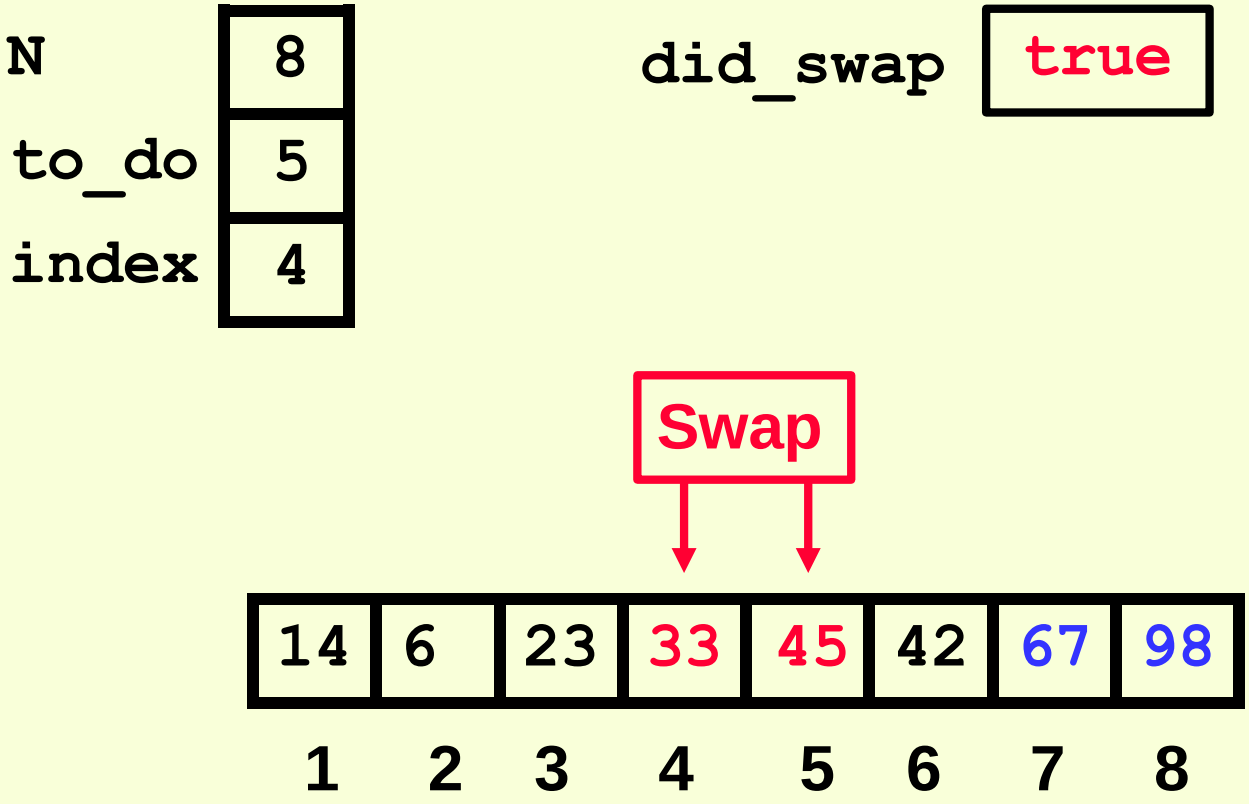
did_swap	true



The Third “Bubble Up”



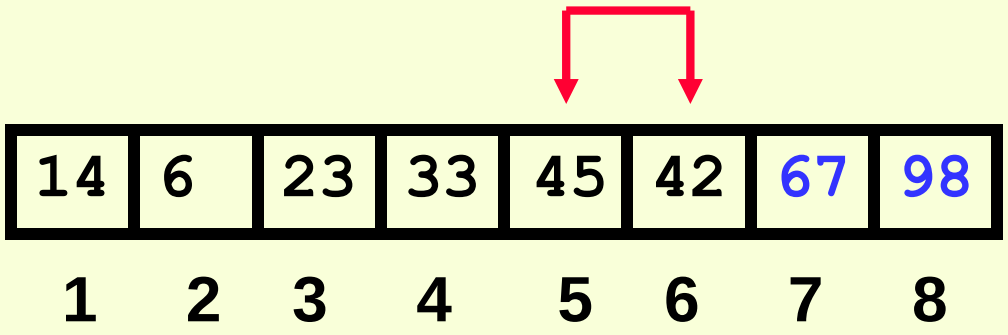
The Third “Bubble Up”



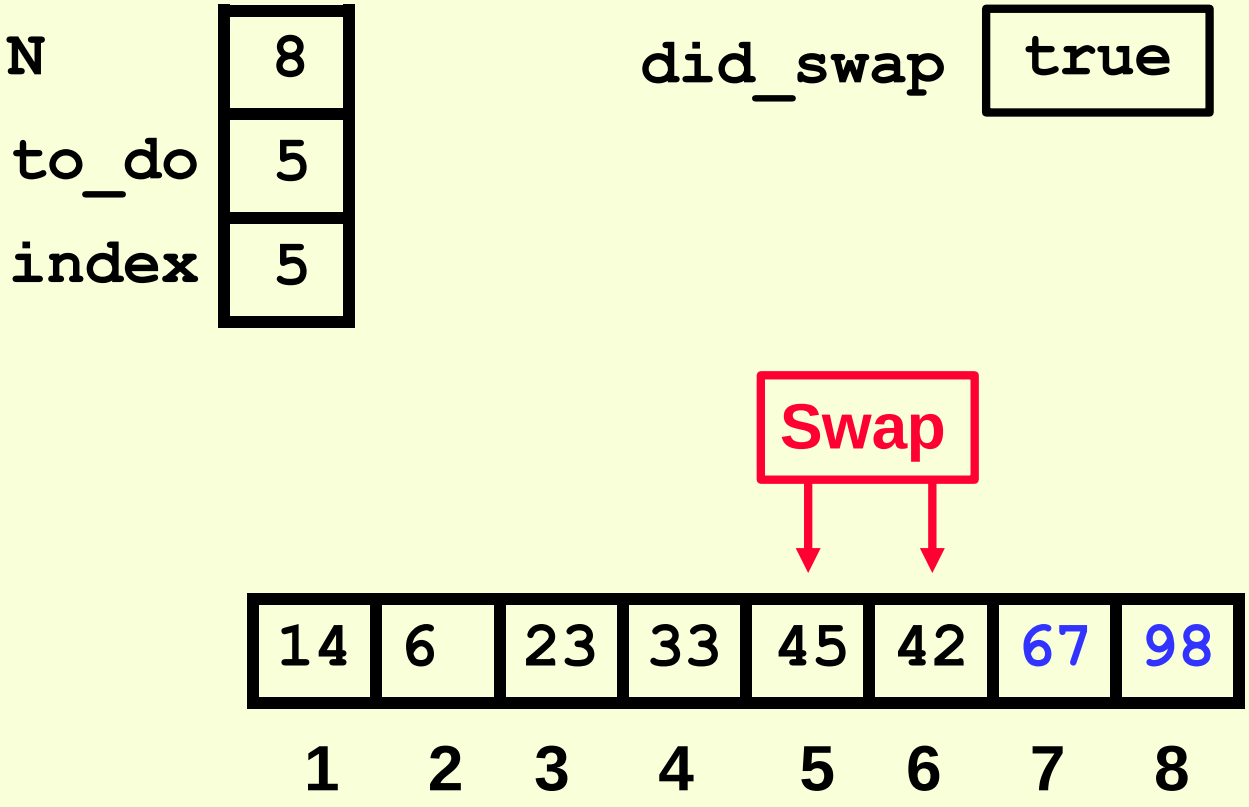
The Third “Bubble Up”

N	8
to_do	5
index	5

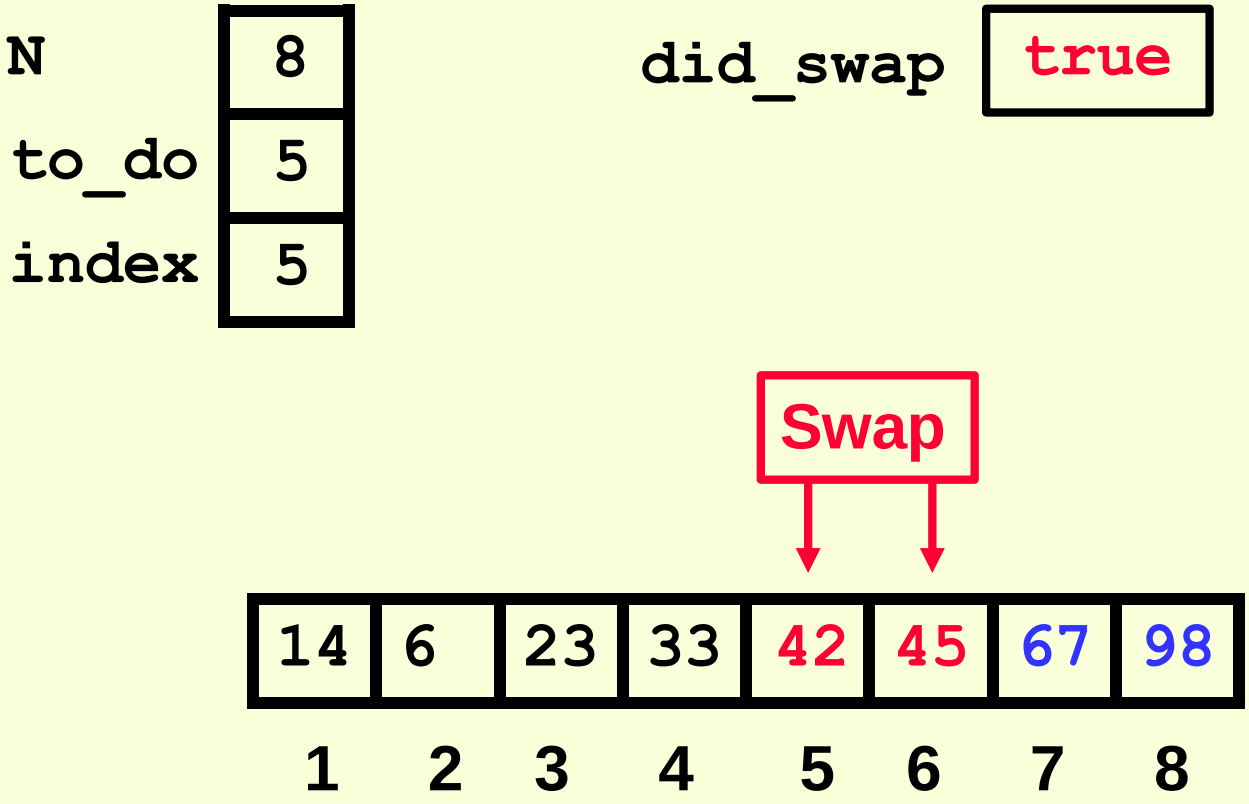
did_swap	true



The Third “Bubble Up”



The Third “Bubble Up”

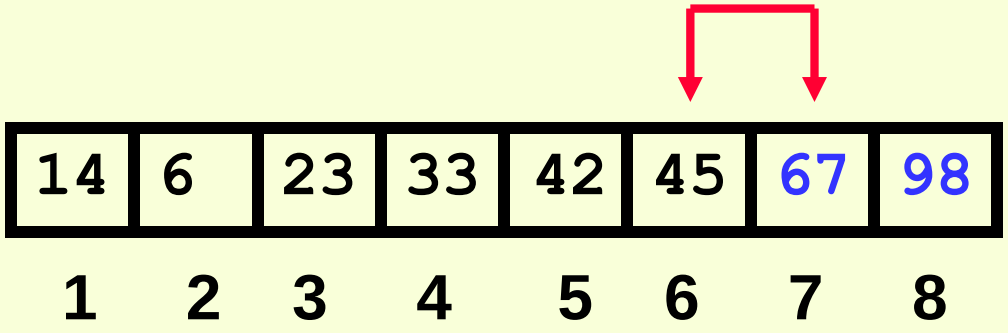


After Third Pass of Outer Loop

N	8
to_do	5
index	6

did_swap	true

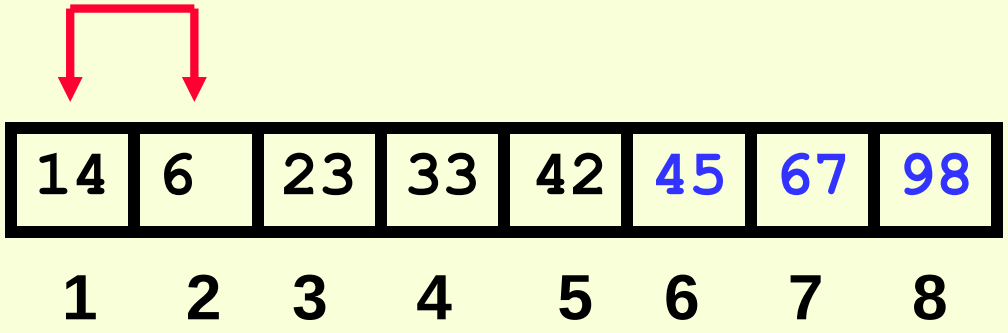
Finished third “Bubble Up”



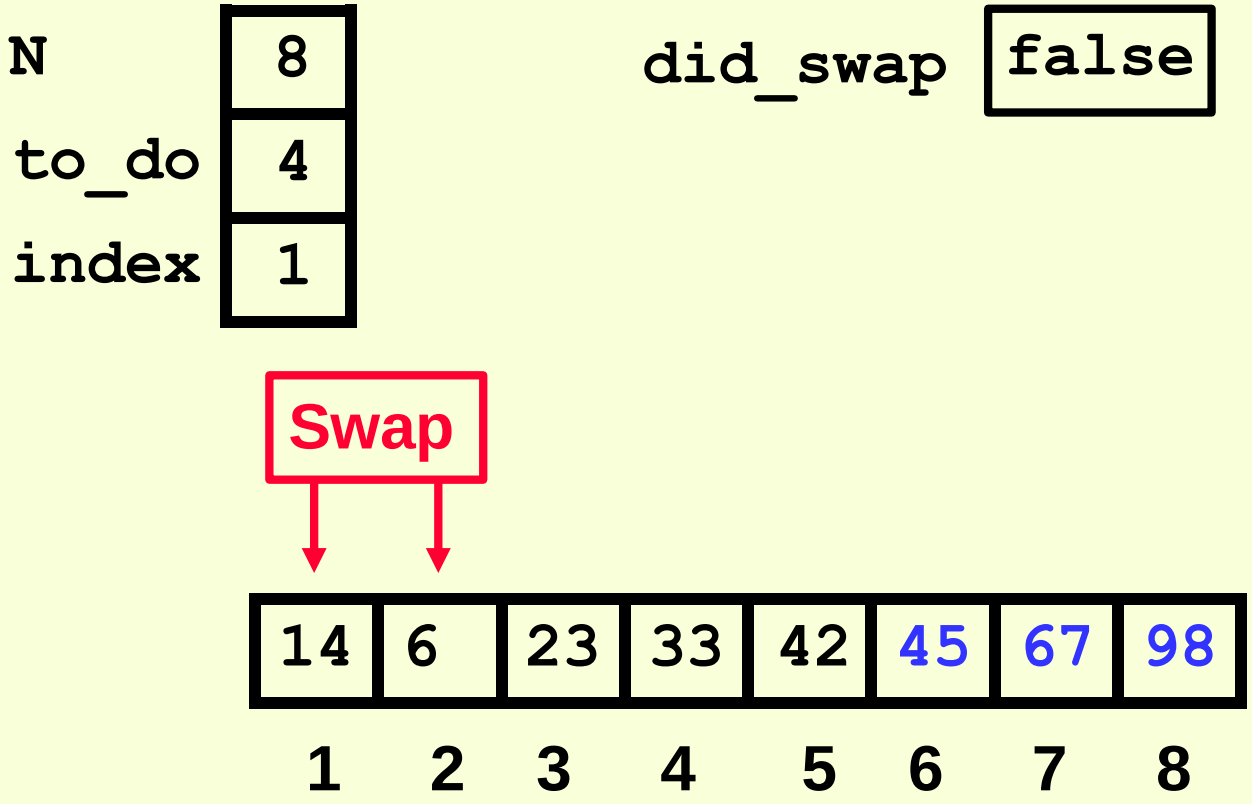
The Fourth “Bubble Up”

N	8
to_do	4
index	1

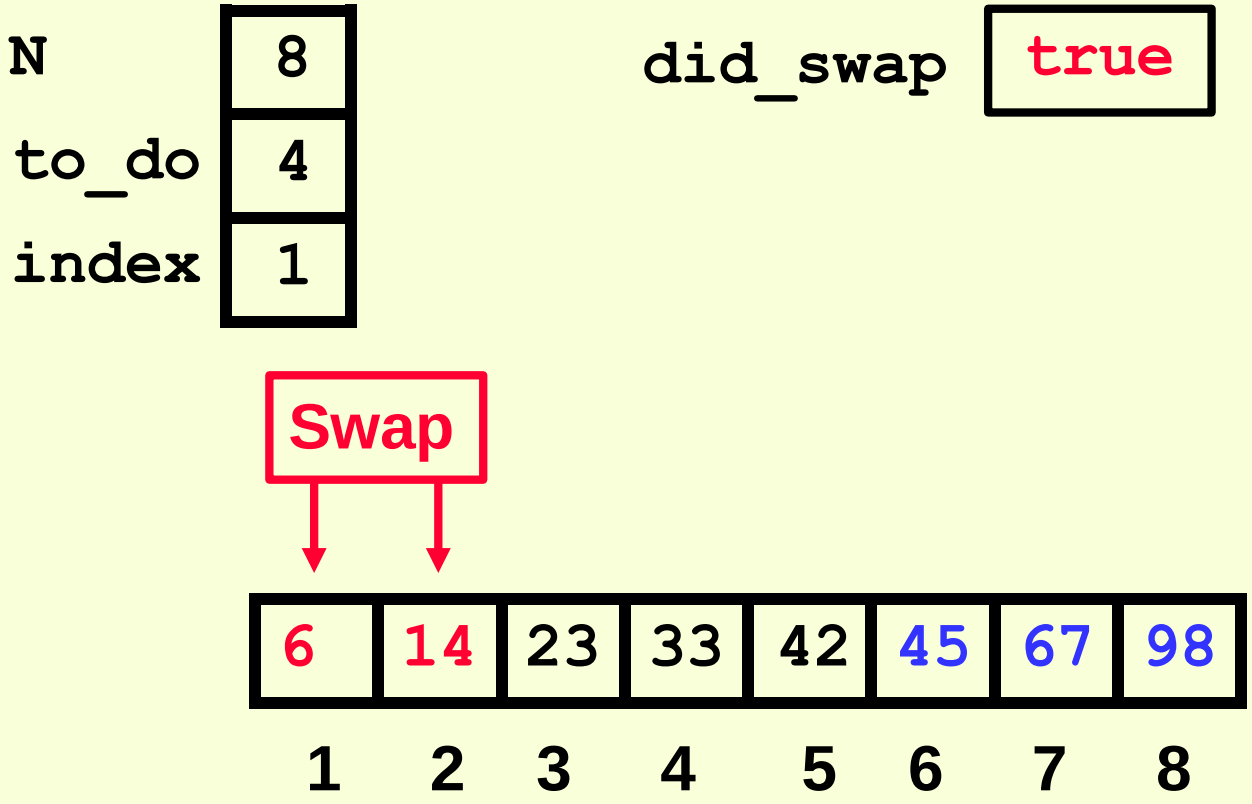
did_swap | false



The Fourth “Bubble Up”



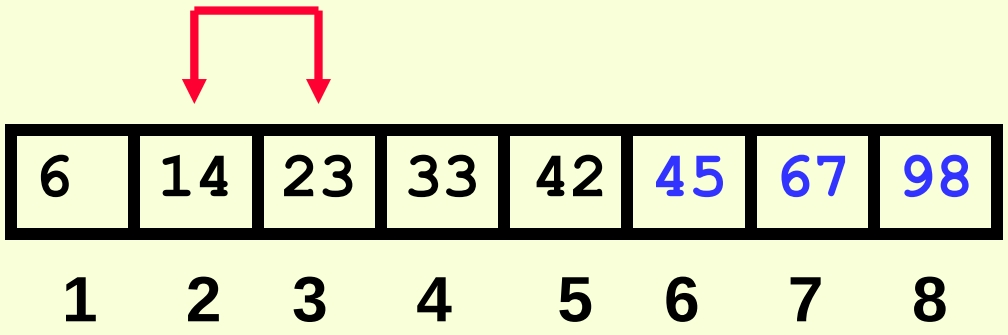
The Fourth “Bubble Up”



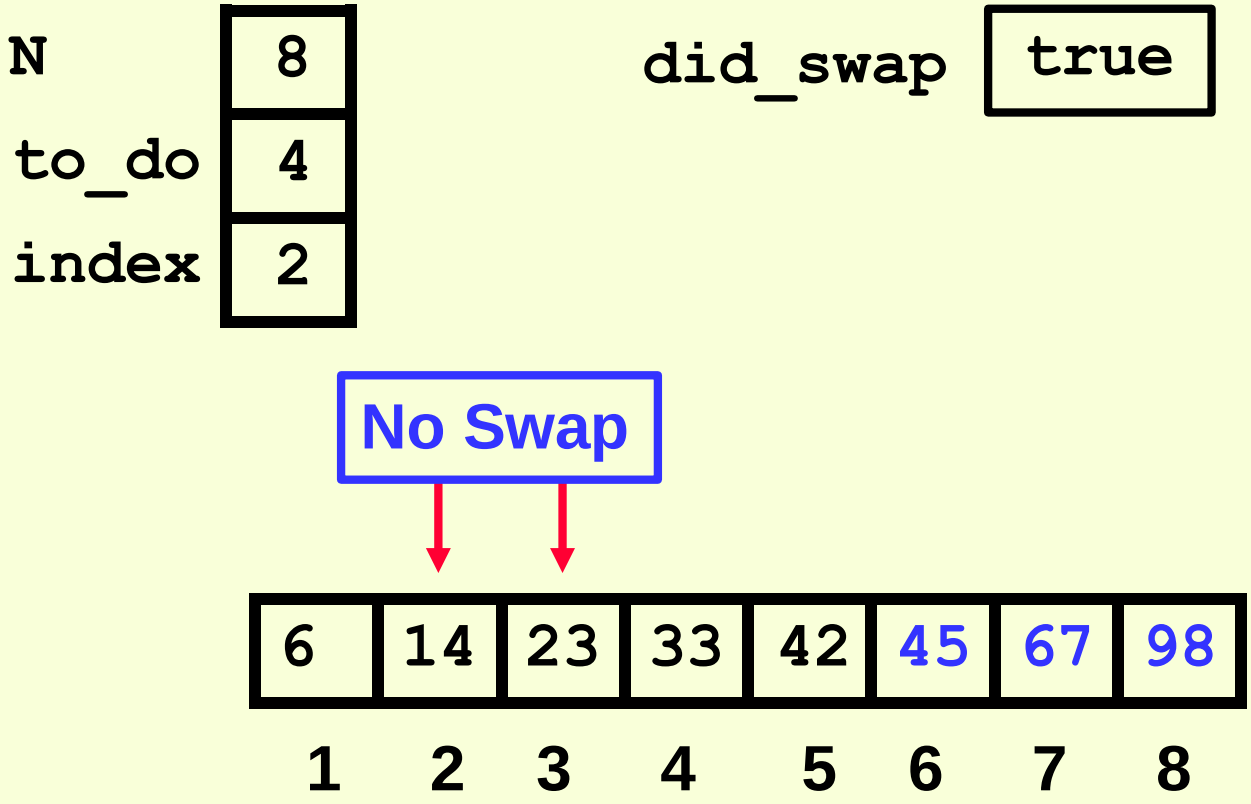
The Fourth “Bubble Up”

N	8
to_do	4
index	2

did_swap	true



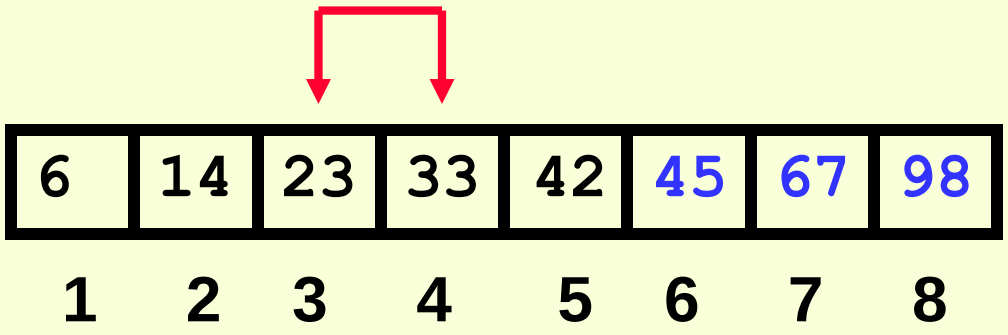
The Fourth “Bubble Up”



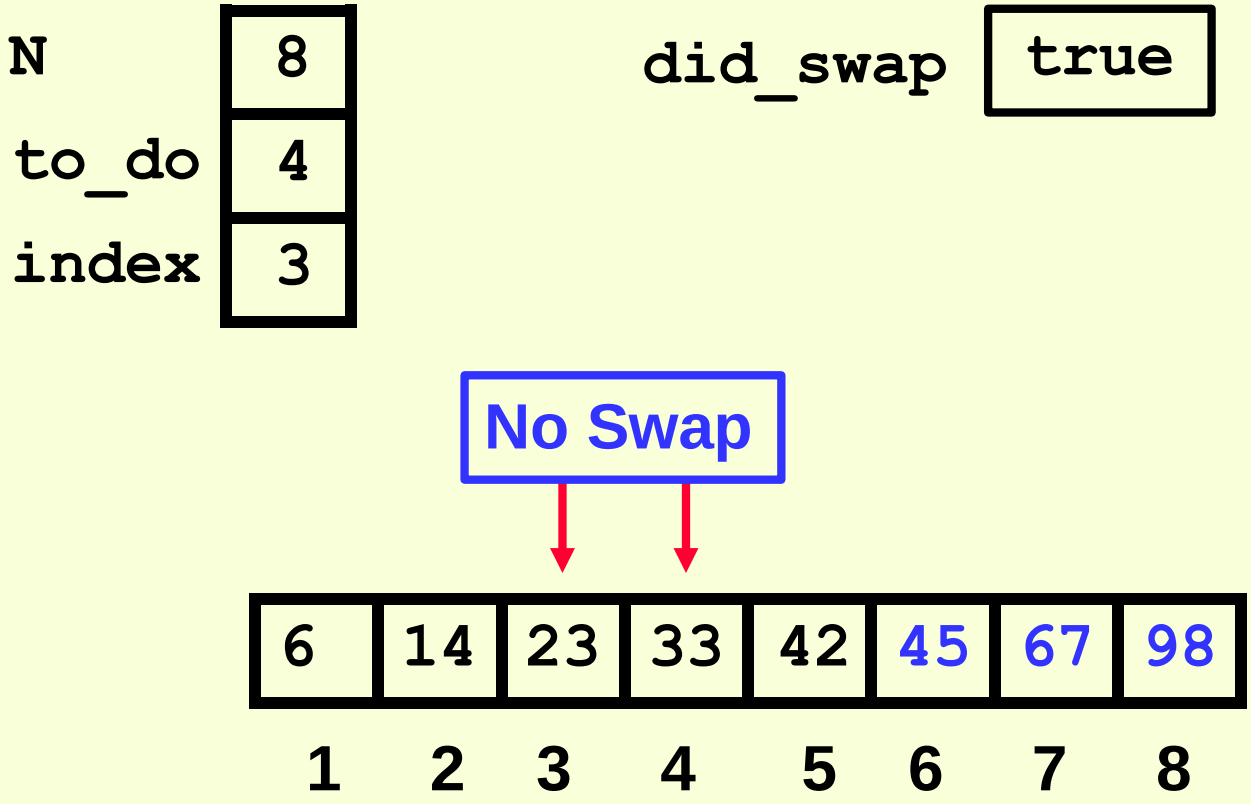
The Fourth “Bubble Up”

N	8
to_do	4
index	3

did_swap	true



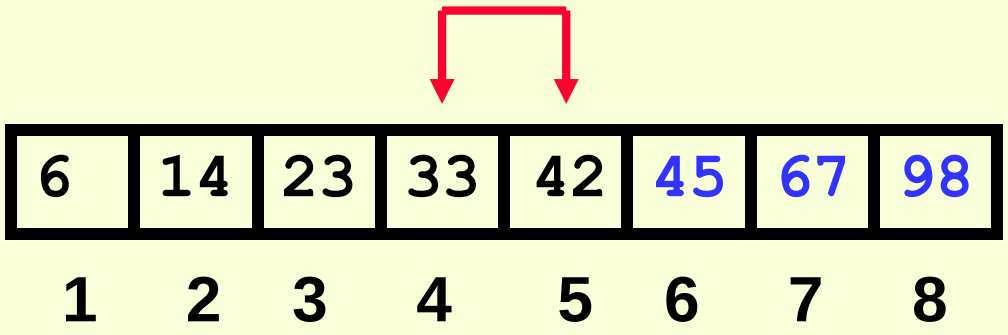
The Fourth “Bubble Up”



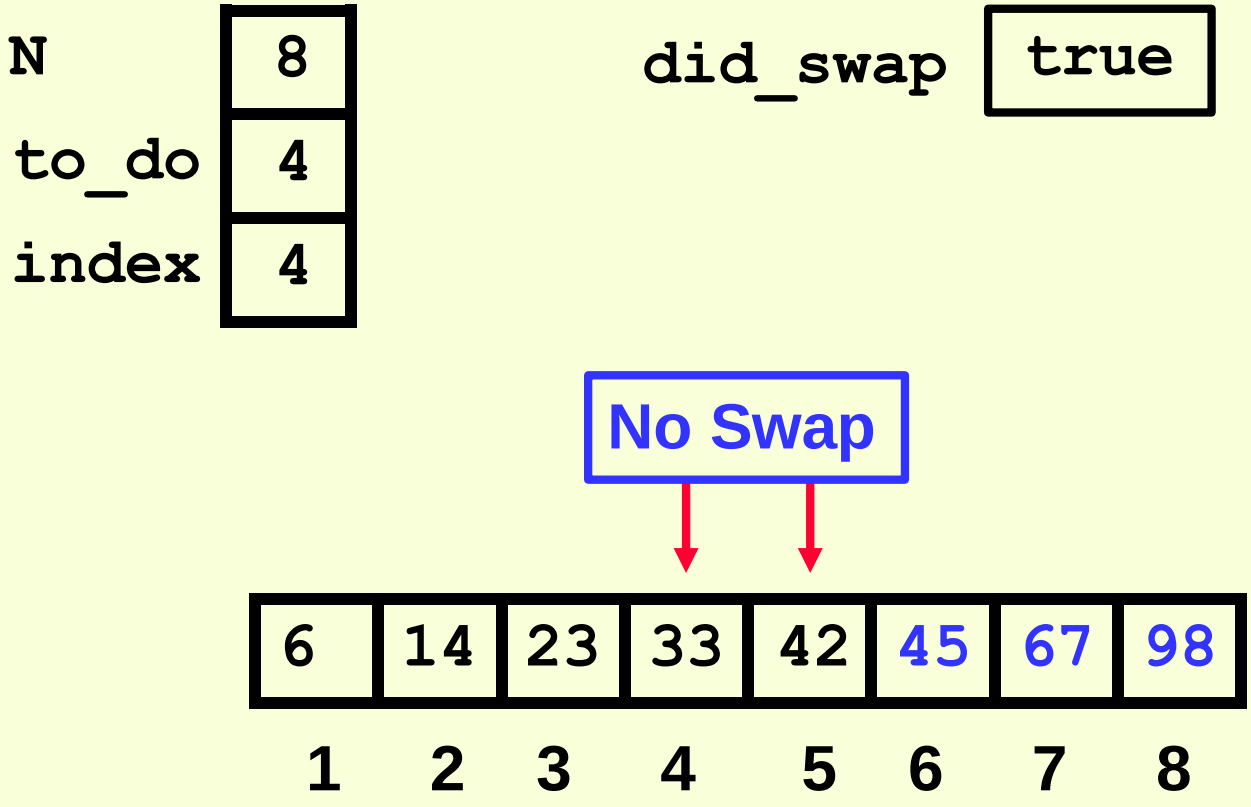
The Fourth “Bubble Up”

N	8
to_do	4
index	4

did_swap	true



The Fourth “Bubble Up”

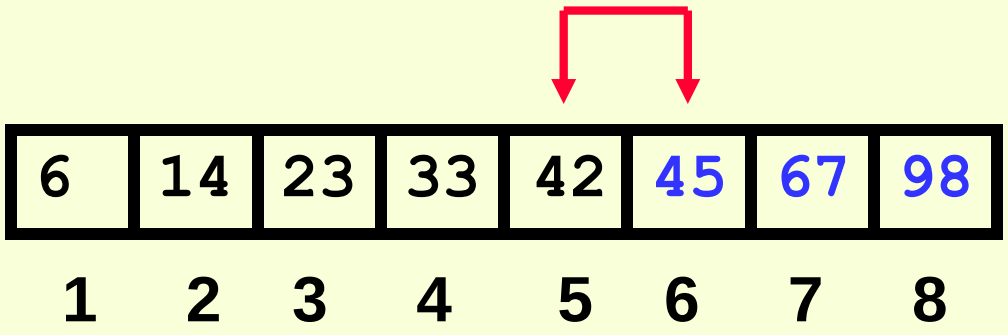


After Fourth Pass of Outer Loop

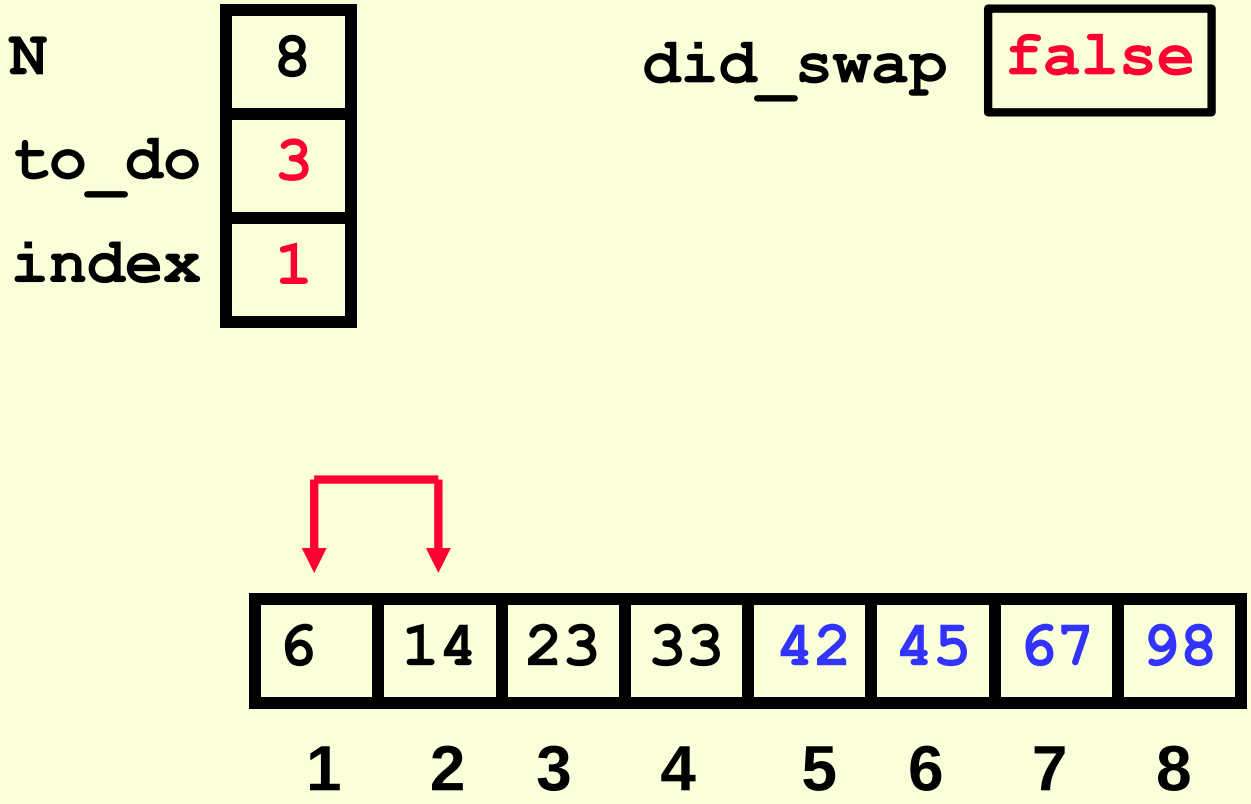
N	8
to_do	4
index	5

did_swap | true

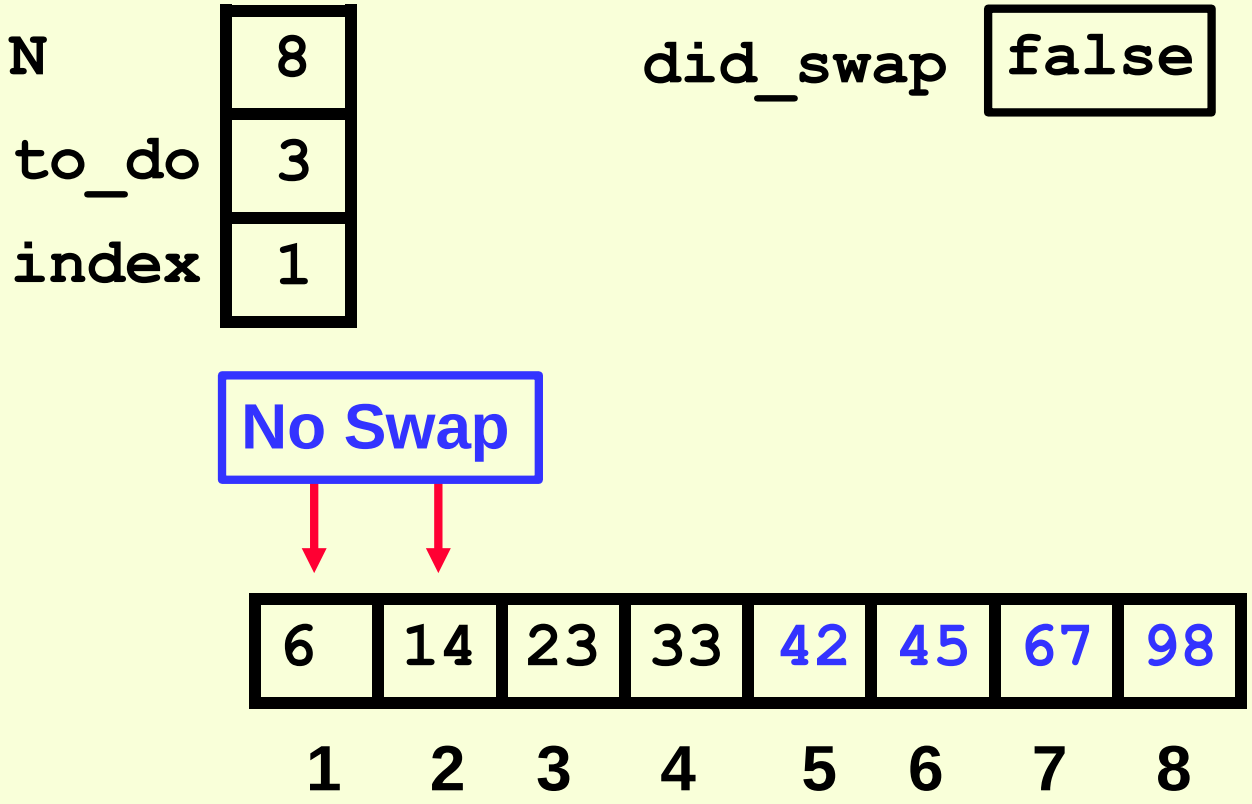
Finished fourth “Bubble Up”



The Fifth “Bubble Up”



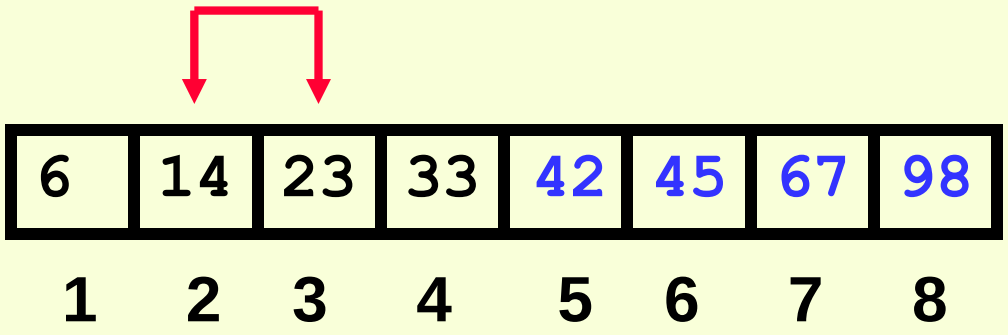
The Fifth “Bubble Up”



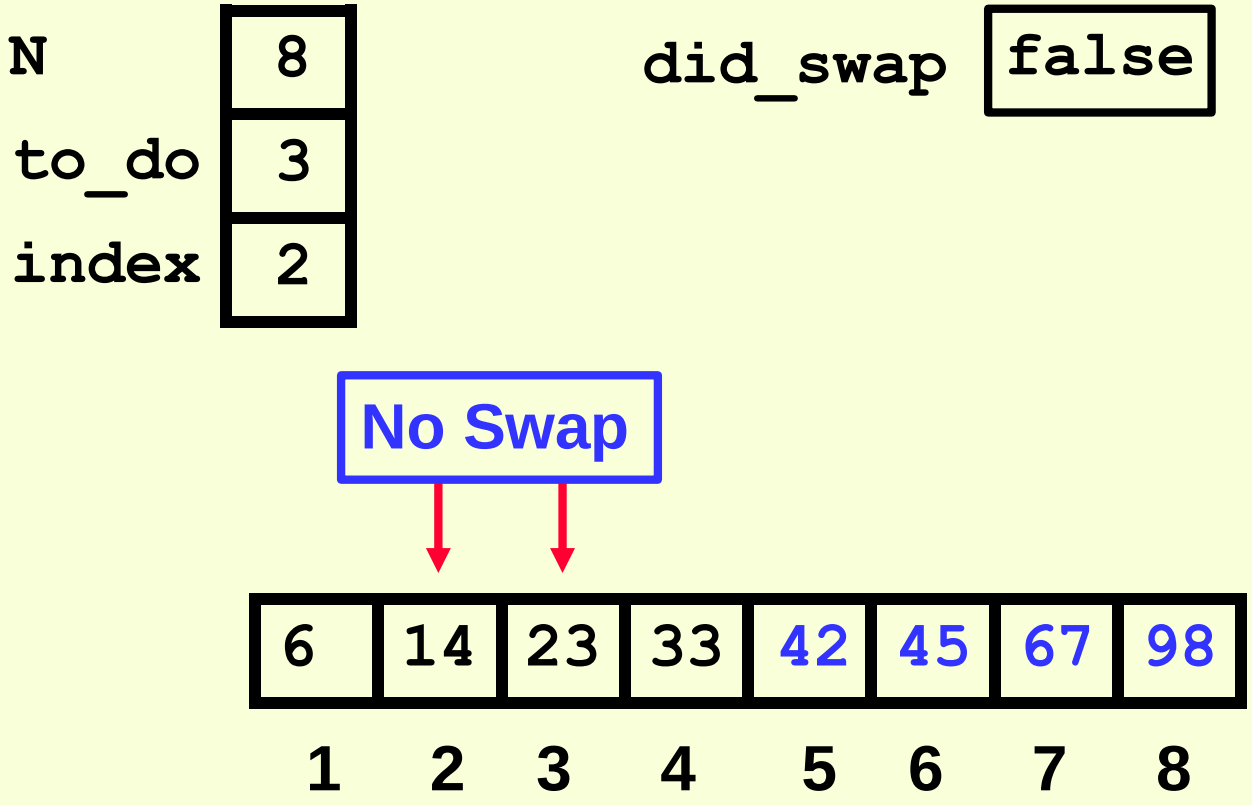
The Fifth “Bubble Up”

N	8
to_do	3
index	2

did_swap | false



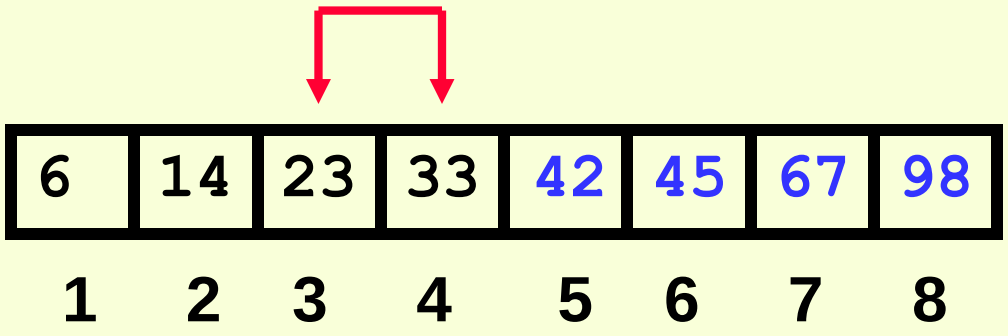
The Fifth “Bubble Up”



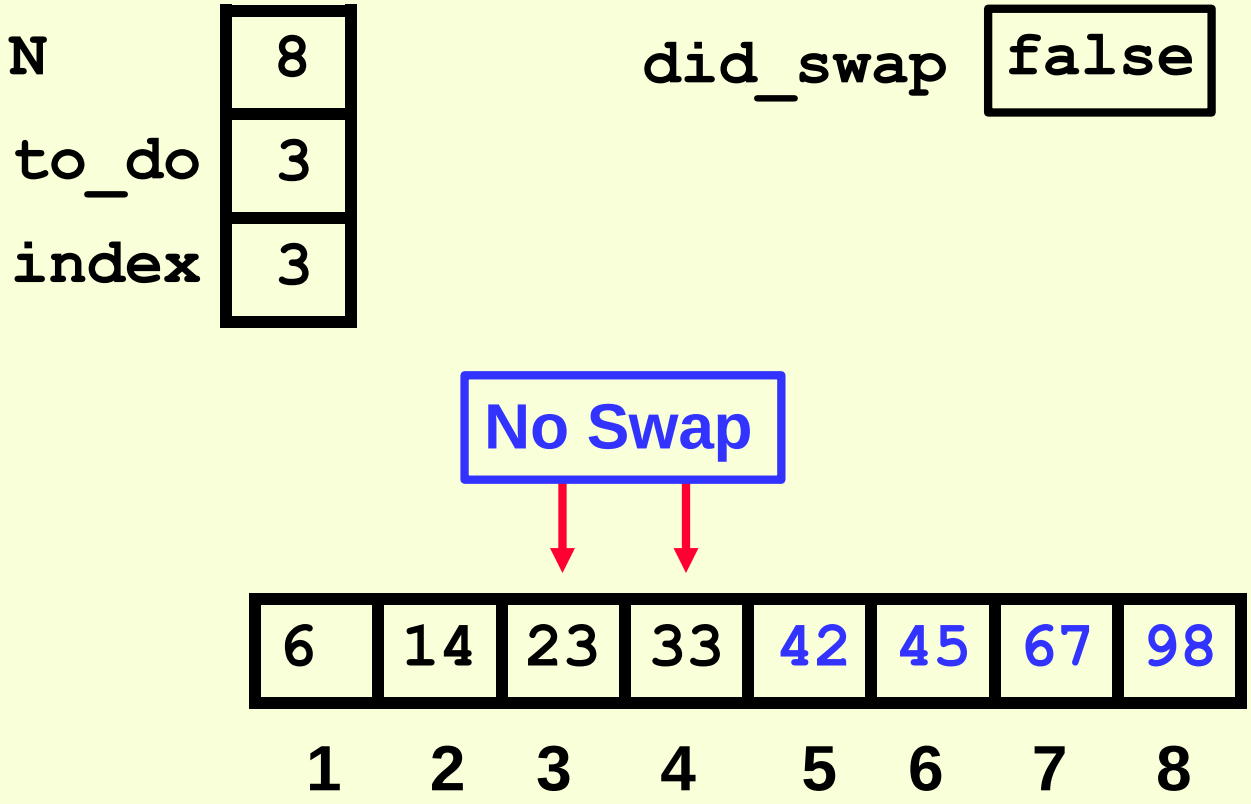
The Fifth “Bubble Up”

N	8
to_do	3
index	3

did_swap	false



The Fifth “Bubble Up”

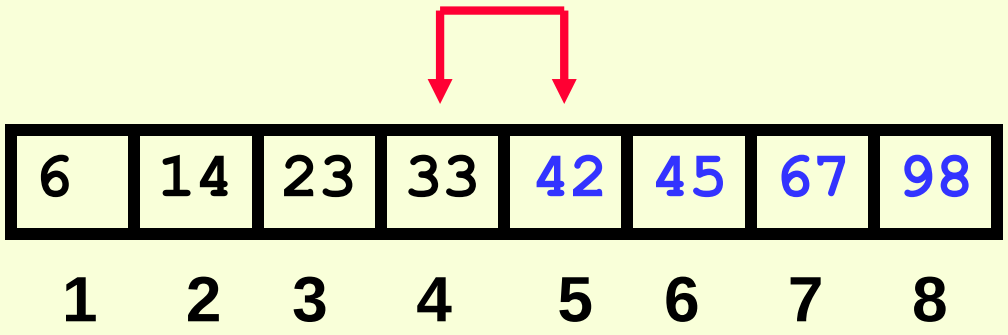


After Fifth Pass of Outer Loop

N	8
to_do	3
index	4

did_swap | false

Finished fifth “Bubble Up”



Finished “Early”

N	8
to_do	3
index	4

did_swap | false

We didn’t do any swapping, so all of the other elements must be correctly placed.

We can “skip” the last two passes of the outer loop.

6	14	23	33	42	45	67	98
1	2	3	4	5	6	7	8

Summary

- **"Bubble Up" algorithm will move largest value to its correct location (to the right)**
- **Repeat "Bubble Up" until all elements are correctly placed:**
  - **Maximum of N-1 times**
  - **Can finish early if no swapping occurs**
- **We reduce the number of elements we compare each time one is correctly placed**

LB

Truth in CS Act

- Nobody ever uses bubble sort
- Nobody
- Not ever
- Because it is extremely inefficient

Questions?

جامعة شط العرب كلية
العلوم
قسم علوم الحاسبات

المرحلة الثالثة
مادة هياكل بيانات2
أستاذ المادة: م.م مصطفى صبحي سلمان

Quicksort

Quicksort Algorithm

Given an array of *n* elements (e.g., integers):

- If array only contains one element, return
- Else
  - pick one element to use as *pivot.*
  - Partition elements into two sub-arrays:
    - Elements less than or equal to pivot
    - Elements greater than pivot
  - Quicksort two sub-arrays
  - Return results

Example

We are given array of n integers to sort:

40	20	10	80	60	50	7	30	100

Pick Pivot Element

There are a number of ways to pick the pivot element. In this example, we will use the first element in the array:

40	20	10	80	60	50	7	30	100

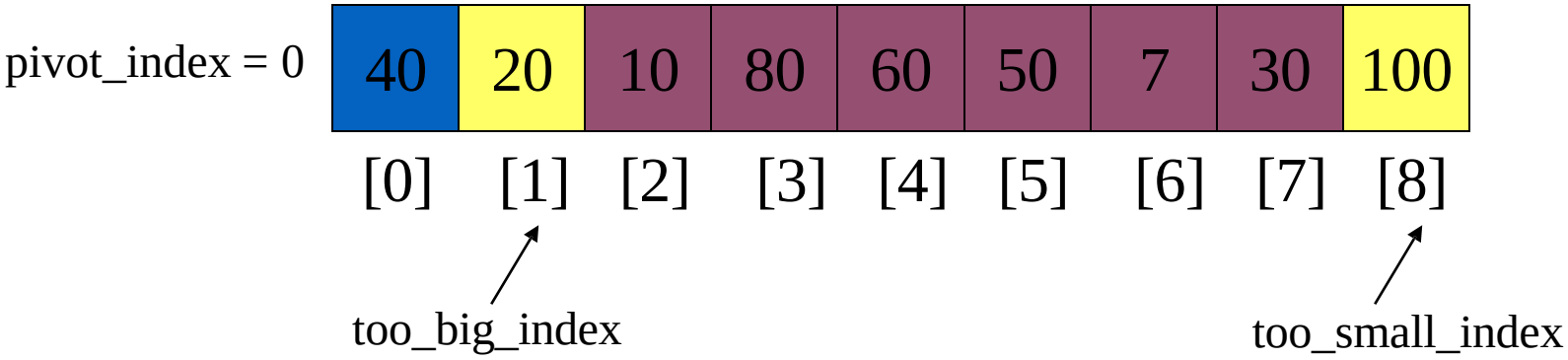
Partitioning Array

Given a pivot, partition the elements of the array such that the resulting array consists of:

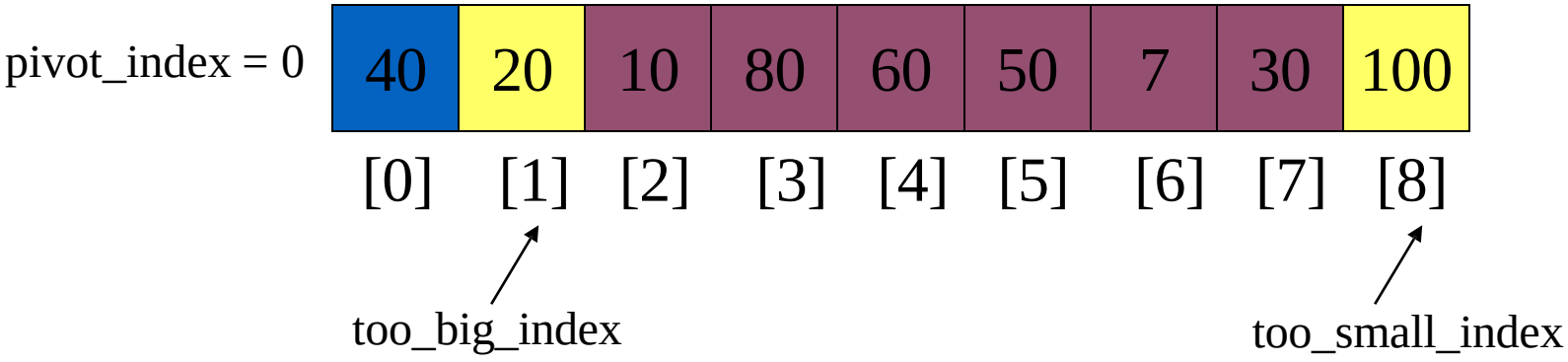
1. One sub-array that contains elements >= pivot
2. Another sub-array that contains elements < pivot

The sub-arrays are stored in the original data array.

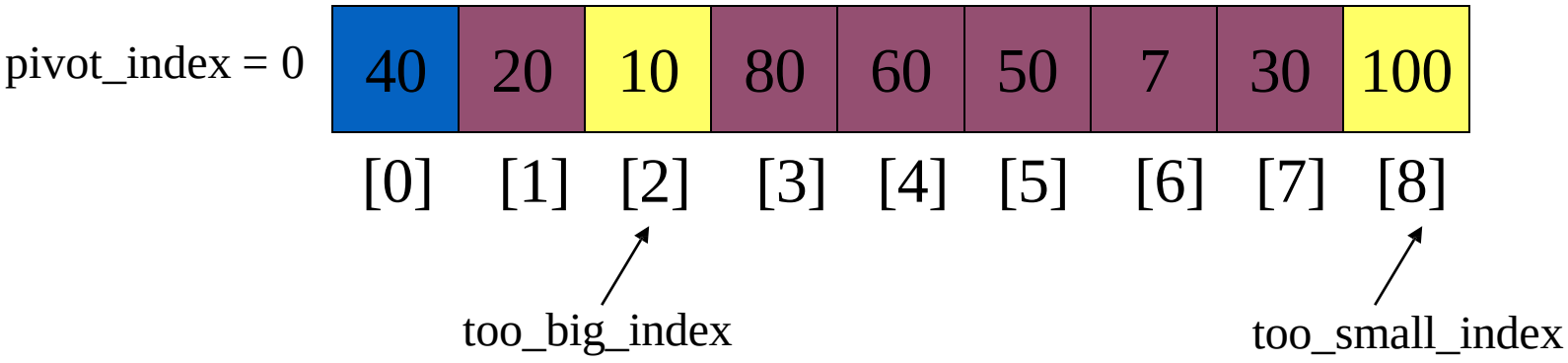
Partitioning loops through, swapping elements below/above pivot.


pivot_index = 0
40
20
10
80
60
50
7
30
100
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
too_big_index
too_small_index

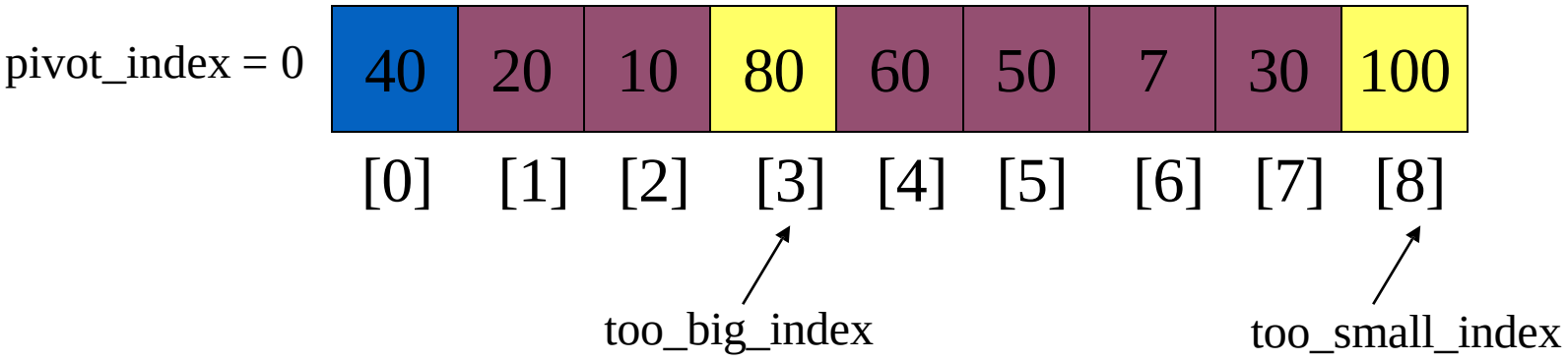
1. While data[too_big_index] <= data[pivot]
   ++too_big_index



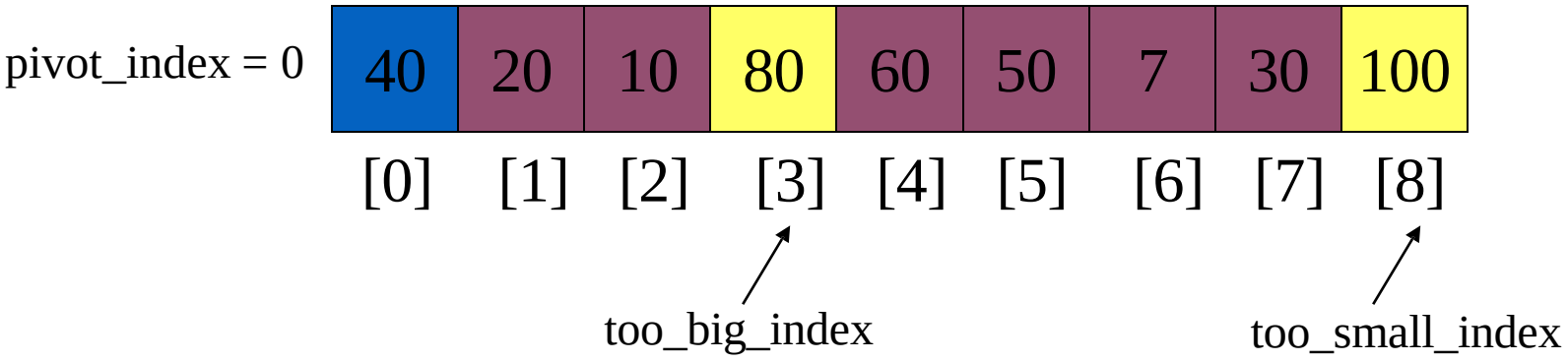
1. While data[too_big_index] <= data[pivot]
   ++too_big_index



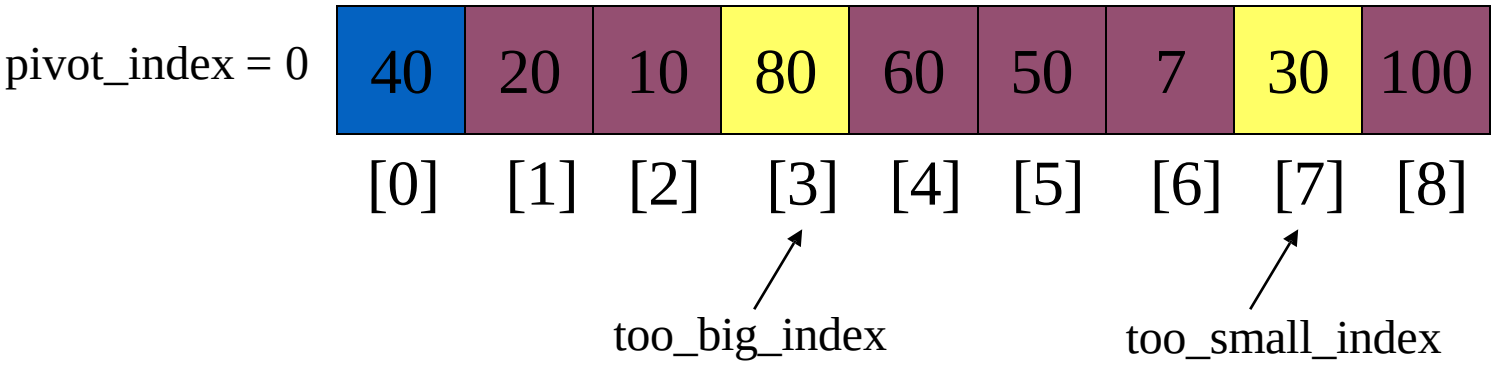
1. While data[too_big_index] <= data[pivot]
   ++too_big_index



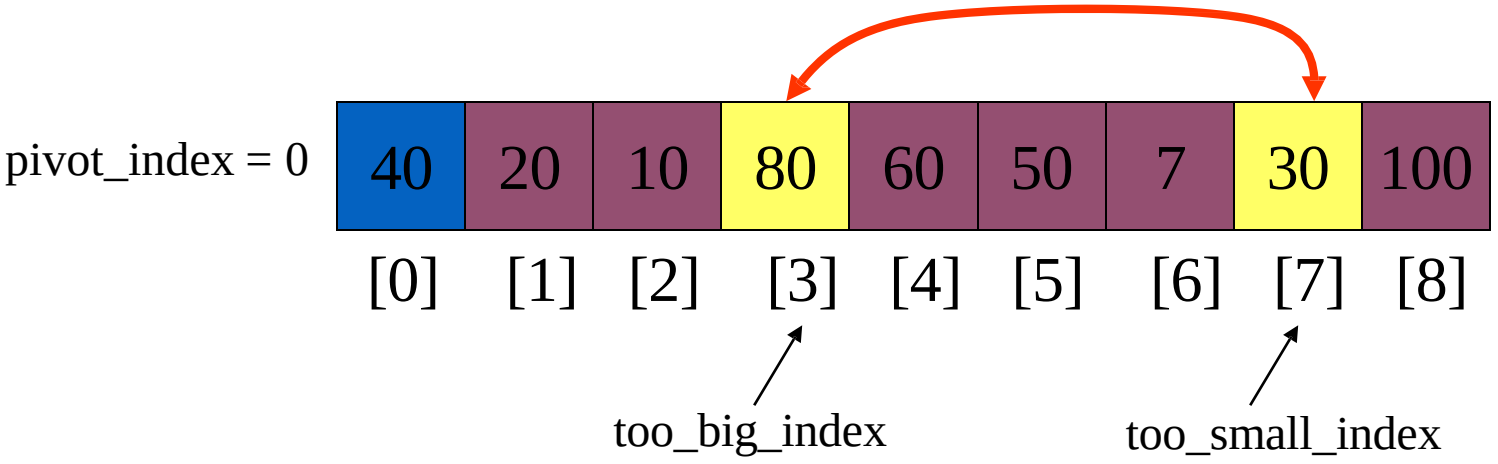
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index



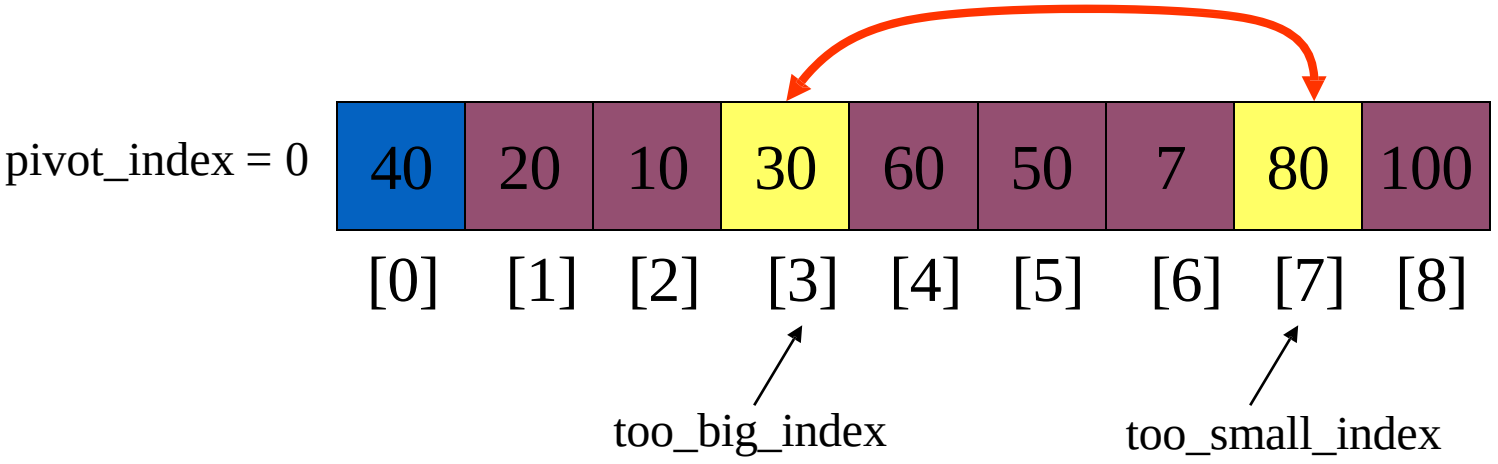
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index



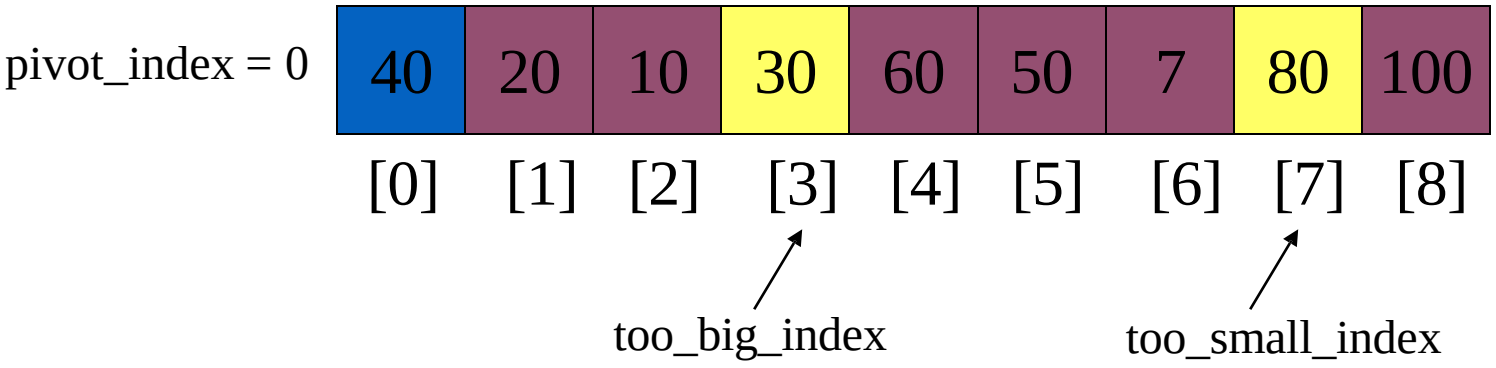
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]



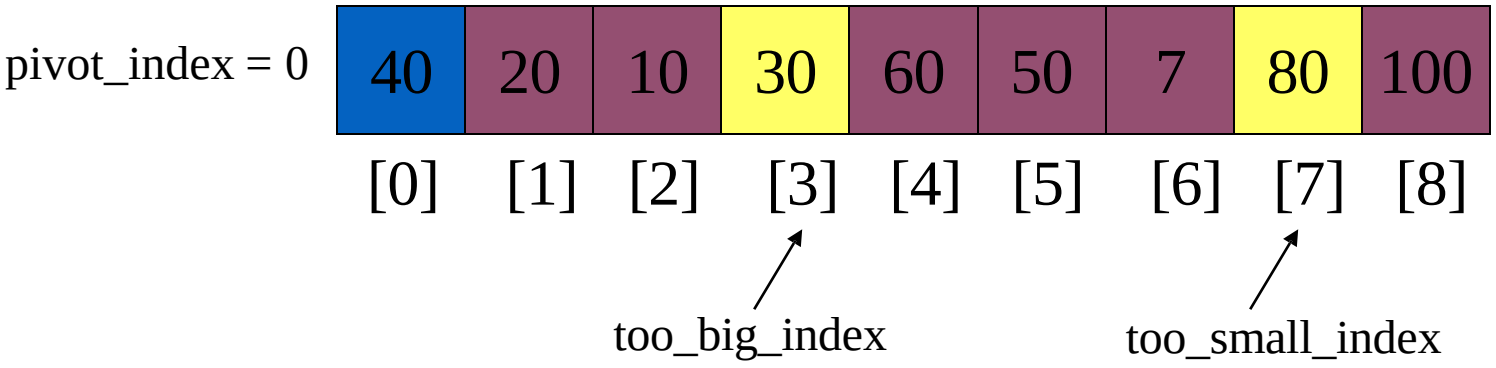
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]



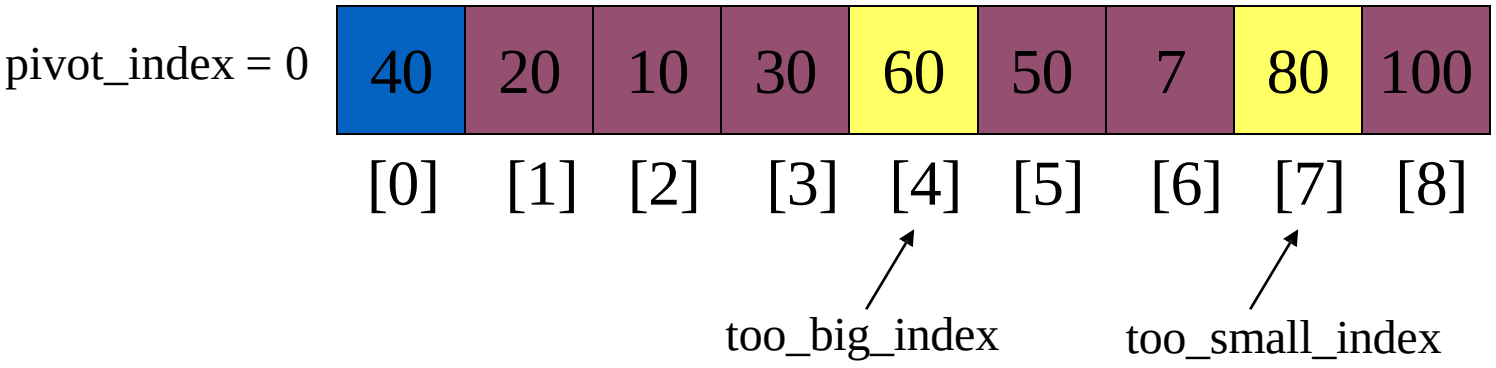
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



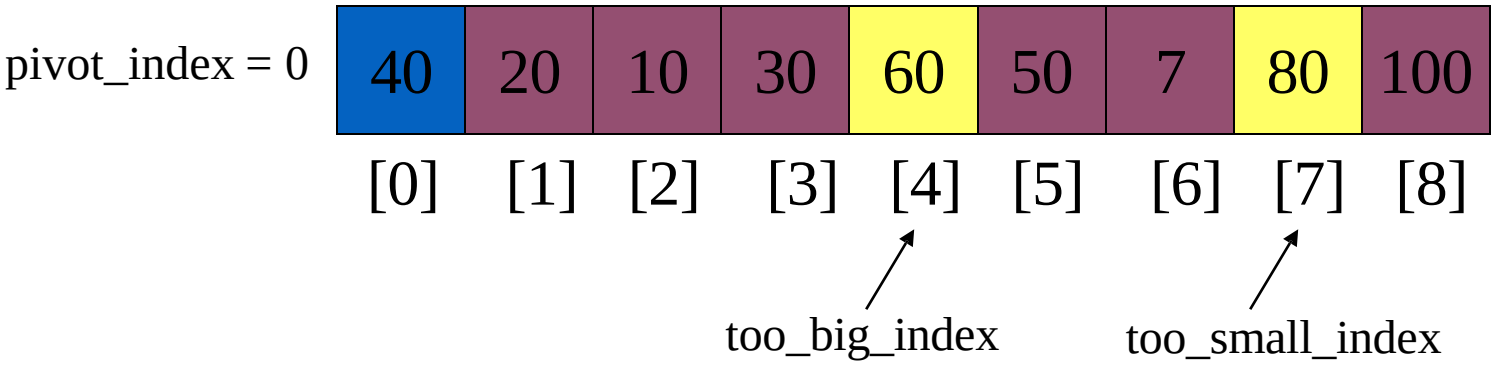
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



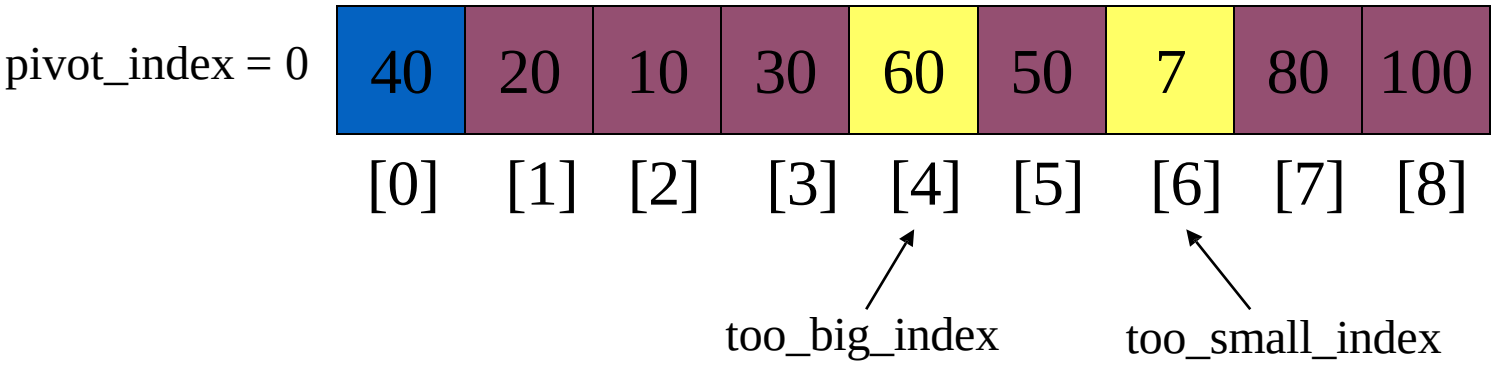
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



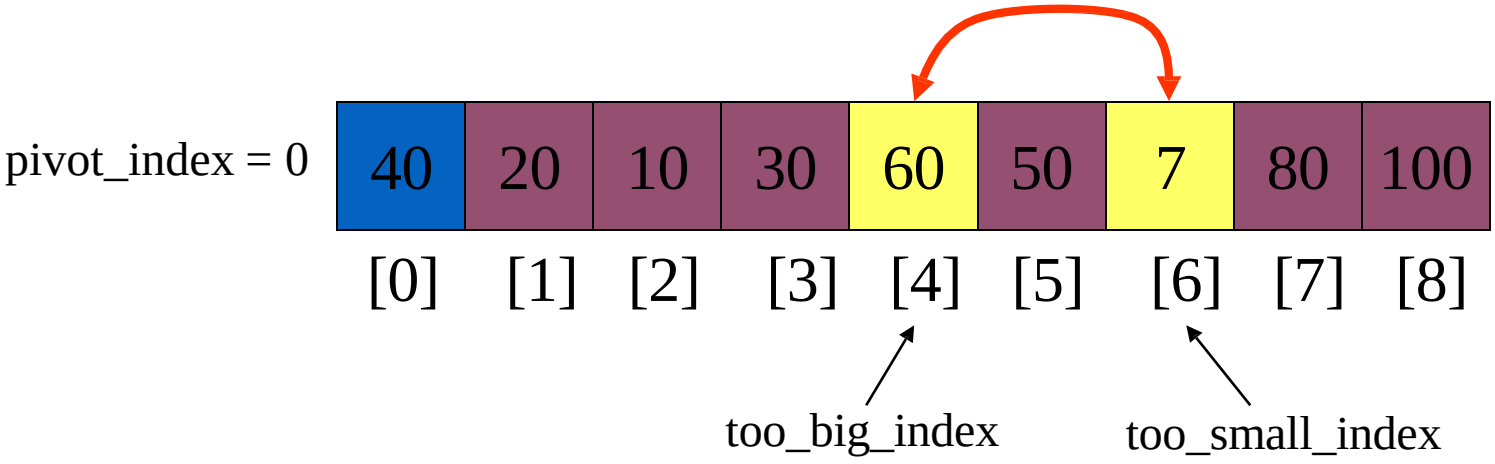
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



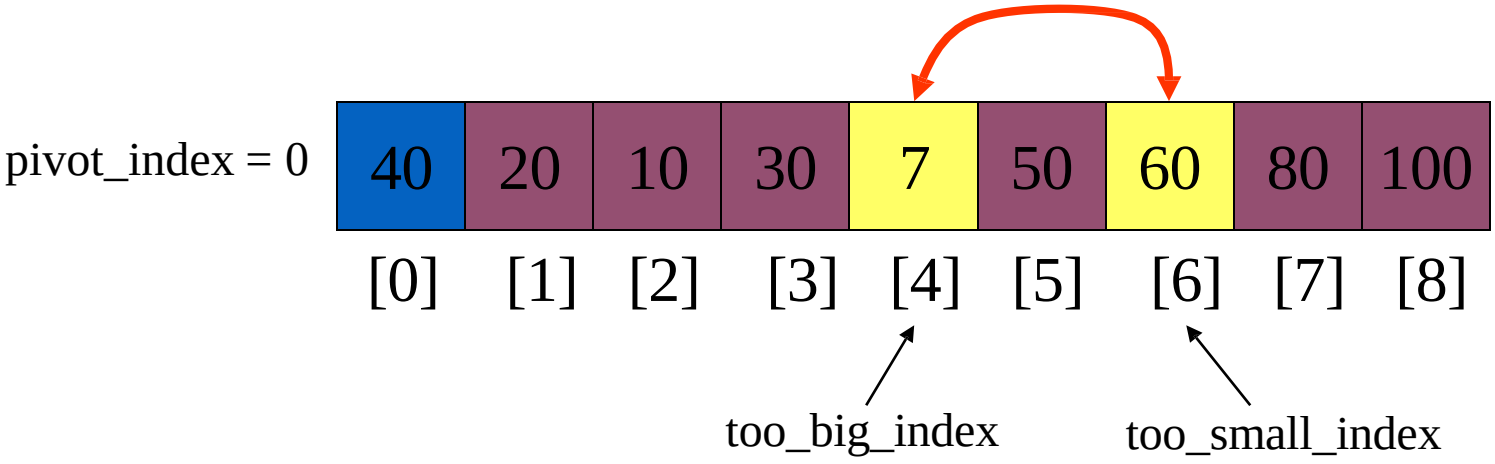
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



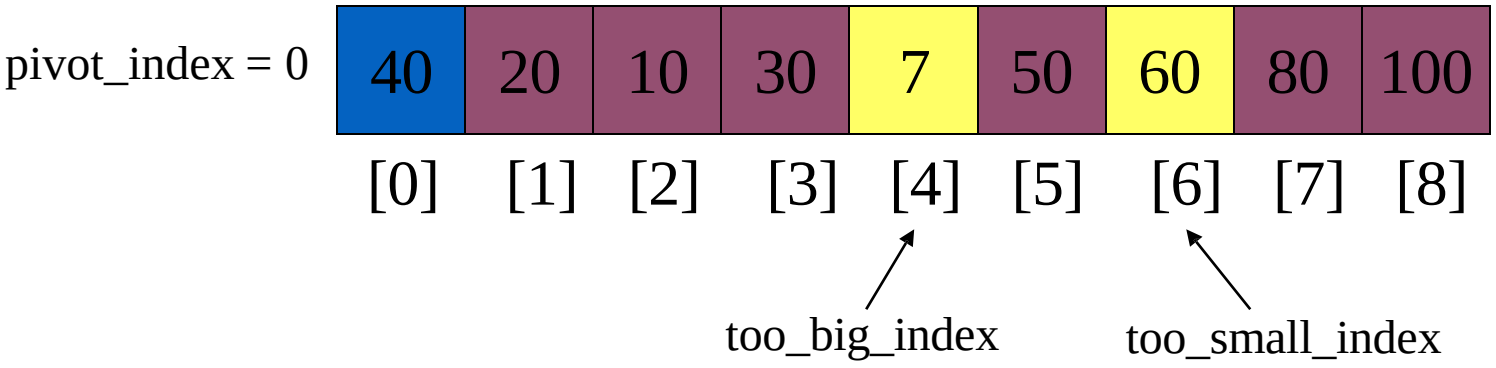
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



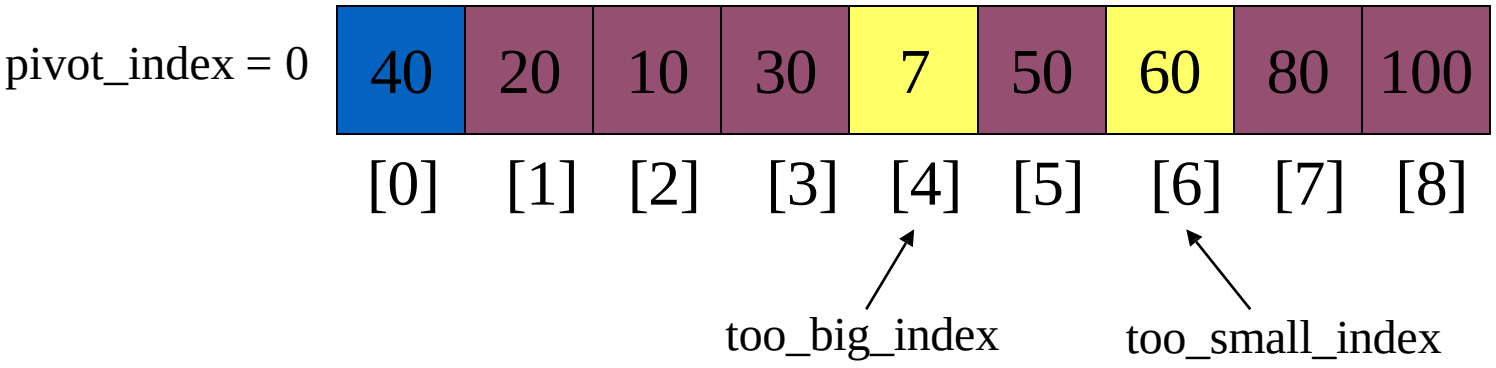
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



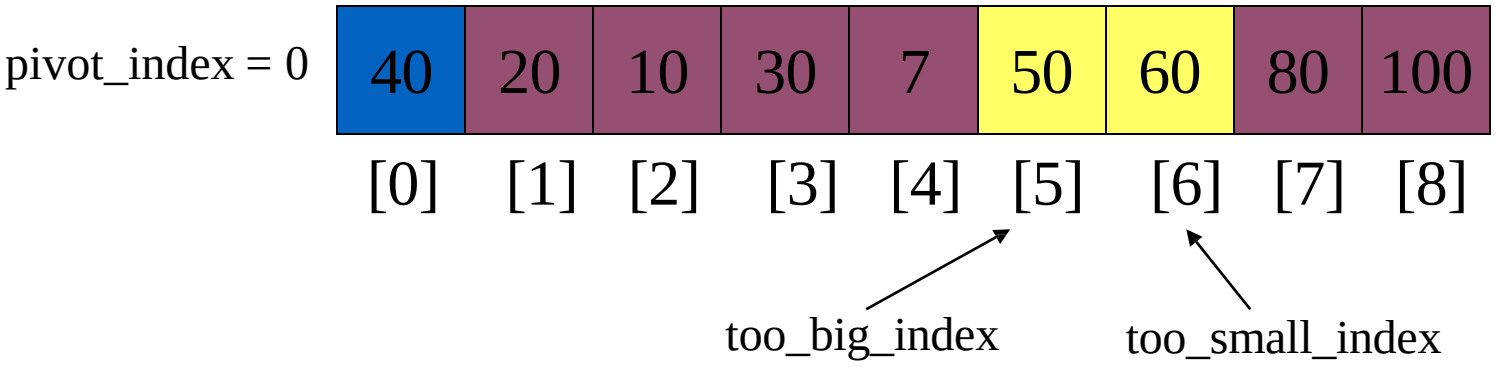
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



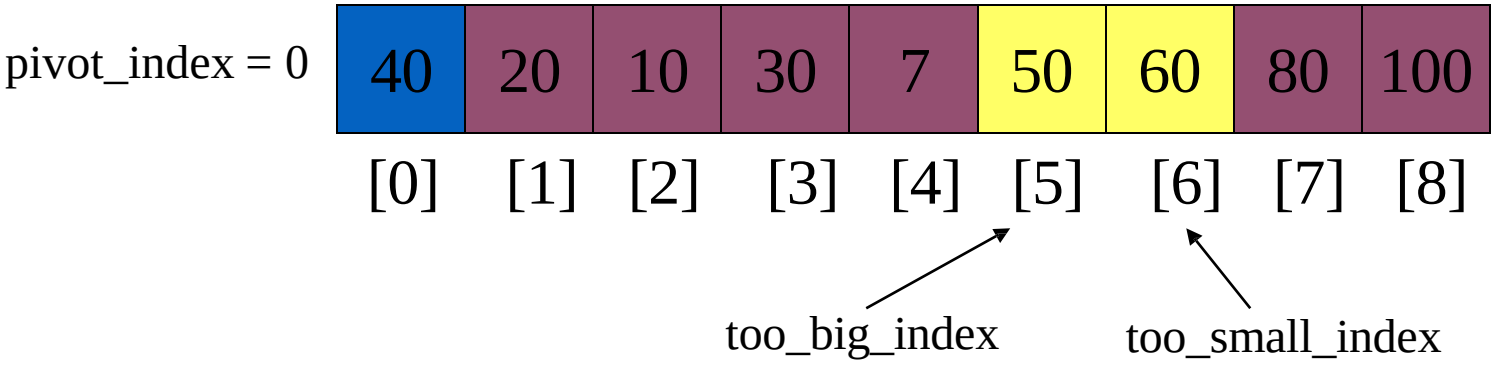
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



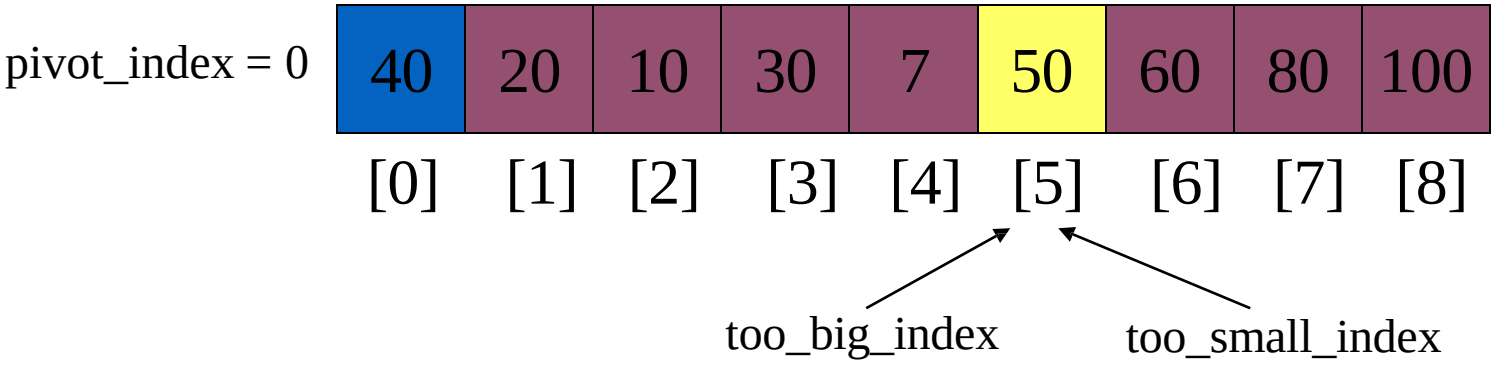
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



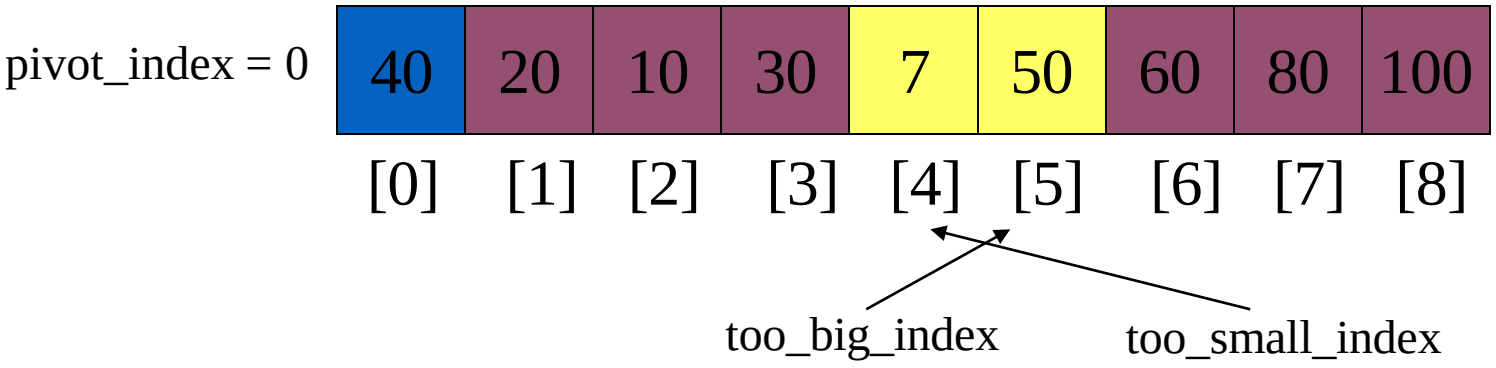
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



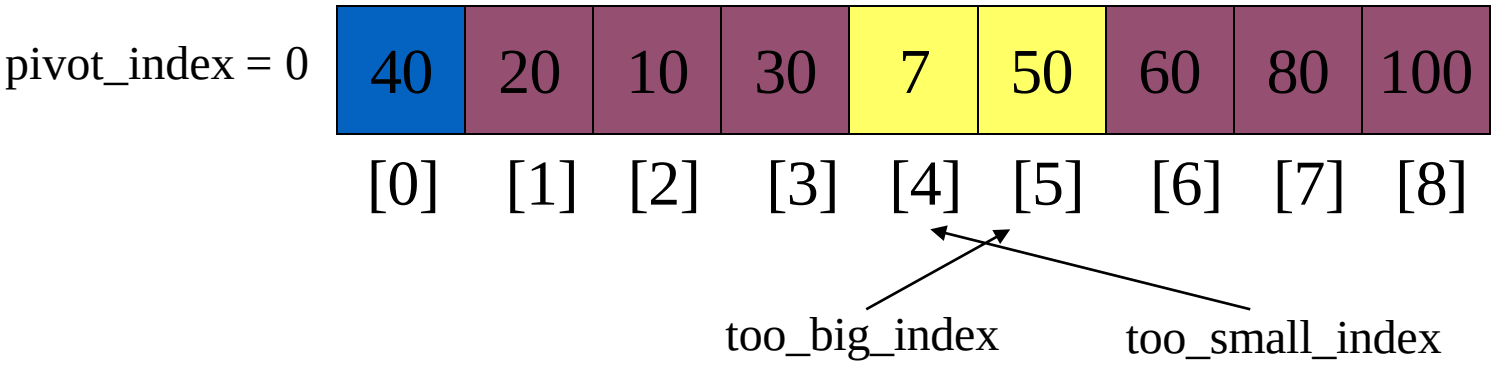
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



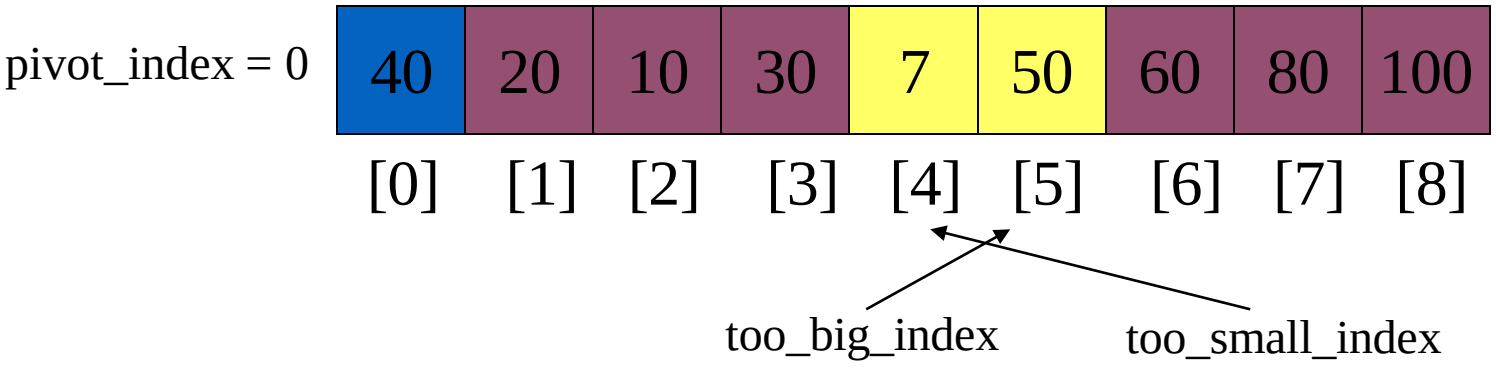
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



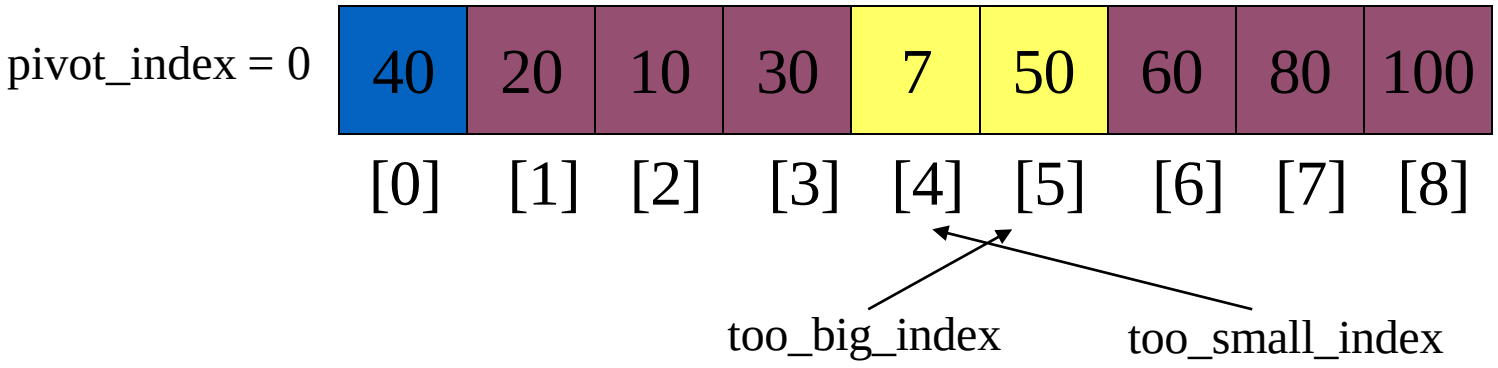
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



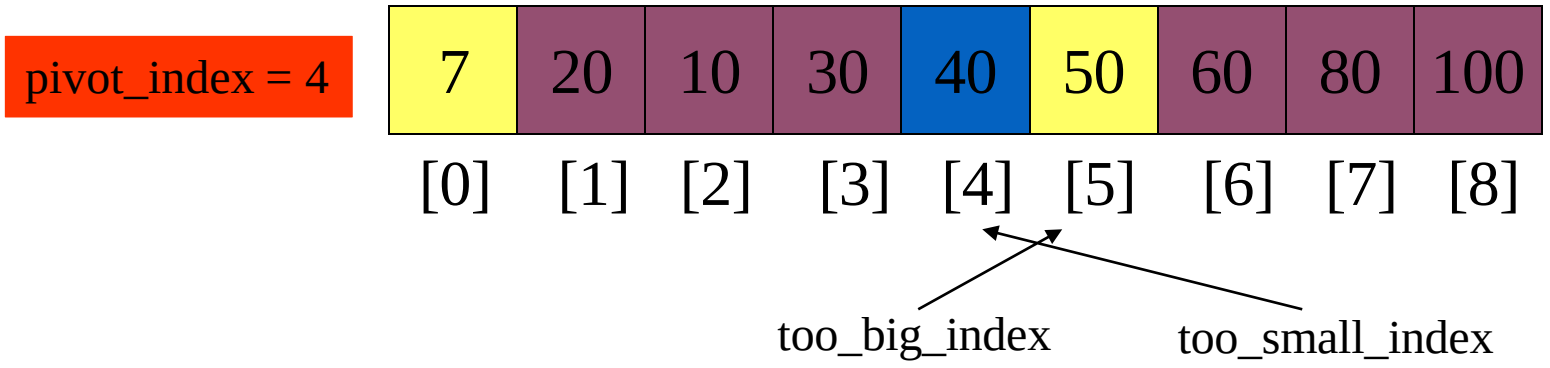
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.



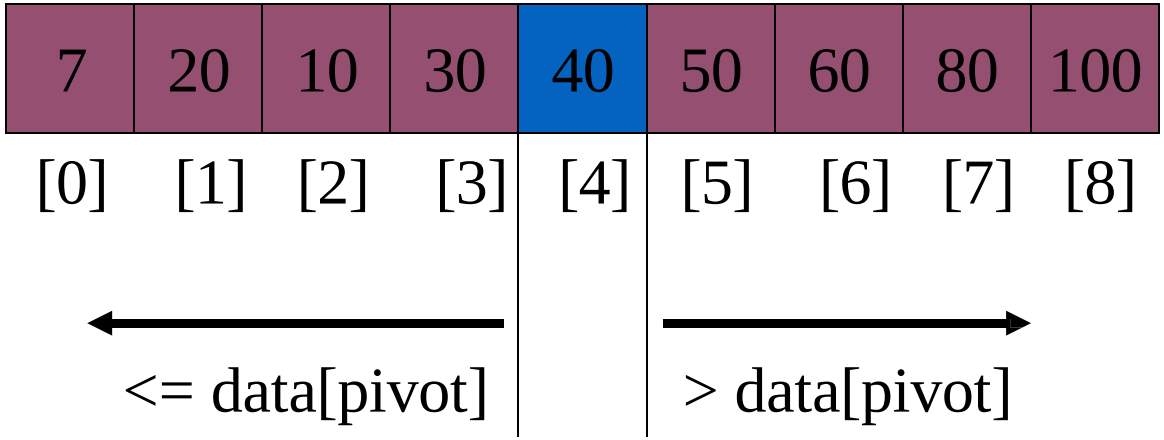
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.
5. Swap data[too_small_index] and data[pivot_index]



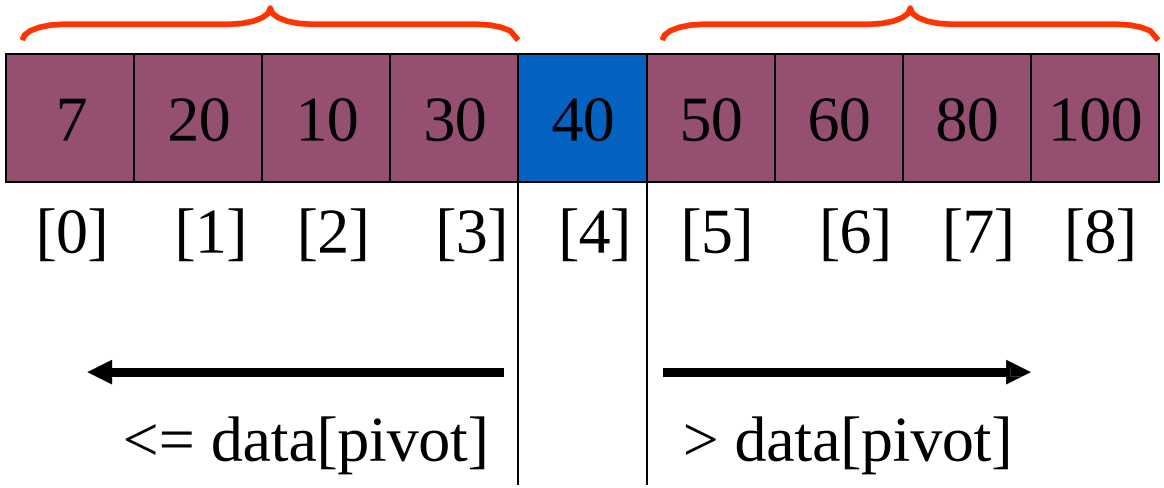
1. While data[too_big_index] <= data[pivot]
   ++too_big_index
2. While data[too_small_index] > data[pivot]
   --too_small_index
3. If too_big_index < too_small_index
   swap data[too_big_index] and data[too_small_index]
4. While too_small_index > too_big_index, go to 1.
5. Swap data[too_small_index] and data[pivot_index]



Partition Result



Recursion: Quicksort Sub-arrays



جامعة شط العرب كلية
العلوم
قسم علوم الحاسبات

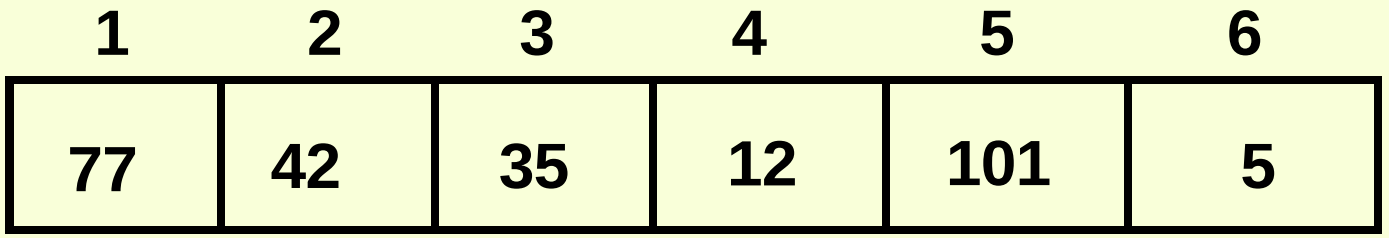
المرحلة الثالثة
مادة هياكل بيانات2
أستاذ المادة: م.م مصطفى صبحي سلمان

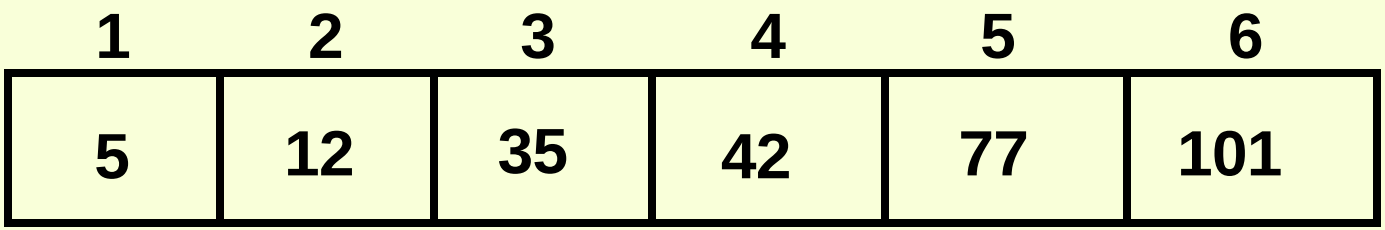
Lecture 2

Merge Sort

Sorting

- **Sorting takes an unordered collection and makes it an ordered one.**





Divide and Conquer

- **Divide and Conquer cuts the problem in half each time, but uses the result of both halves:**
  - **cut the problem in half until the problem is trivial**
  - **solve for both halves**
  - **combine the solutions**

Mergesort

- A divide-and-conquer algorithm:
- Divide the unsorted array into 2 halves until the sub-arrays only contain one element
- Merge the sub-problem solutions together:
  - Compare the sub-array's first elements
  - Remove the smallest element and put it into the result array
  - Continue the process until all elements have been put into the result array

37	23	6	89	15	12	2	19

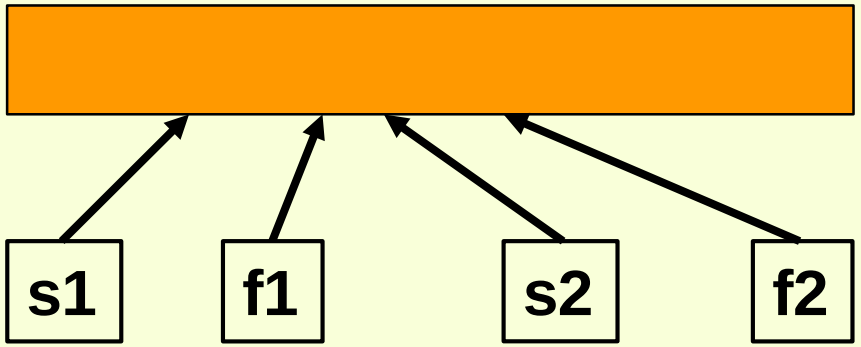
Algorithm

```
Mergesort(Passed an array)
  if array size > 1
    Divide array in half
    Call Mergesort on first half.
    Call Mergesort on second half.
    Merge two halves.

Merge(Passed two arrays)
  Compare leading element in each array
  Select lower and place in new array.
    (If one input array is empty then place
     remainder of other array in output array)
```

LB

More TRUTH in CS

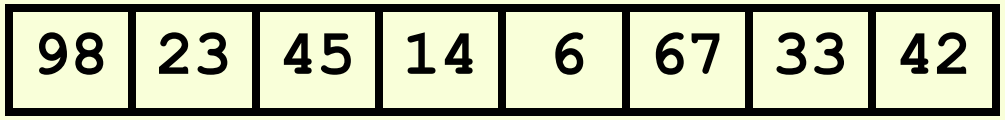
- We don’t really pass in two arrays!
- We pass in one array with indicator variables which tell us where one set of data starts and finishes and where the other set of data starts and finishes.
- Honest.

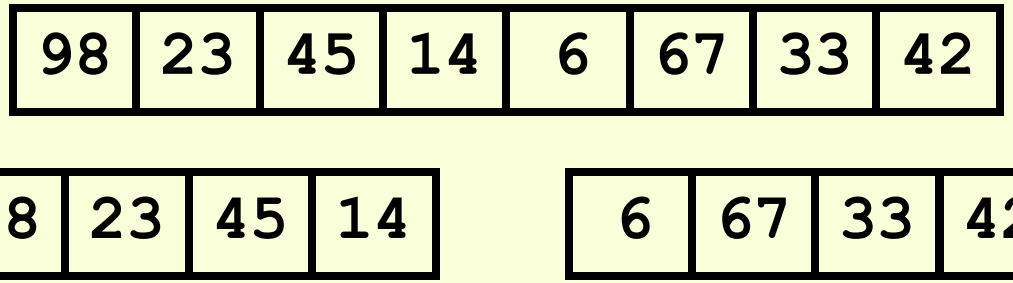


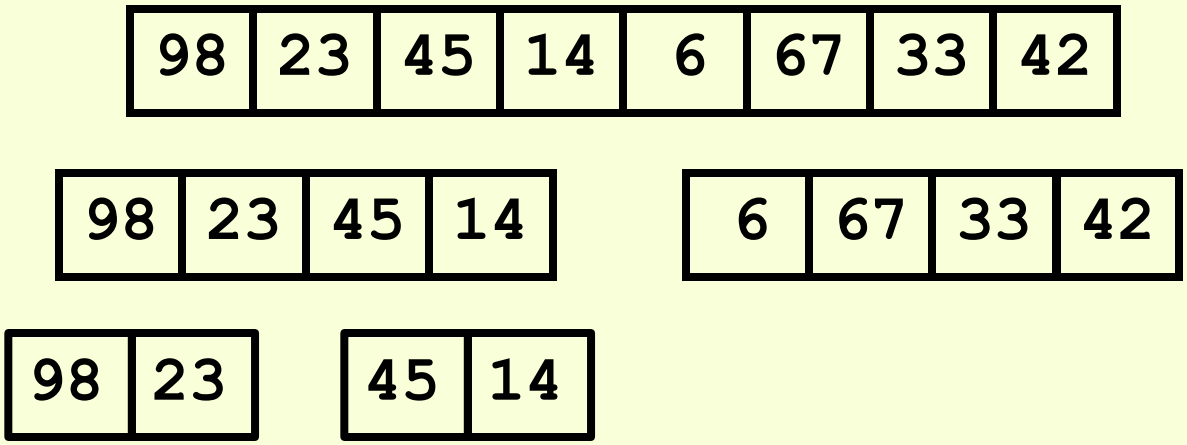
LB

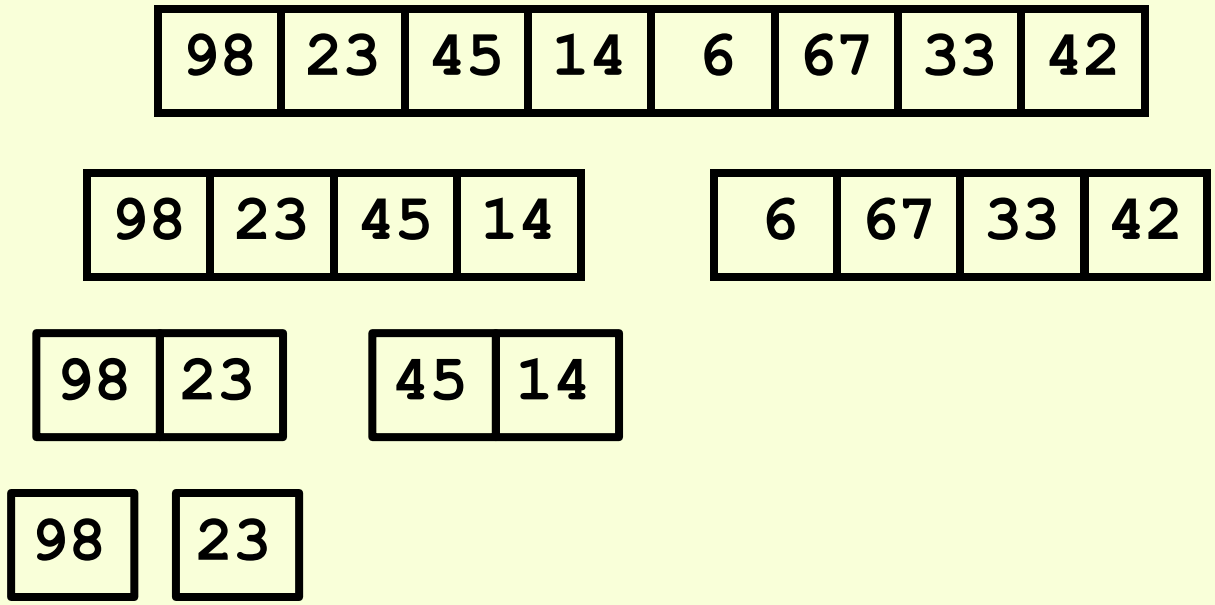
Algorithm

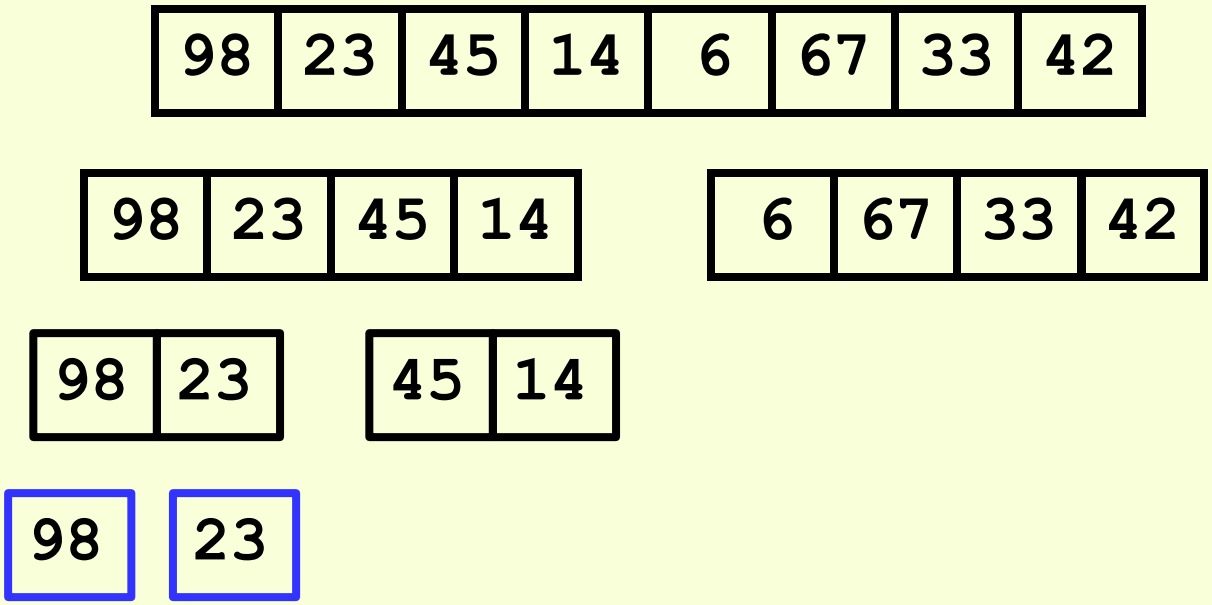
```
Mergesort(Passed an array)
   if array size > 1
      Divide array in half
      Call Mergesort on first half.
      Call Mergesort on second half.
      Merge two halves.

Merge(Passed two arrays)
   Compare leading element in each array
   Select lower and place in new array.
      (If one input array is empty then place
       remainder of other array in output array)
```

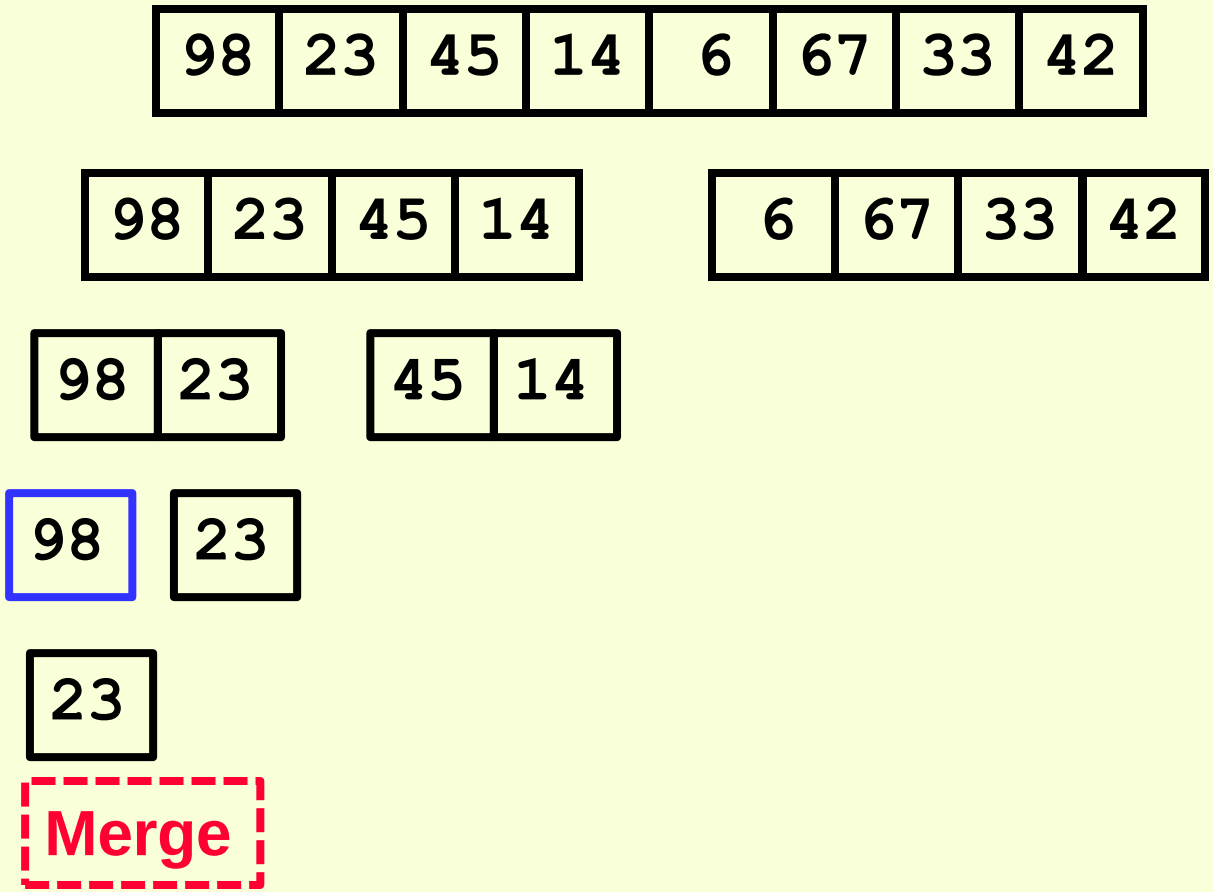

98
23
45
14
6
67
33
42

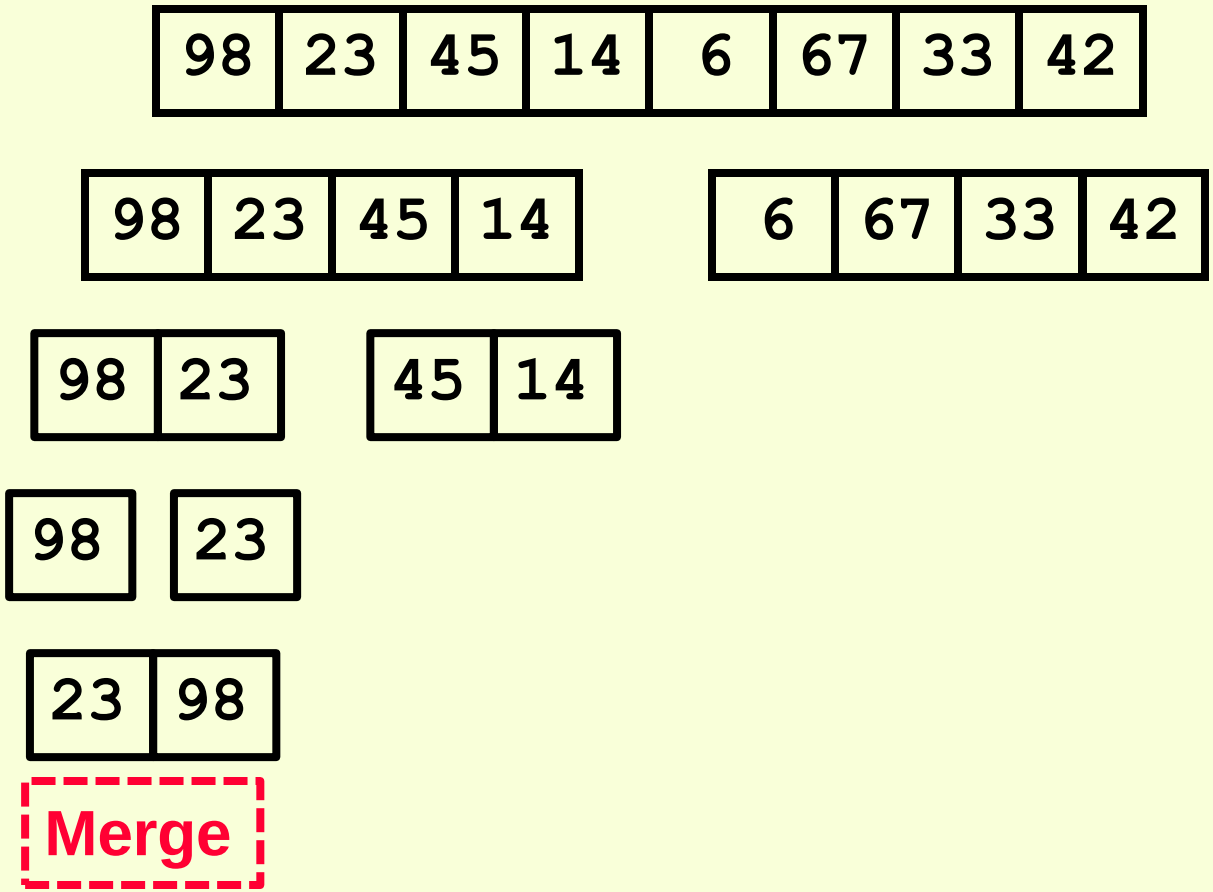

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42

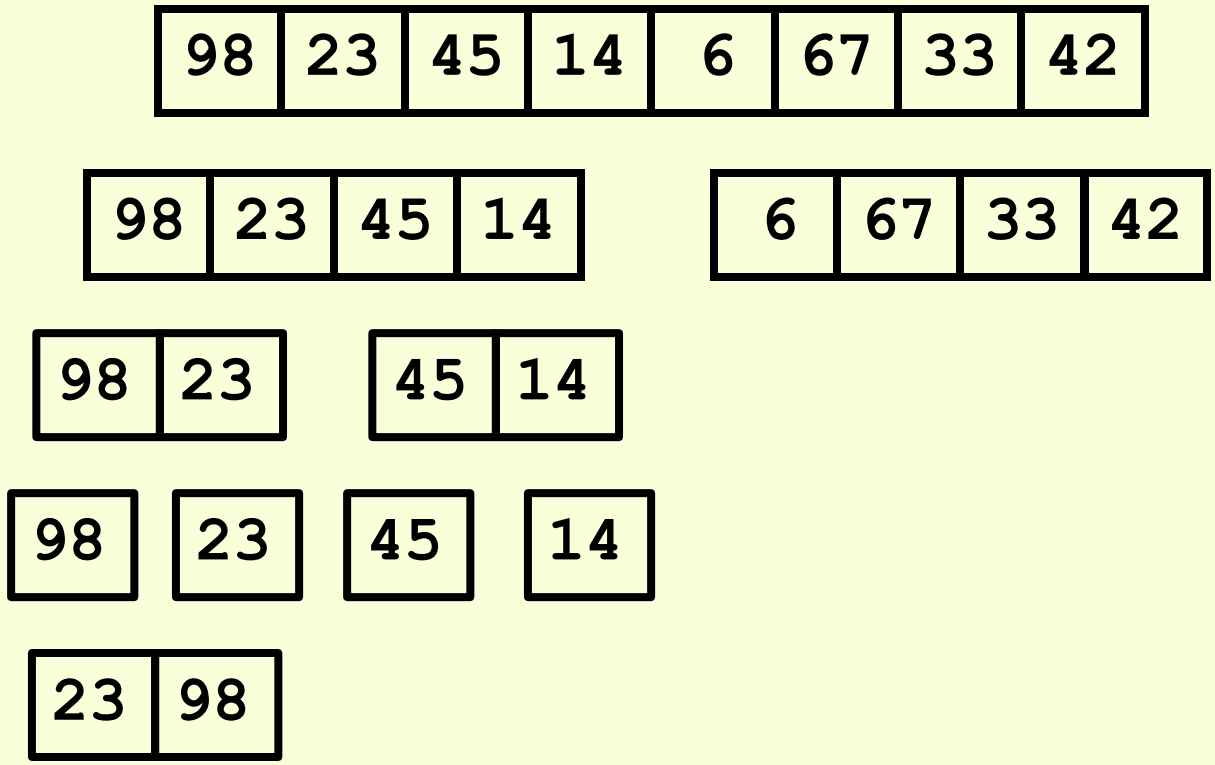

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14

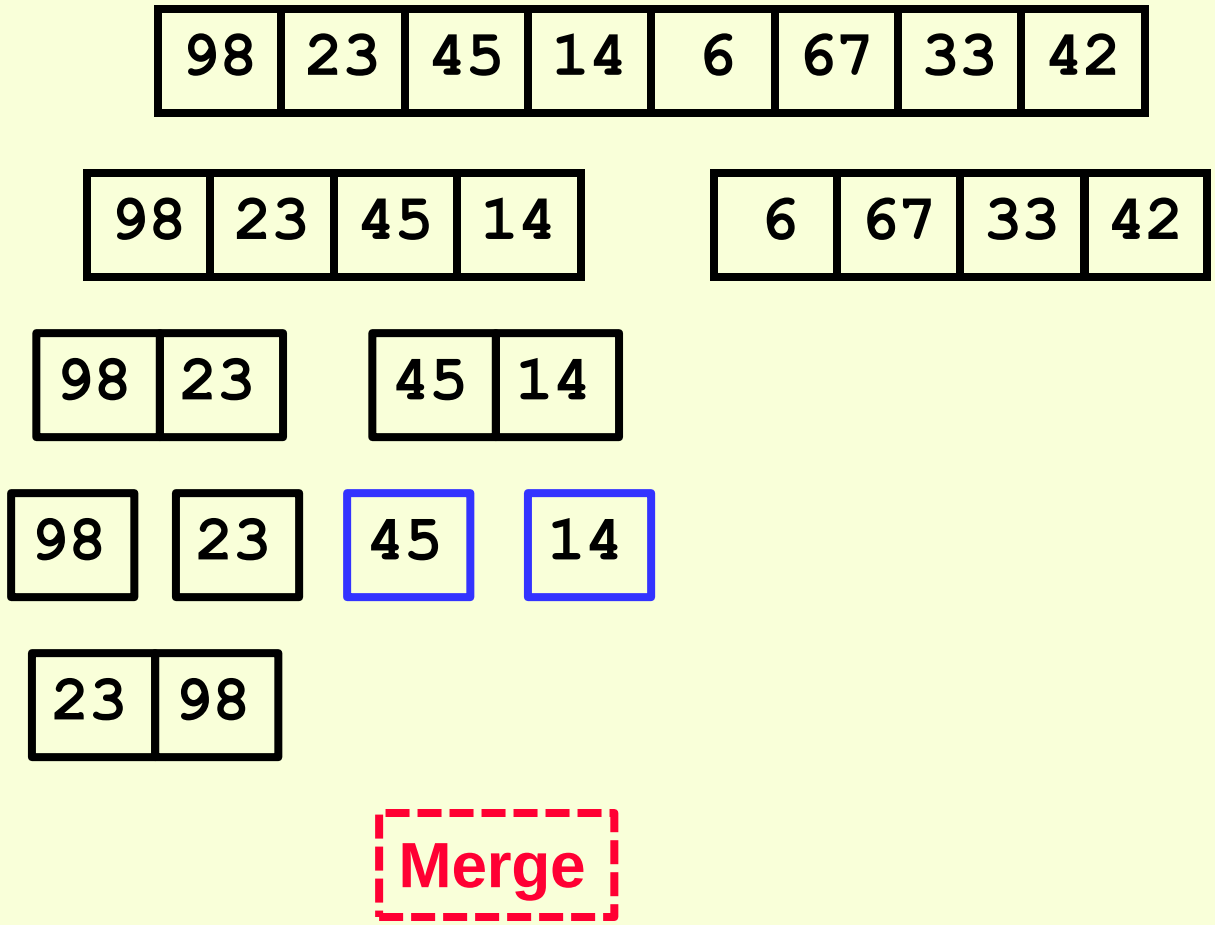

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23

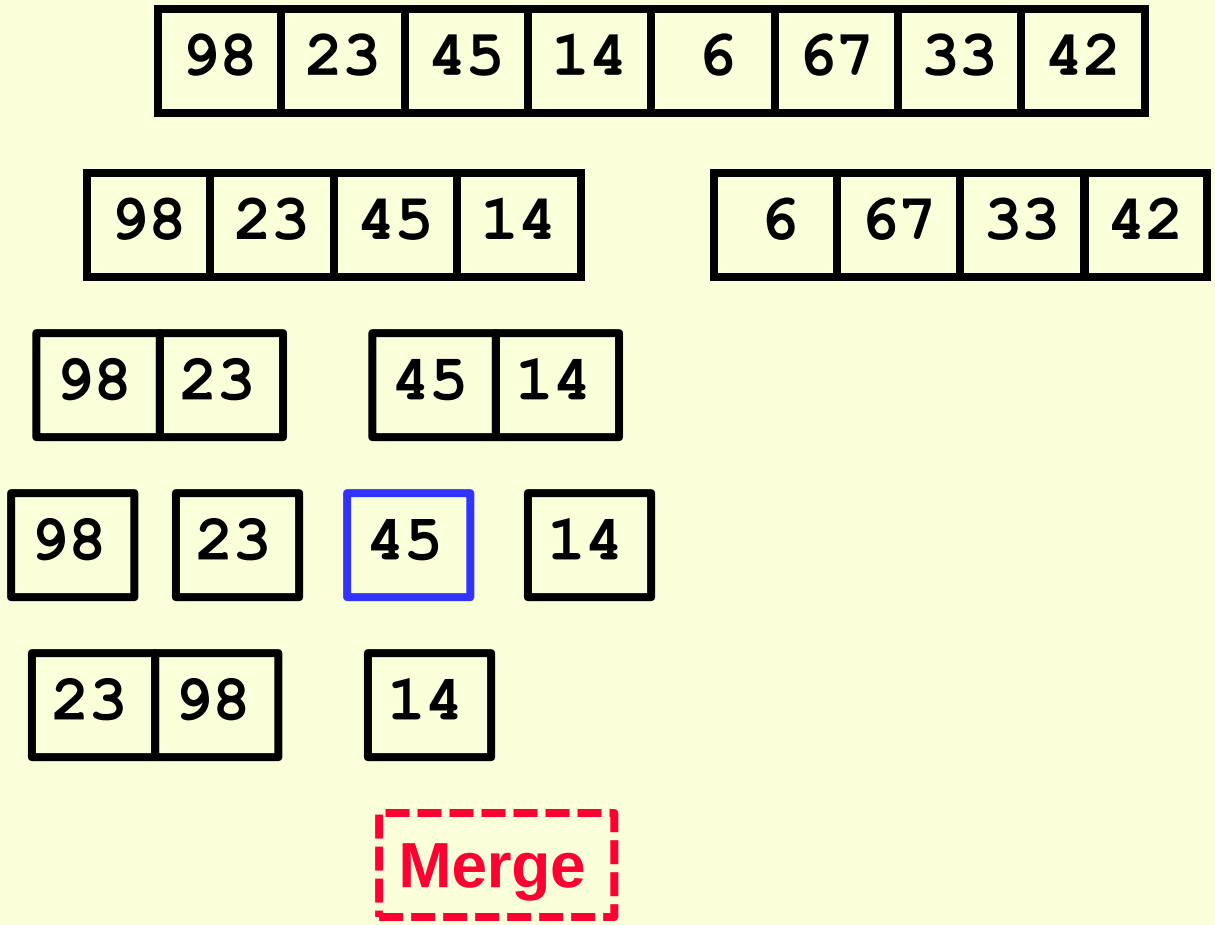

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
Merge

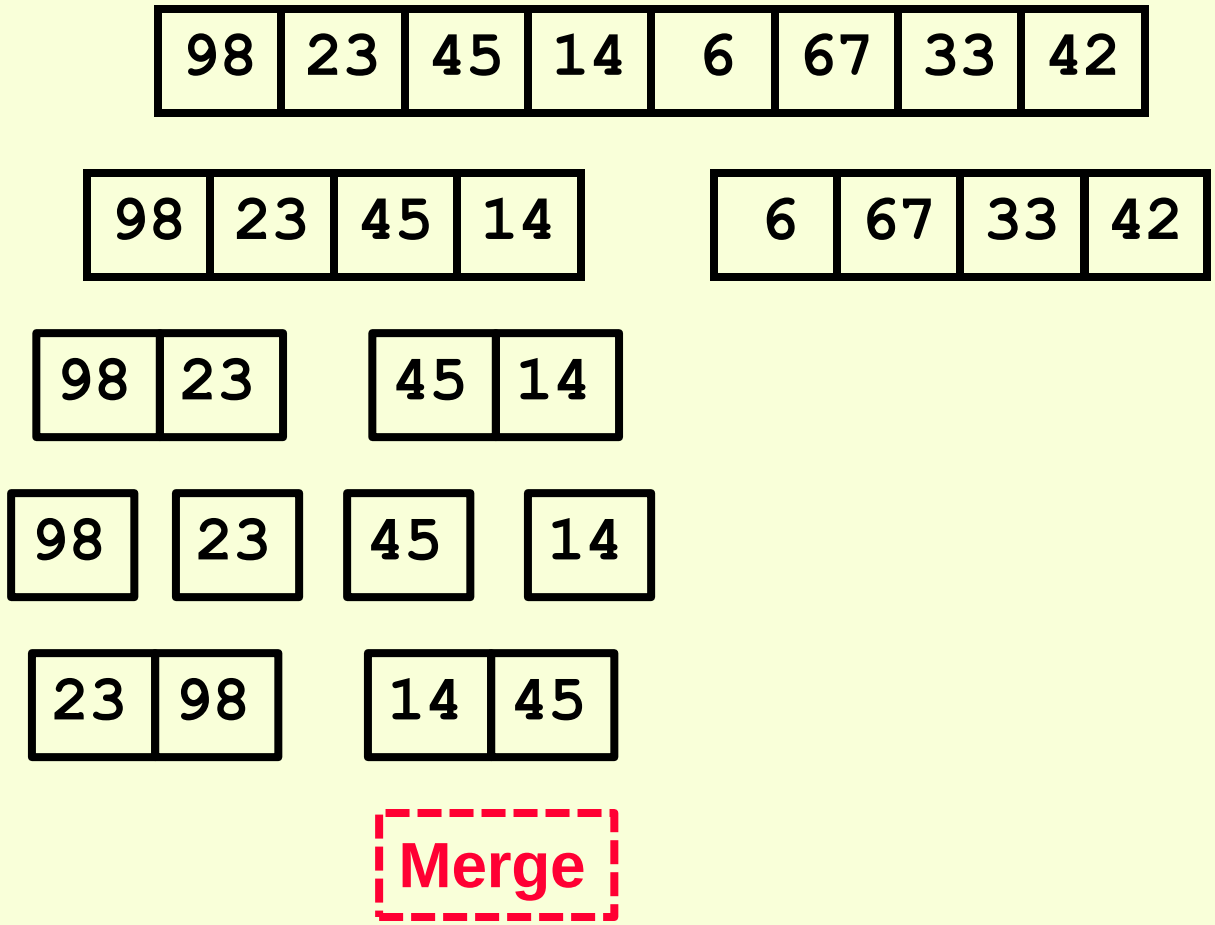

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
23
Merge

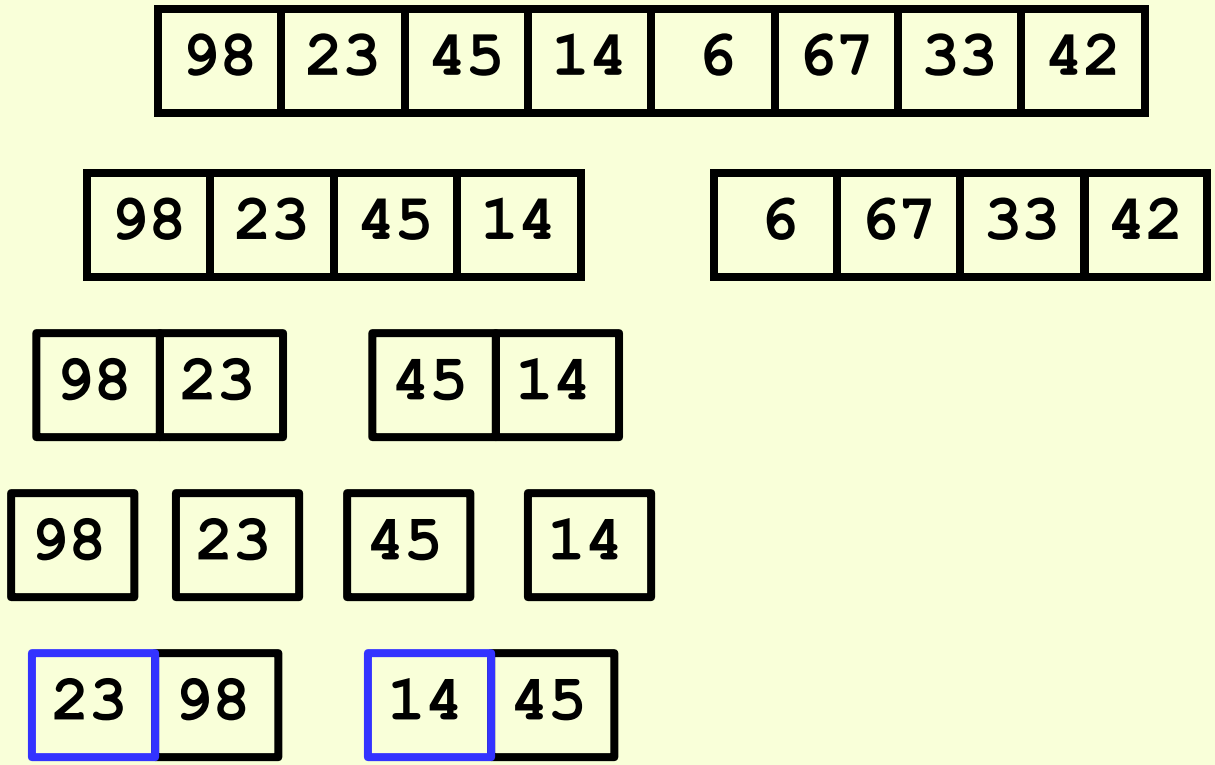

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
23 98
Merge

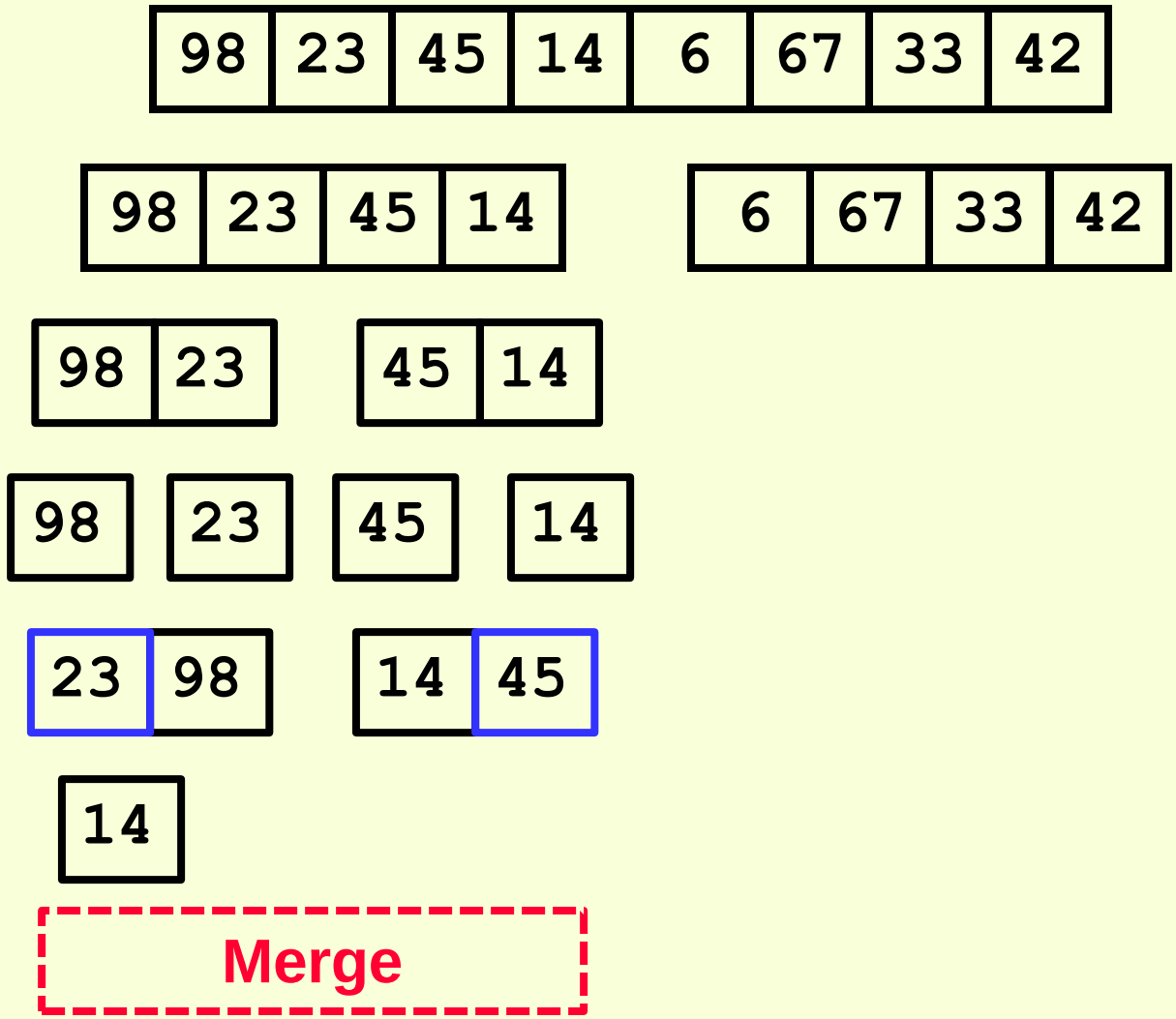

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98

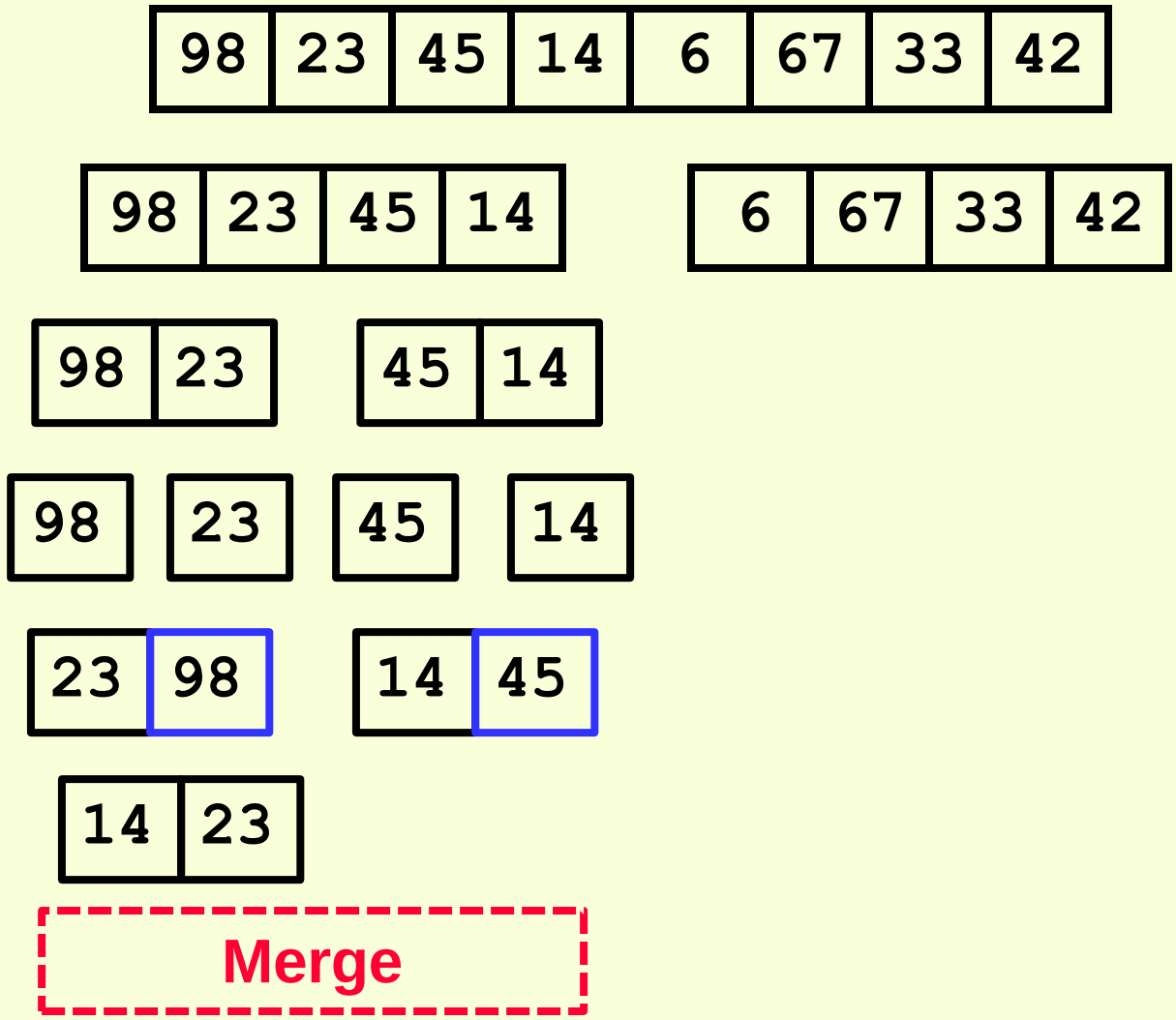

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
Merge

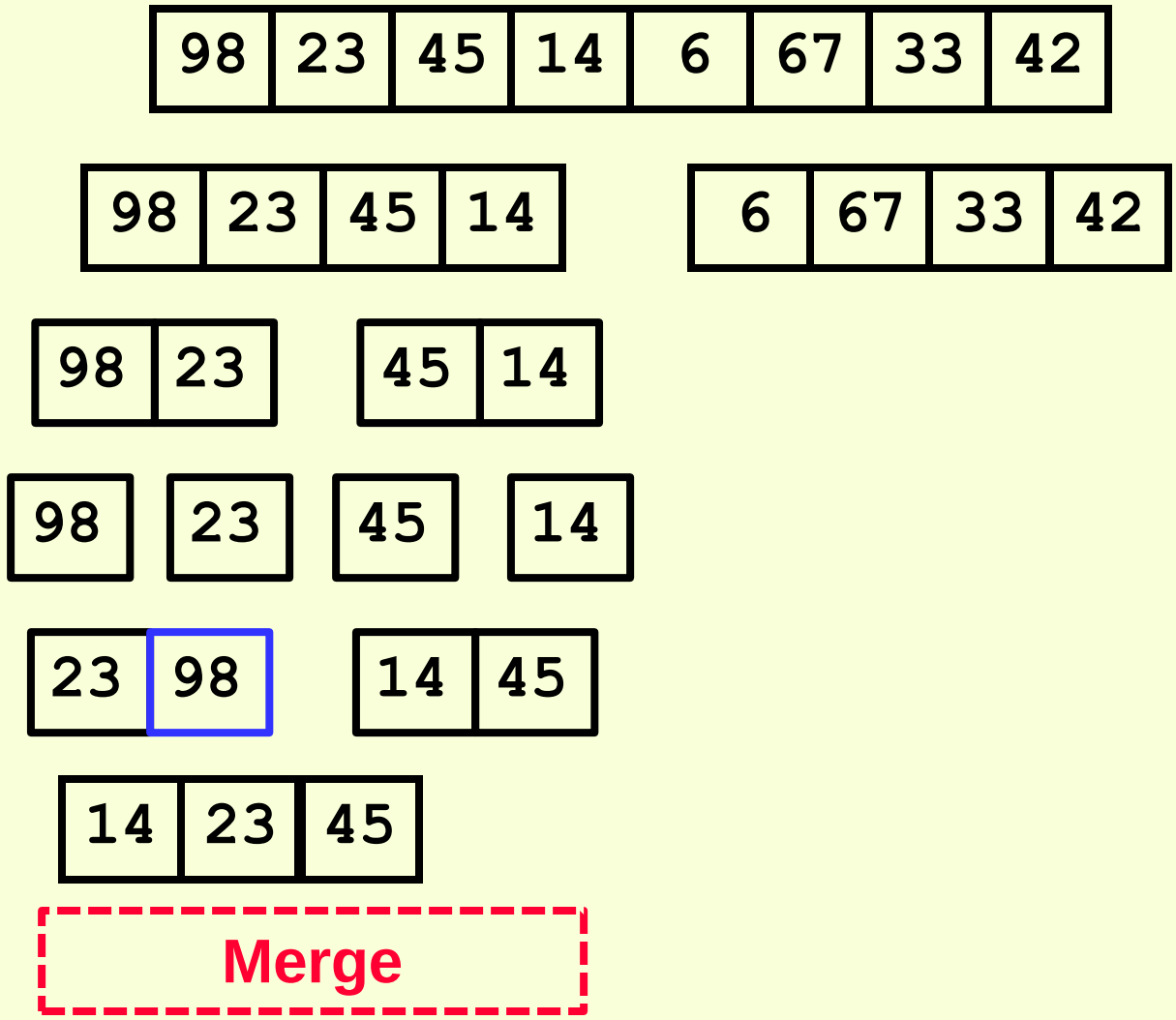

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14
Merge

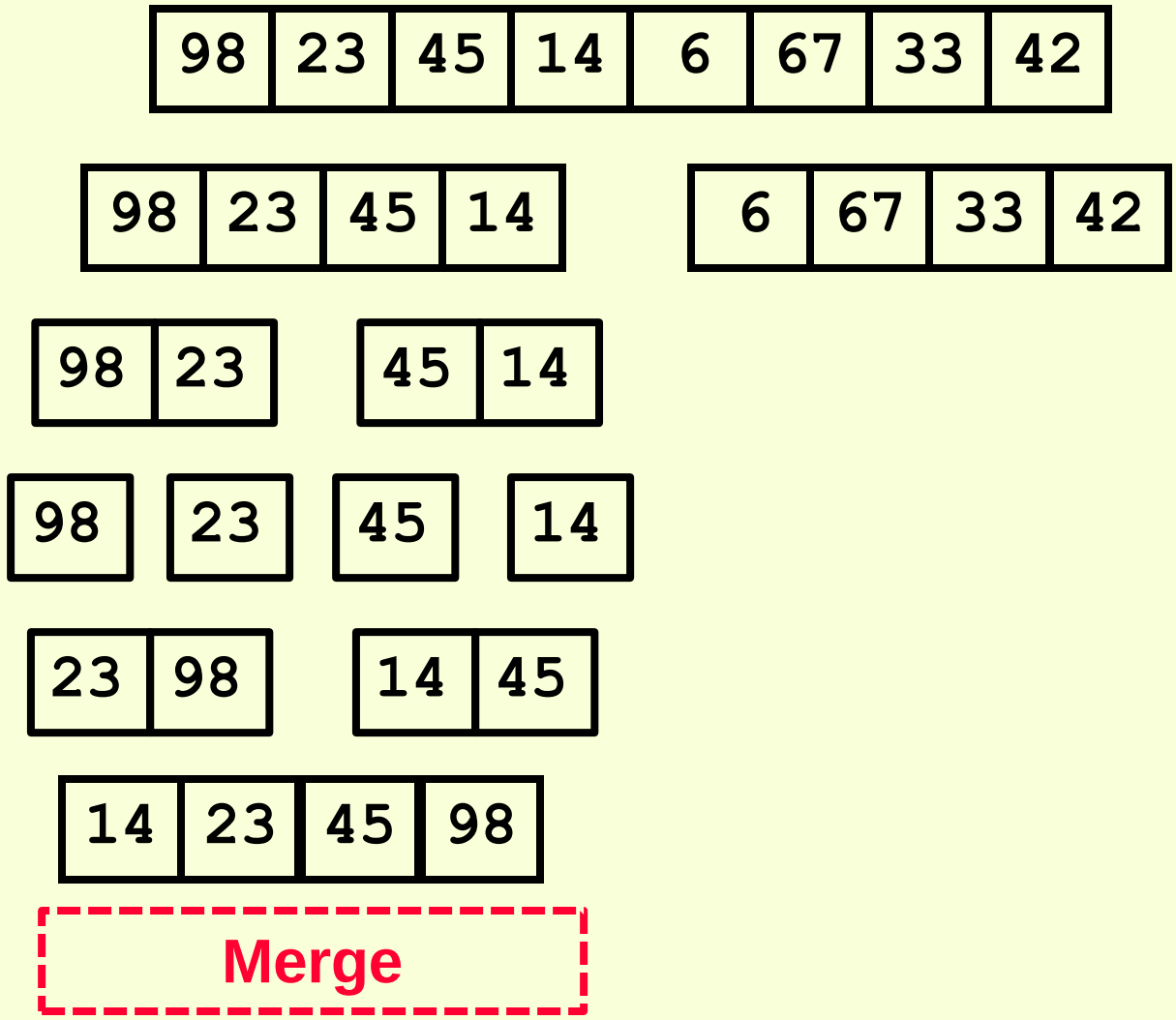

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
Merge

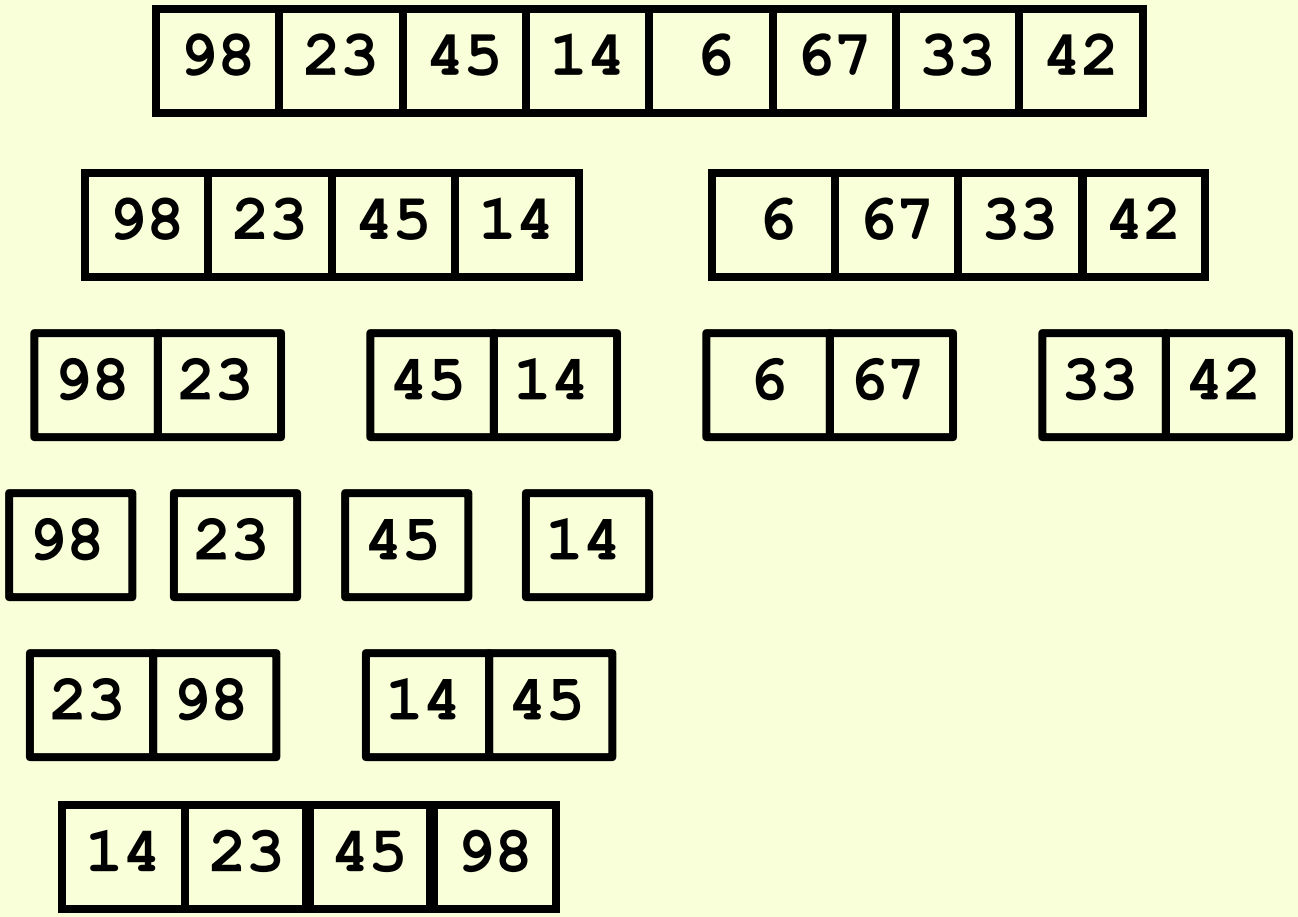

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
Merge

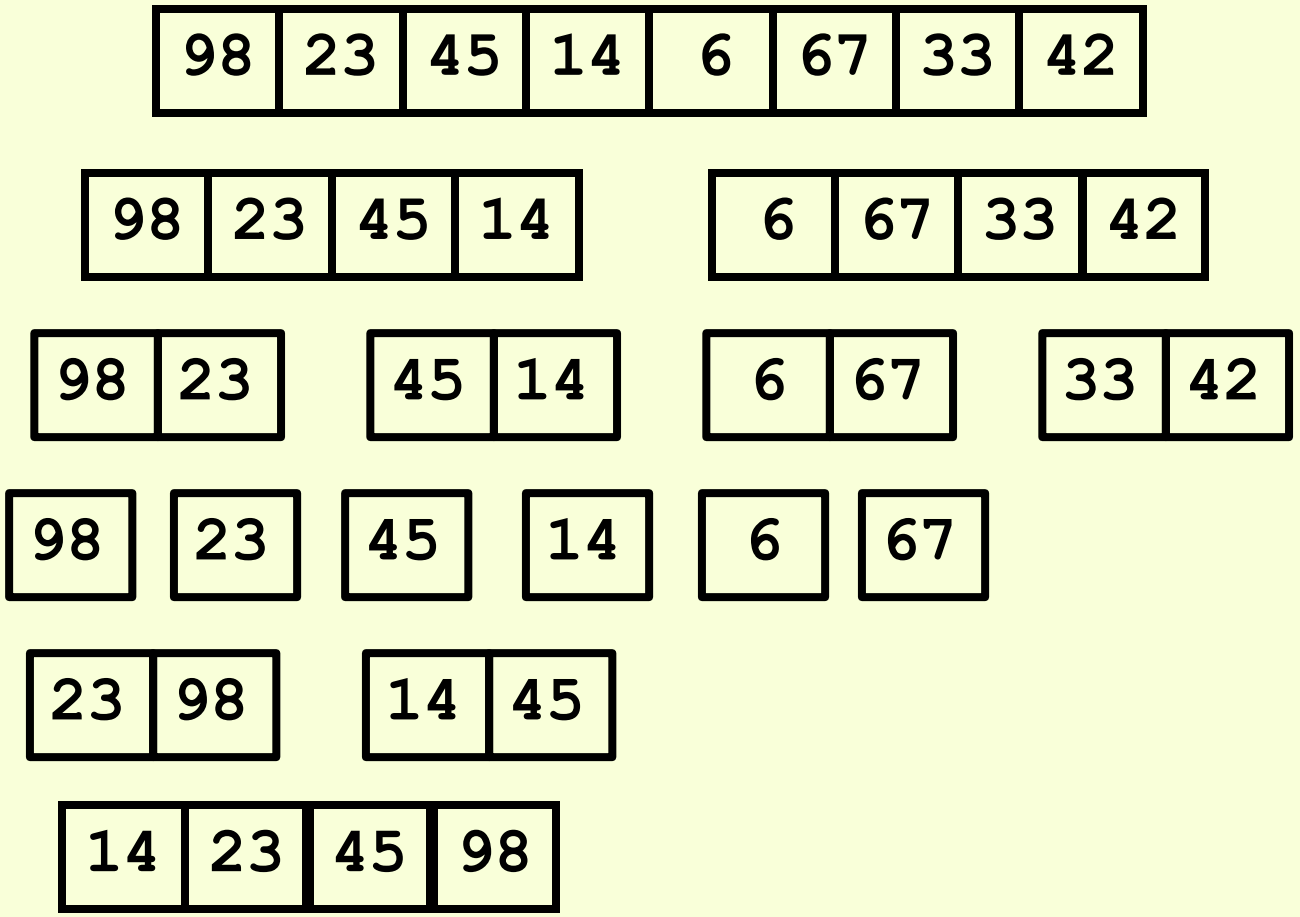

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14
Merge

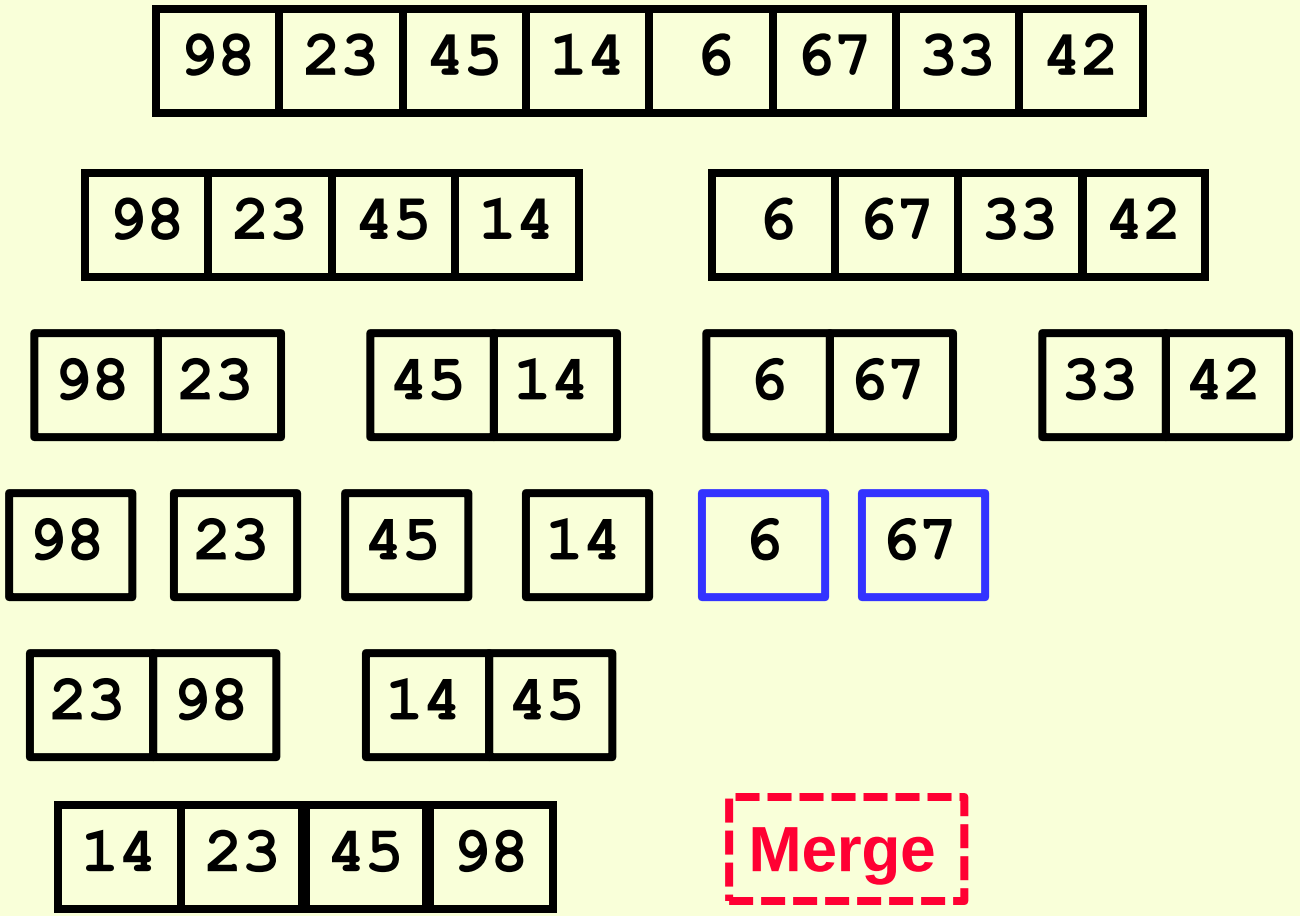

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23
Merge

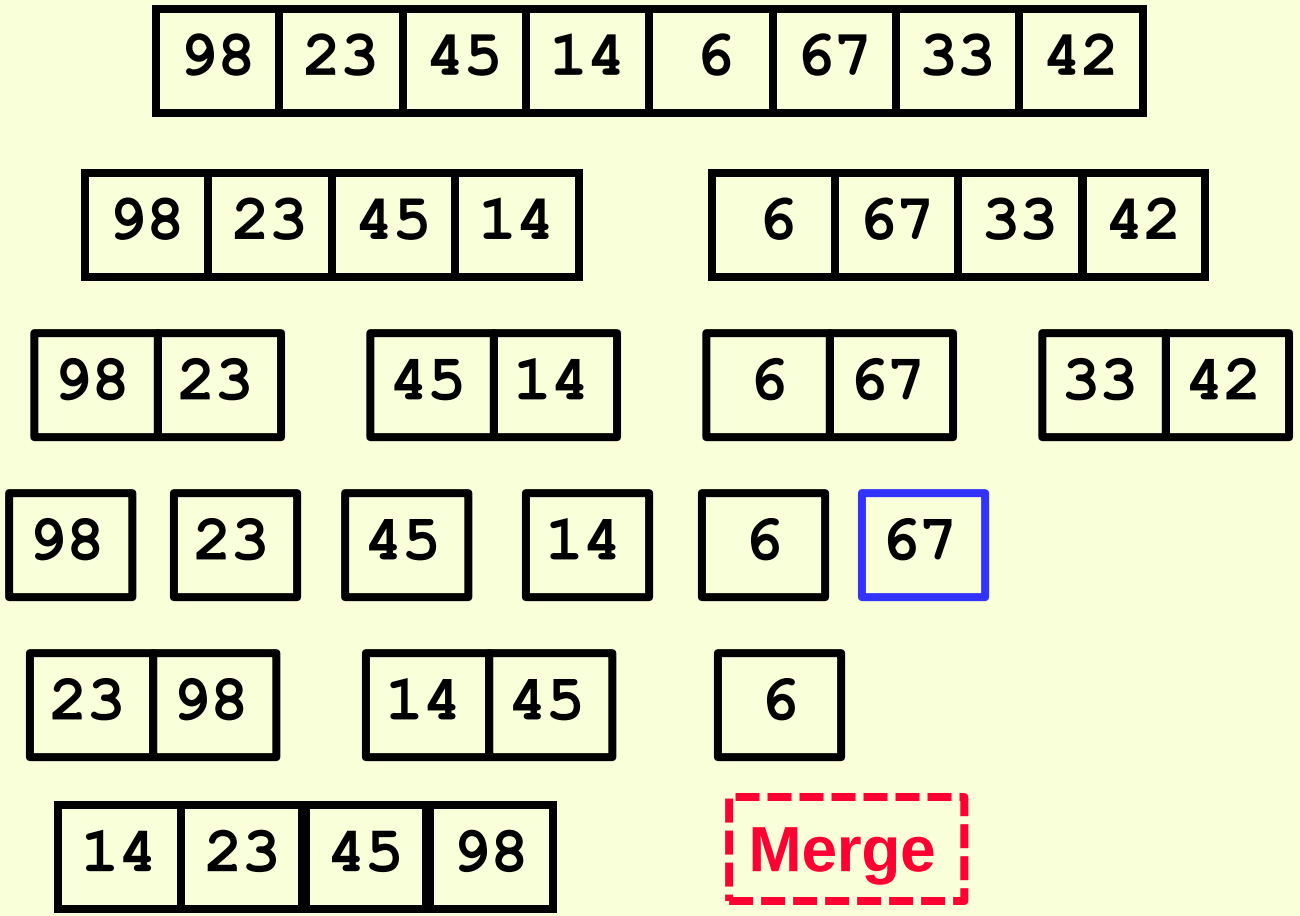

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23 45
Merge

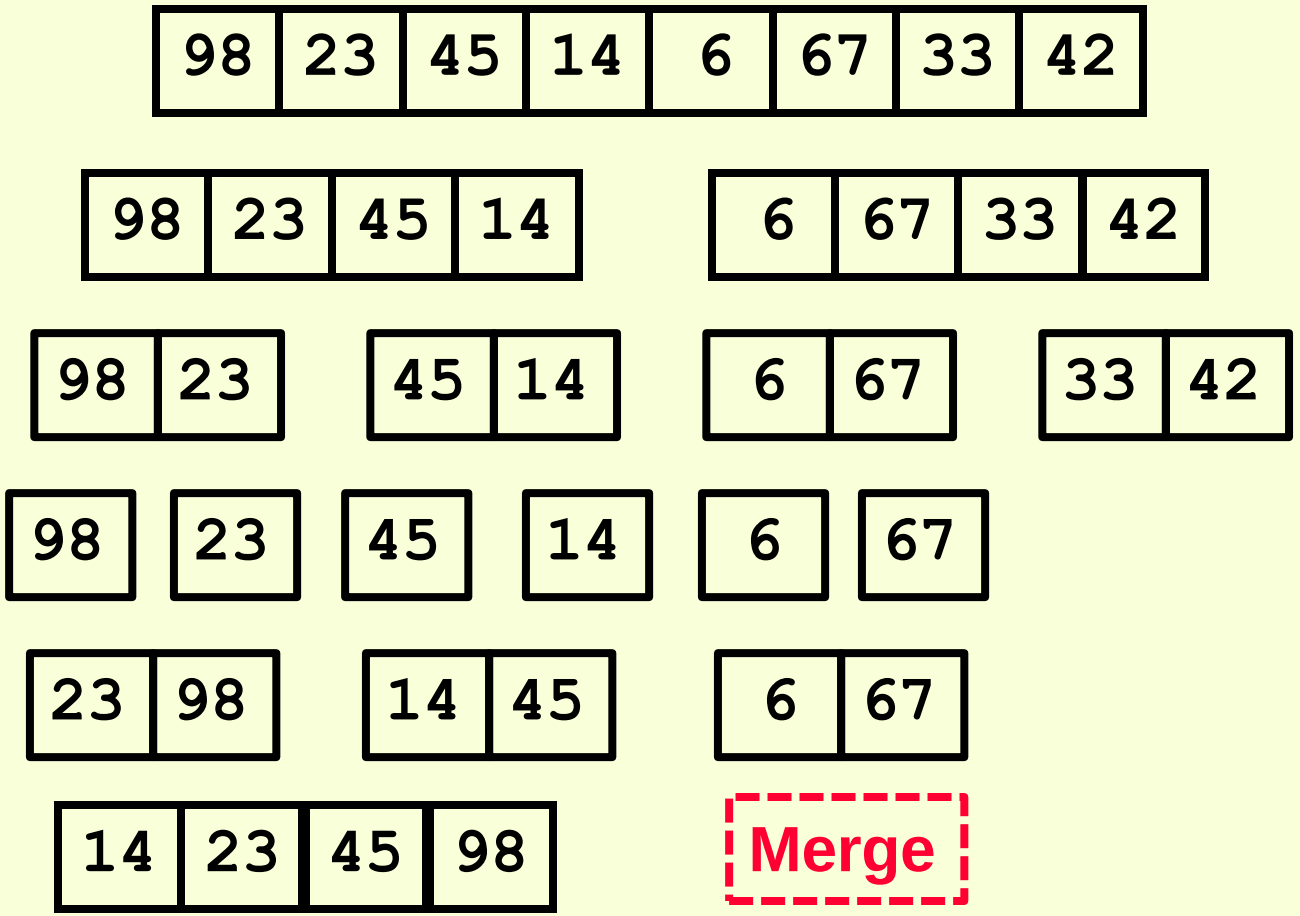

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
98
23
45
14
23 98
14 45
14 23 45 98
Merge

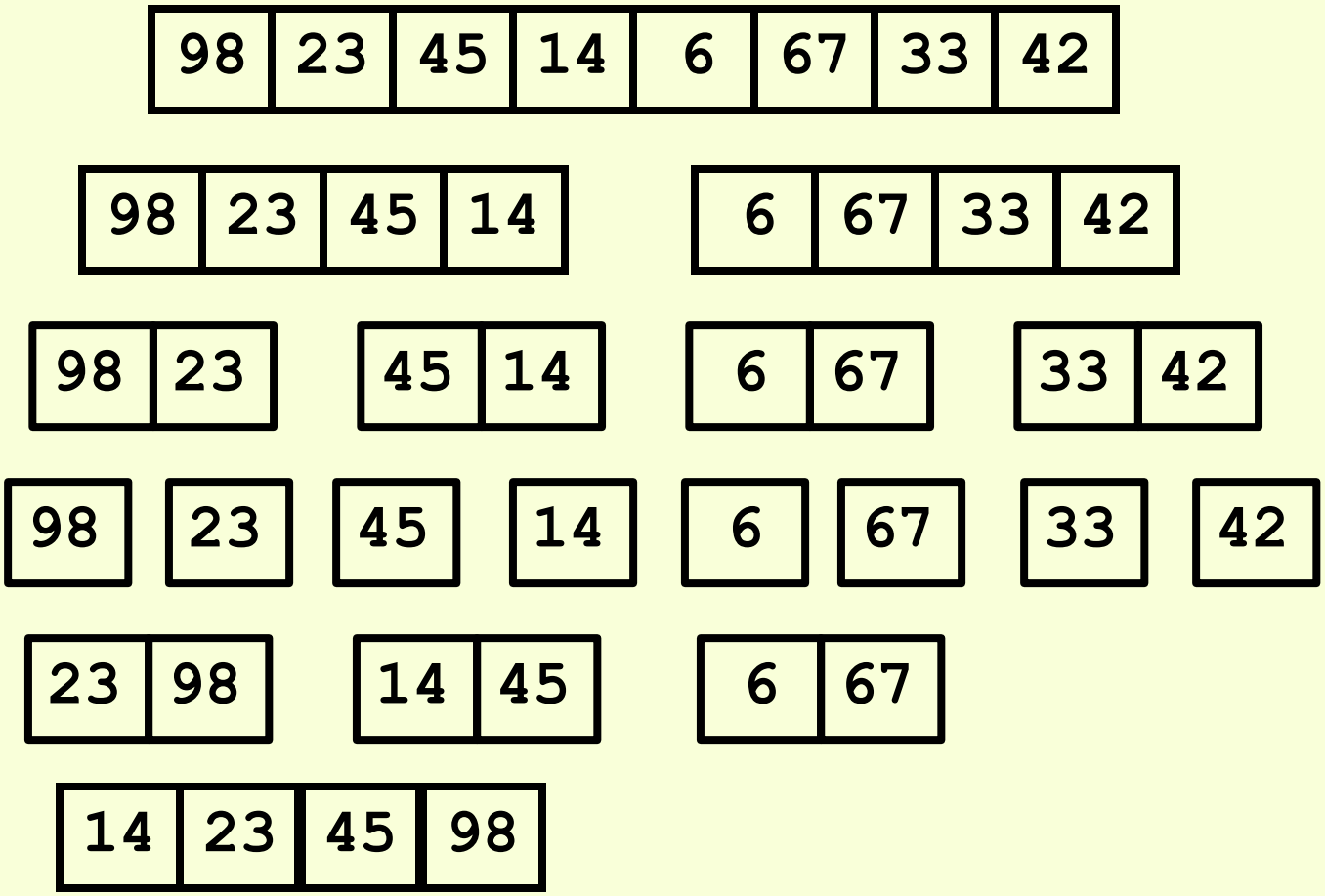

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
23 98
14 45
14 23 45 98

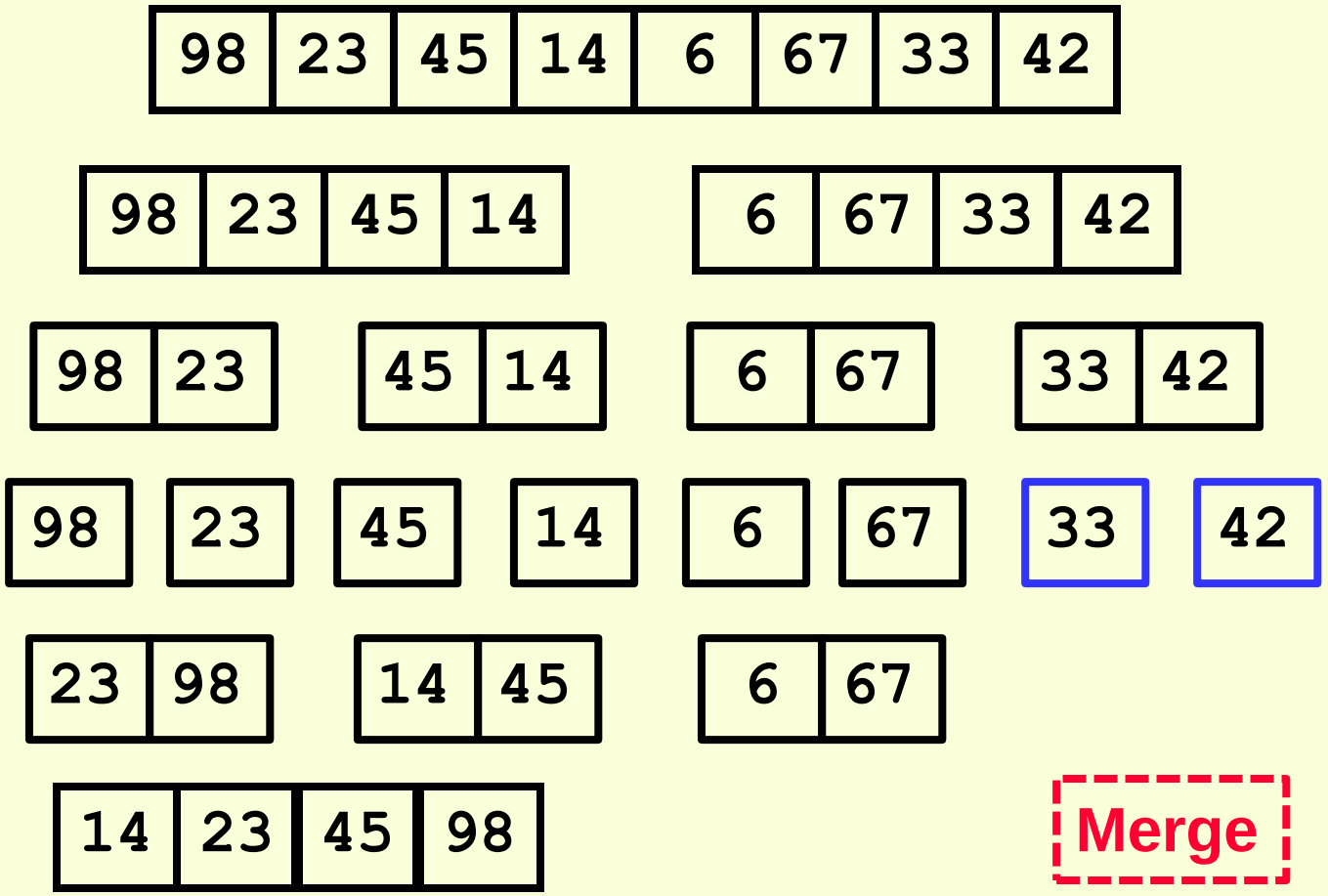

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
14 23 45 98

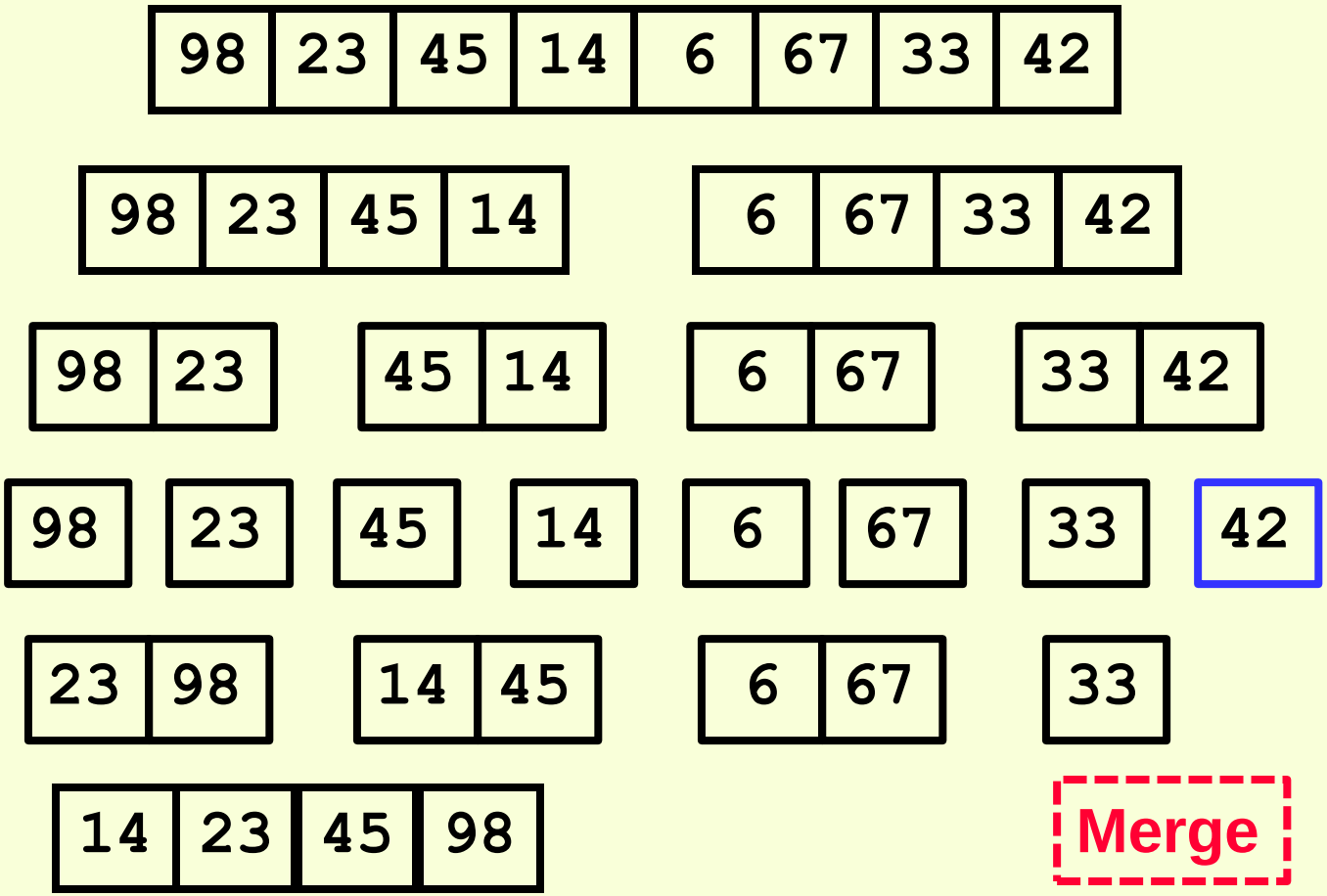

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
14 23 45 98
Merge

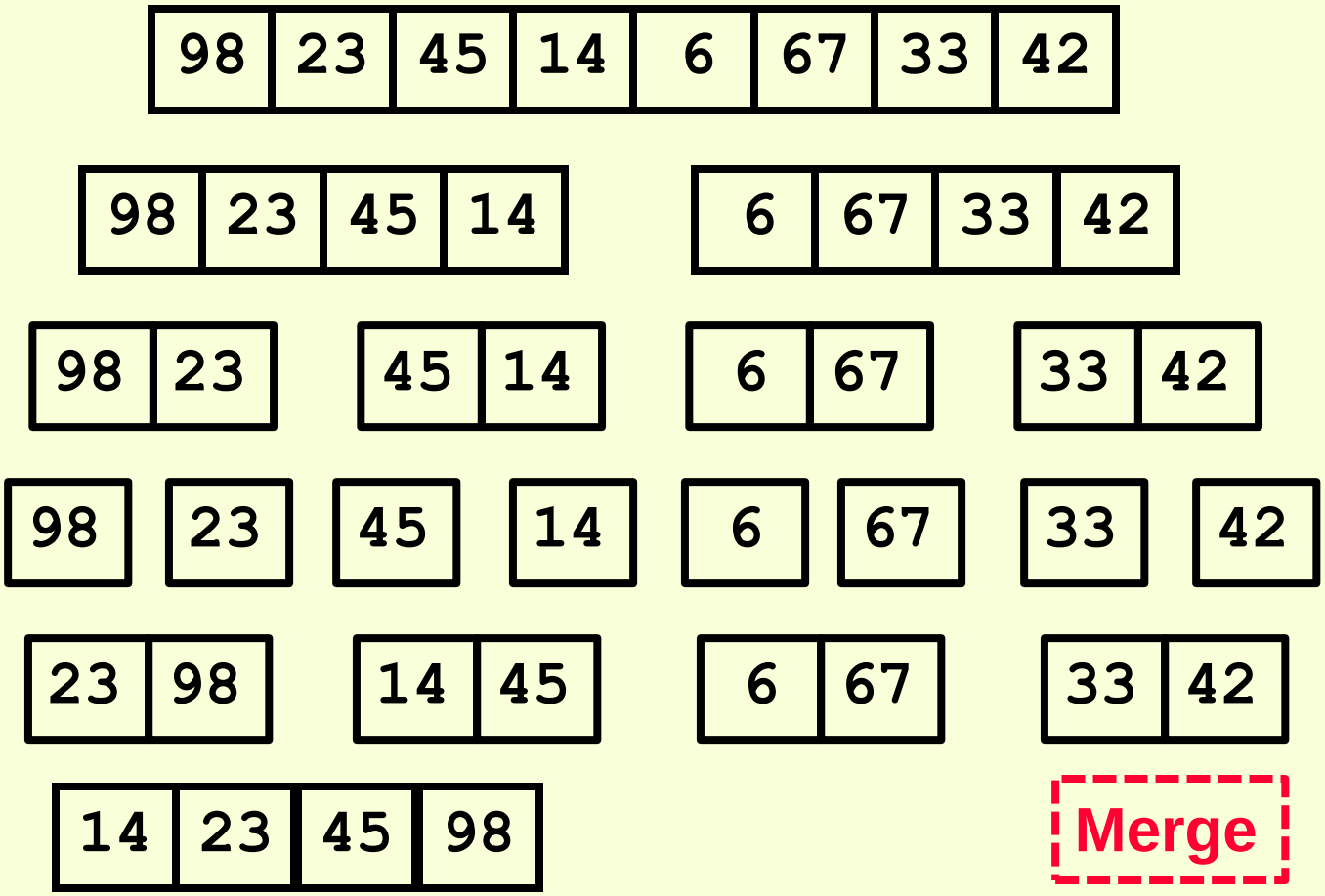

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
6
14 23 45 98
Merge

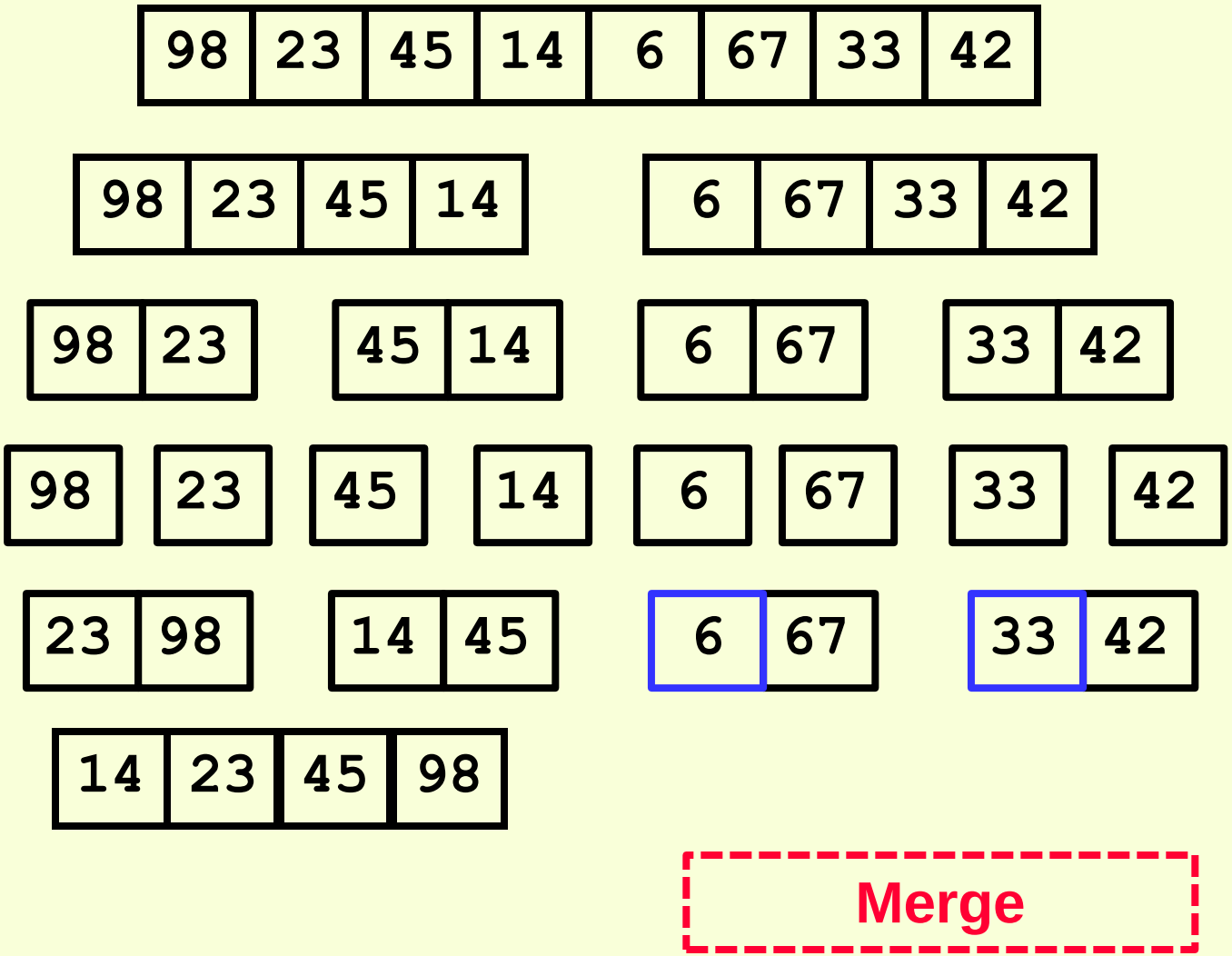

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
23 98
14 45
6 67
14 23 45 98
Merge

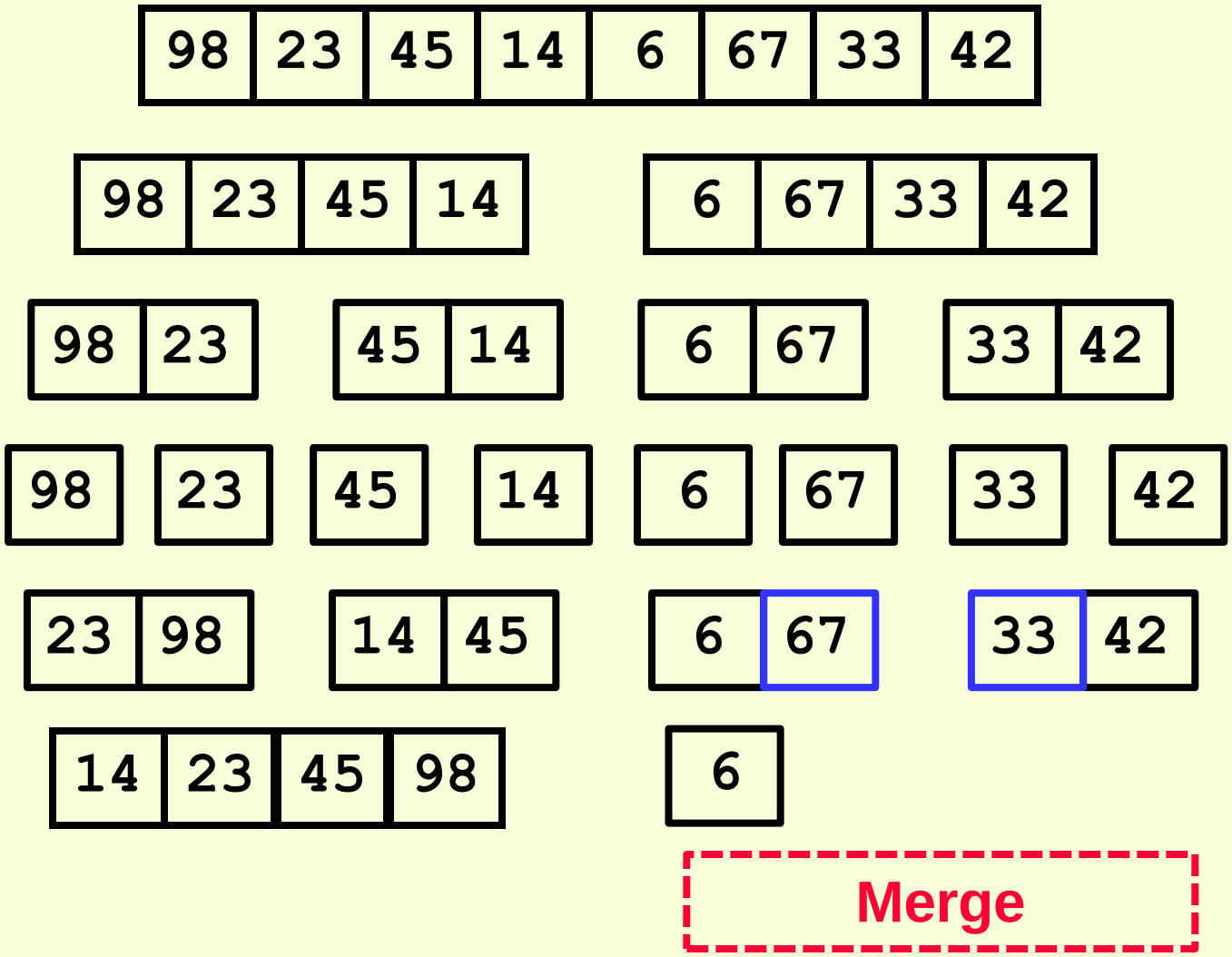

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
14 23 45 98

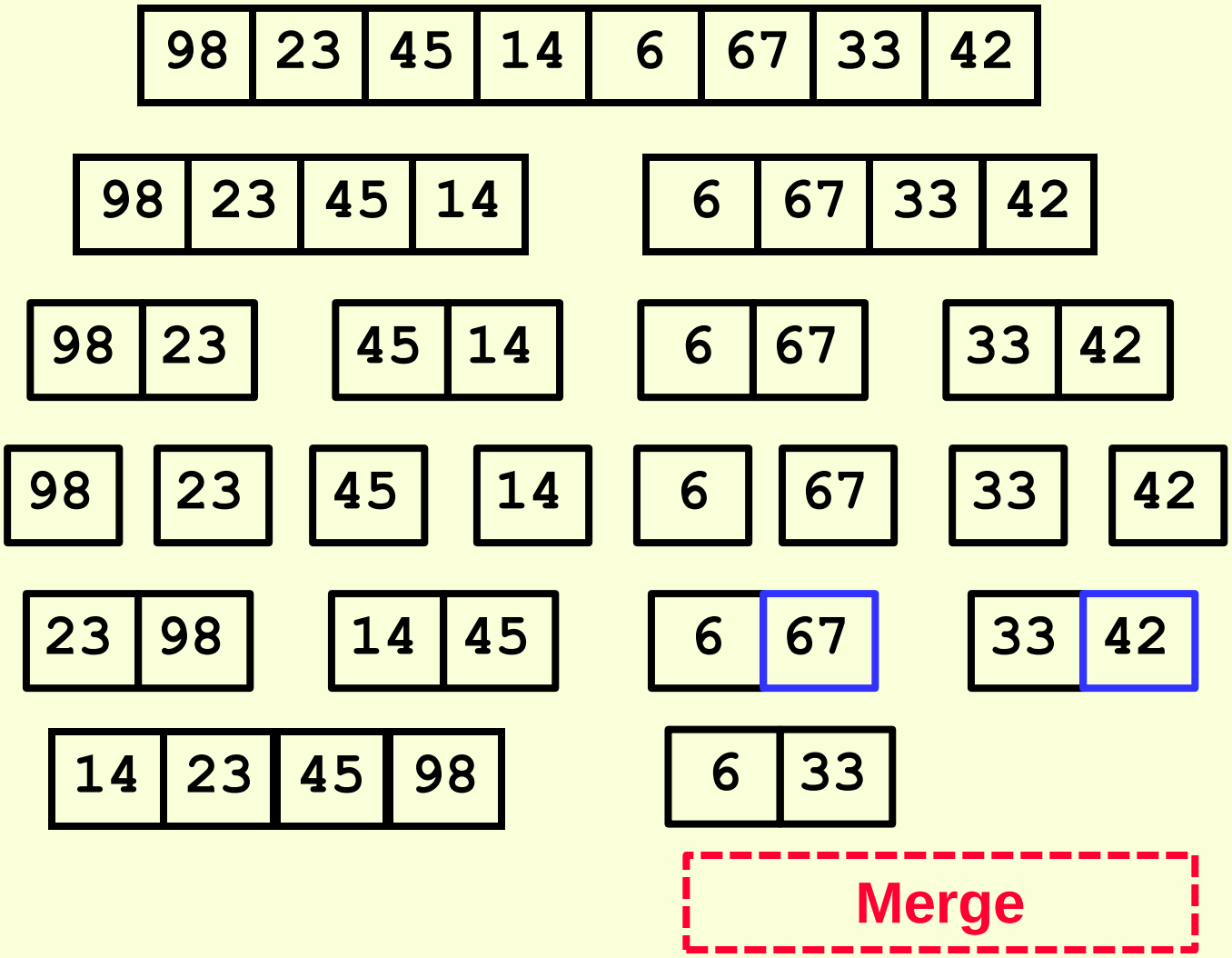

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
14 23 45 98
Merge

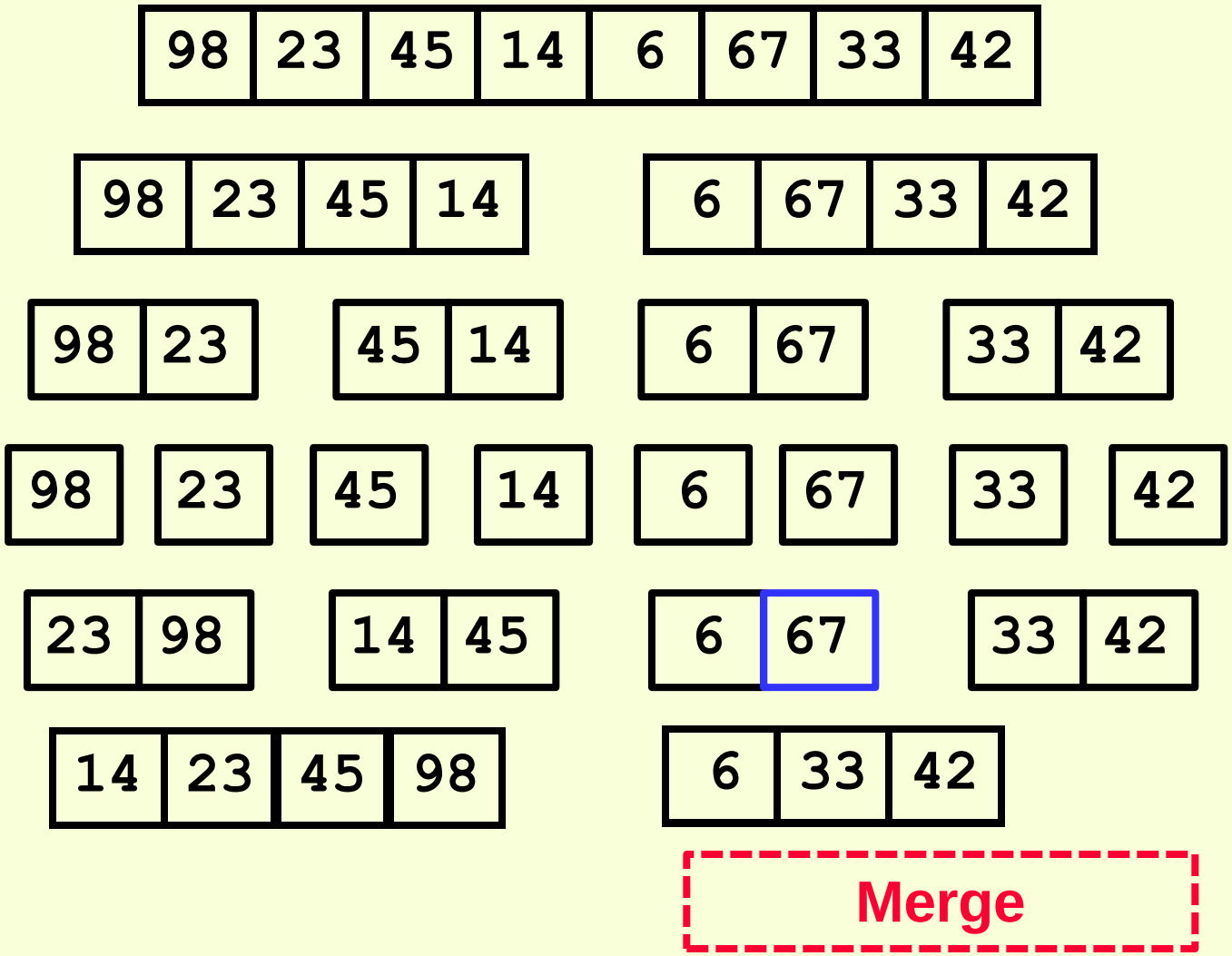

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33
14 23 45 98
Merge

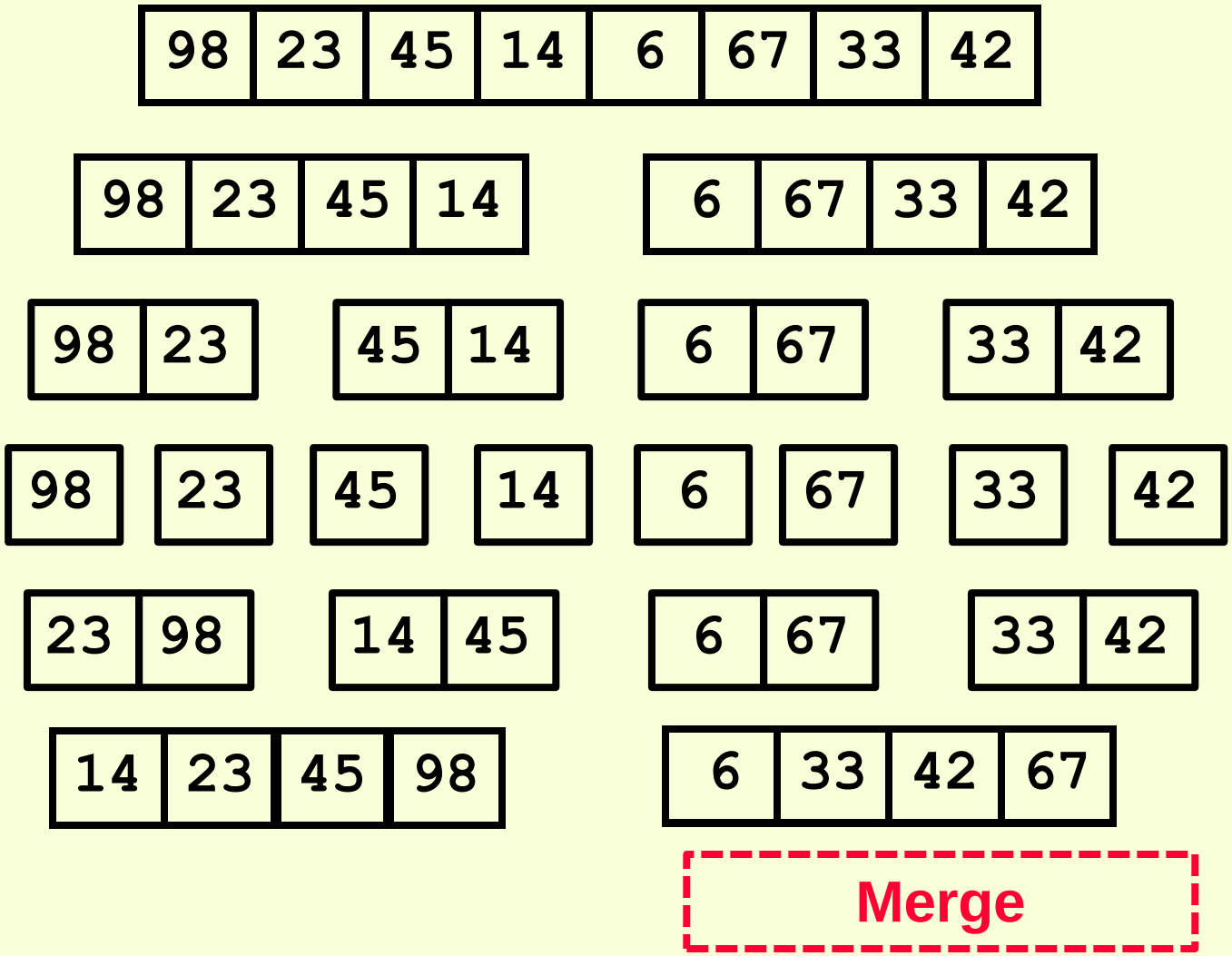

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
Merge

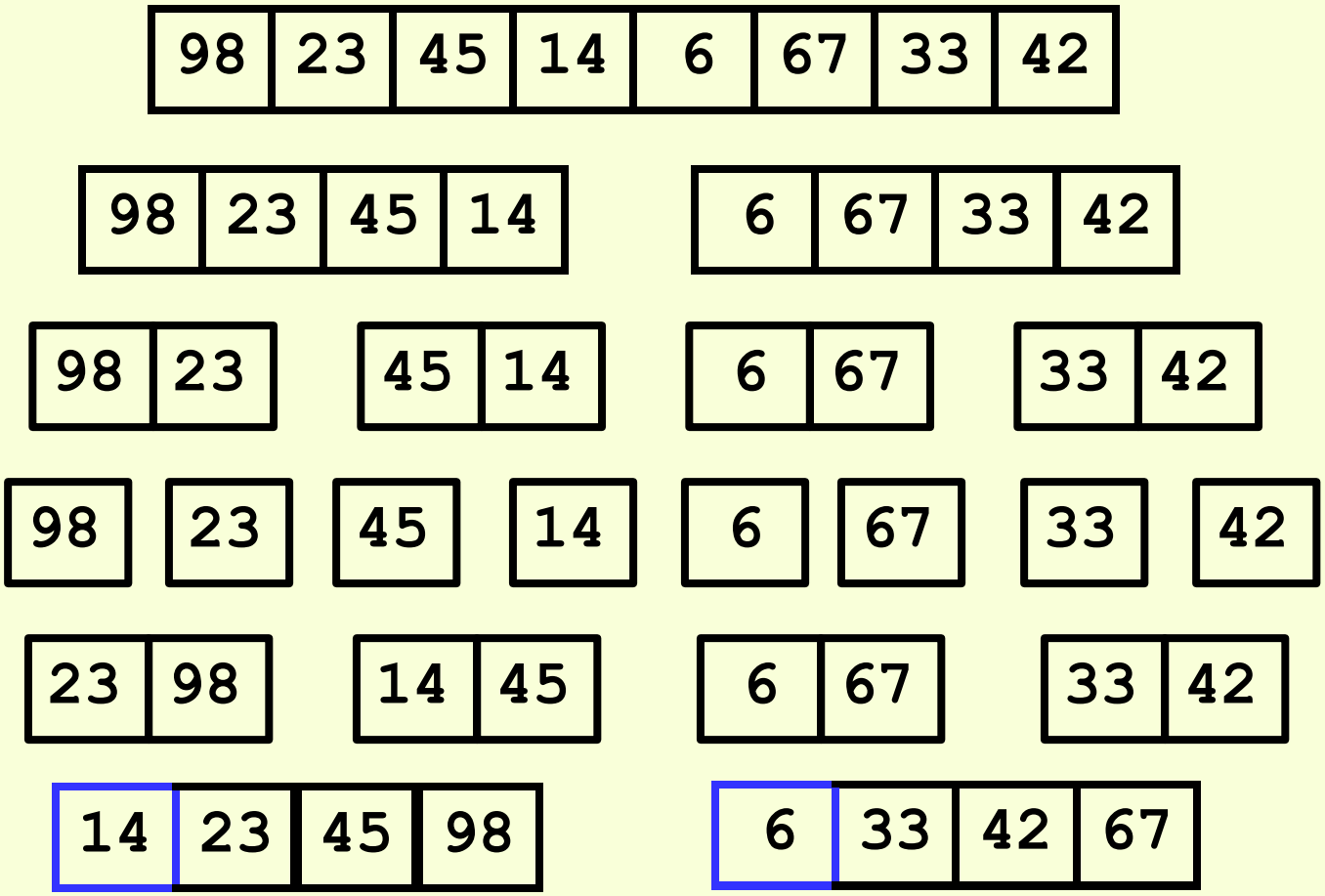

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
Merge

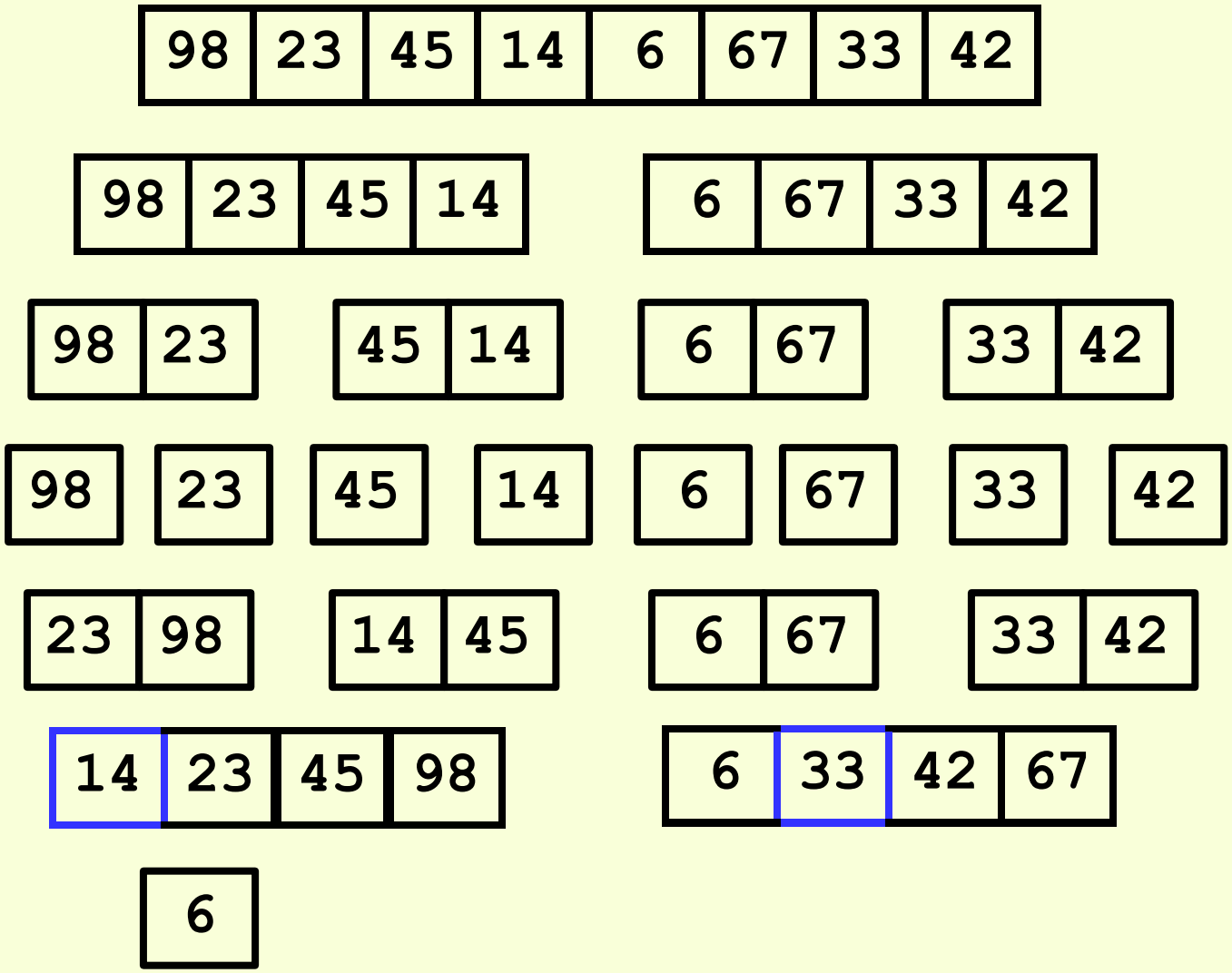

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6
Merge

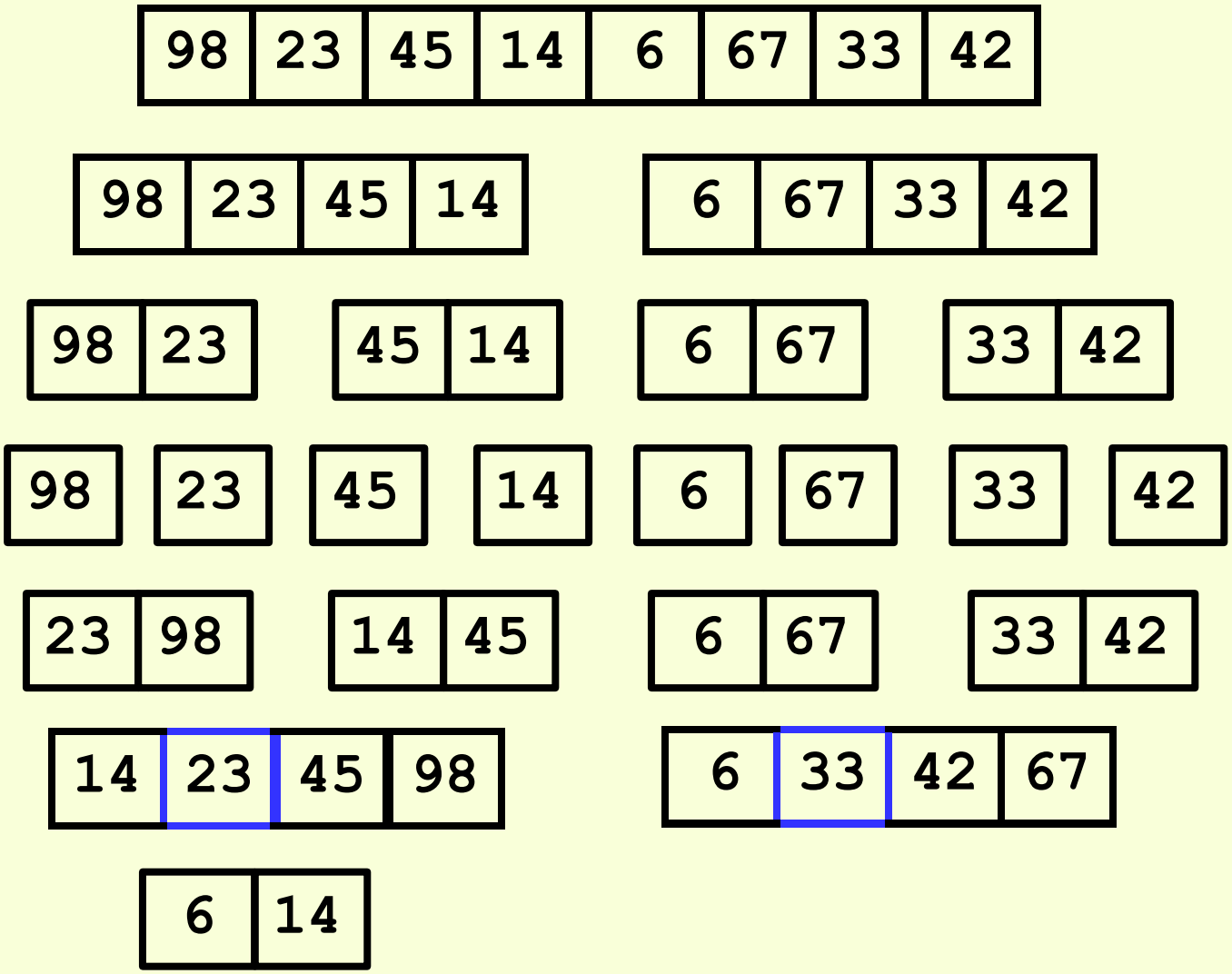

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33
Merge

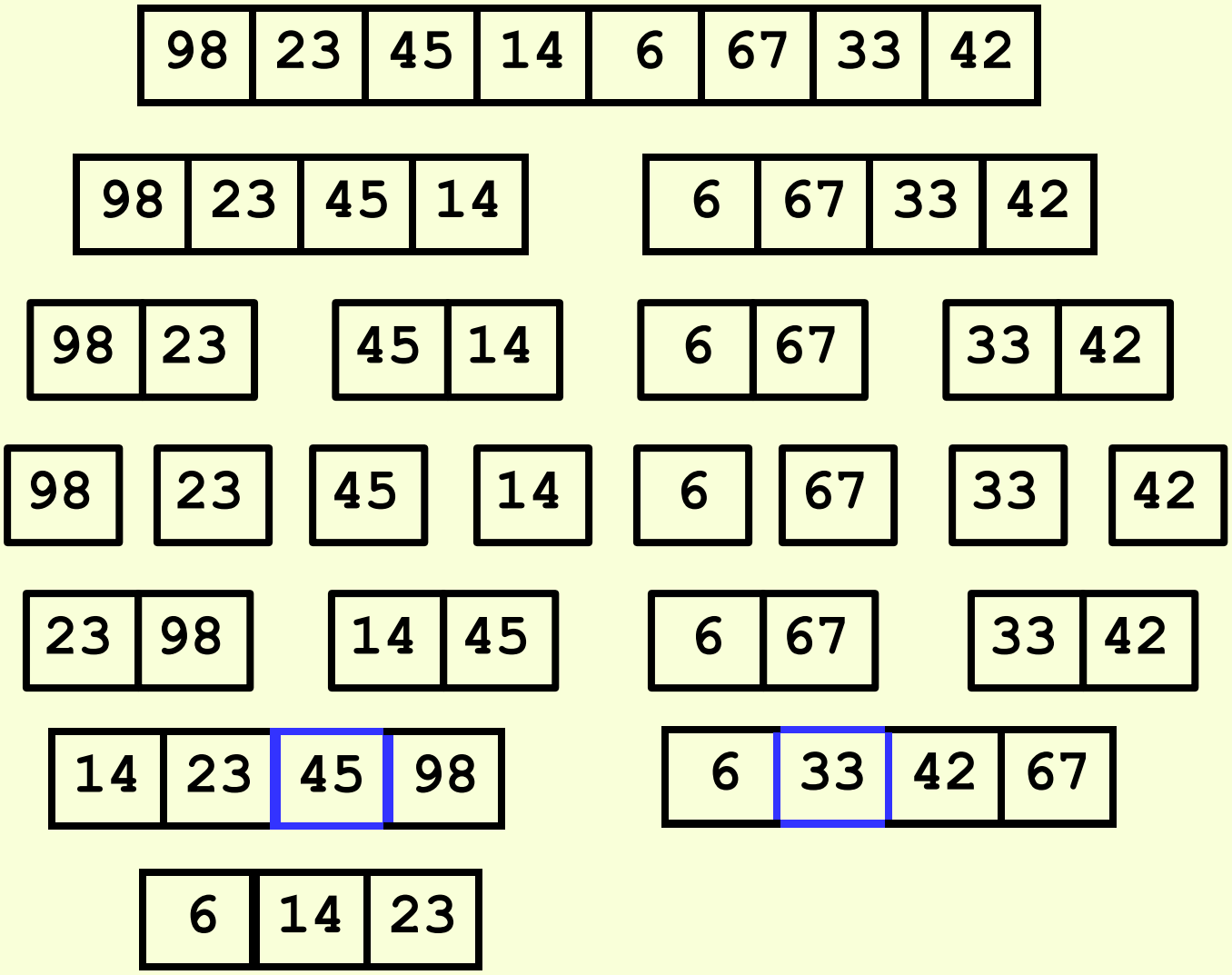

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42
Merge

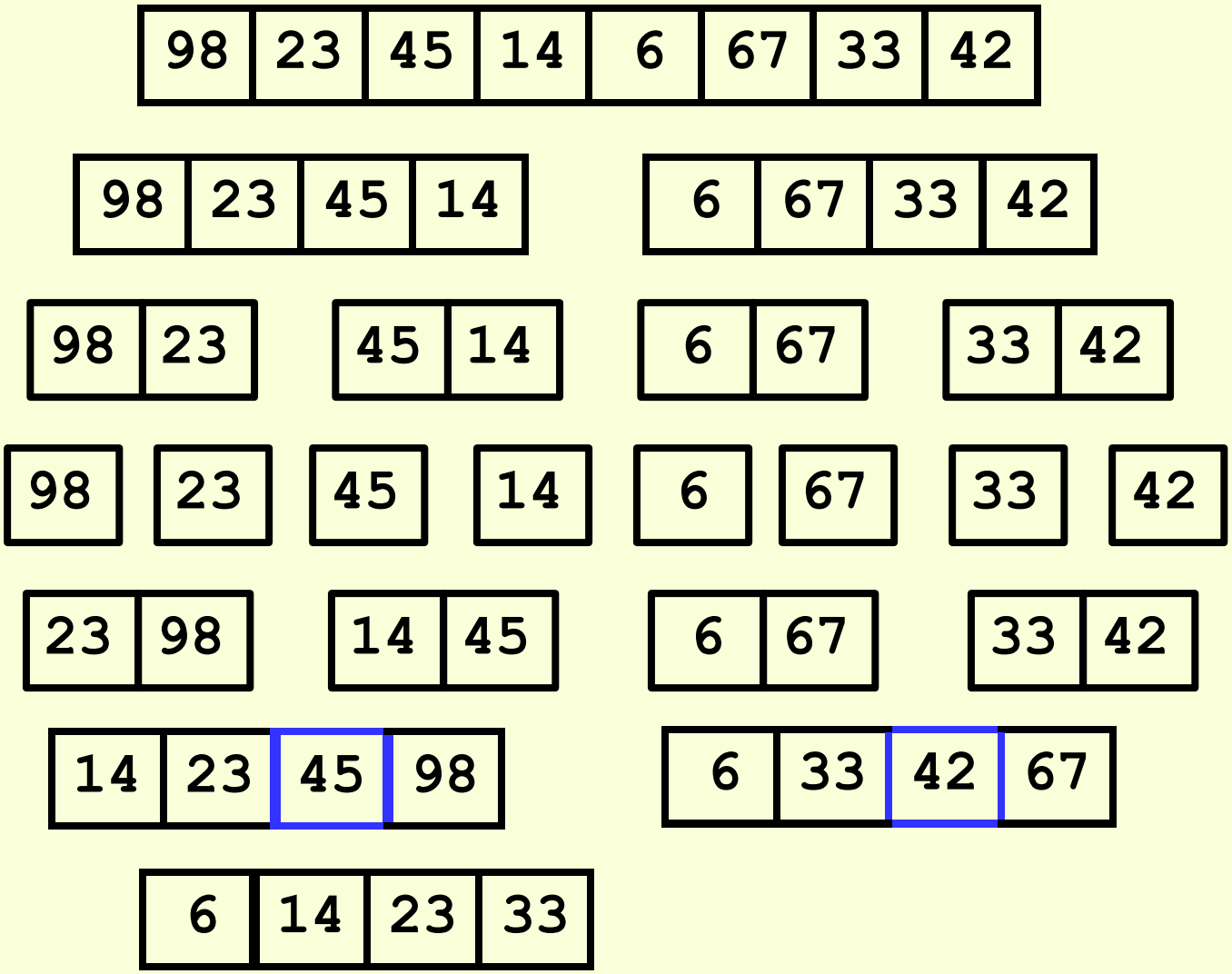

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
Merge

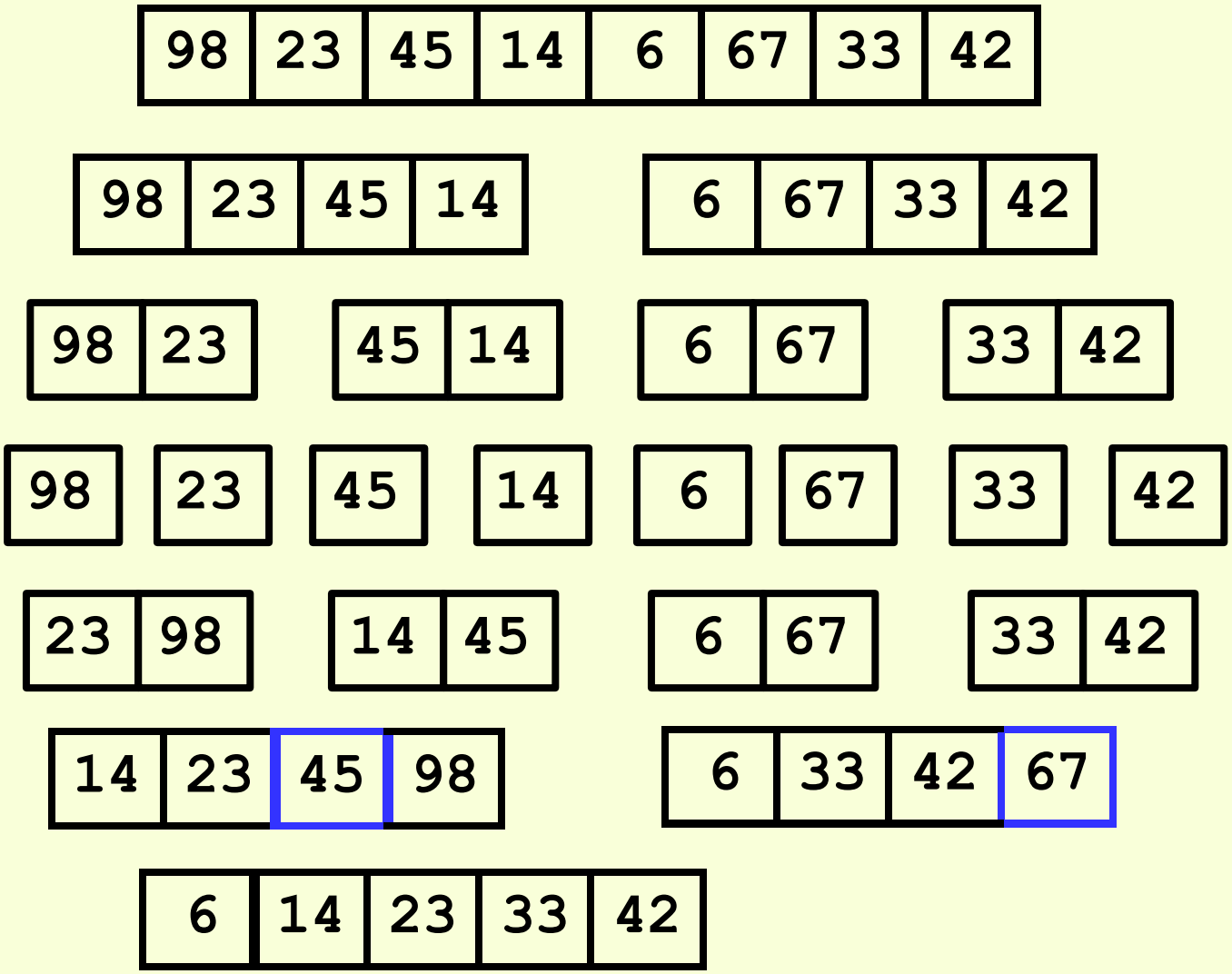

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
Merge

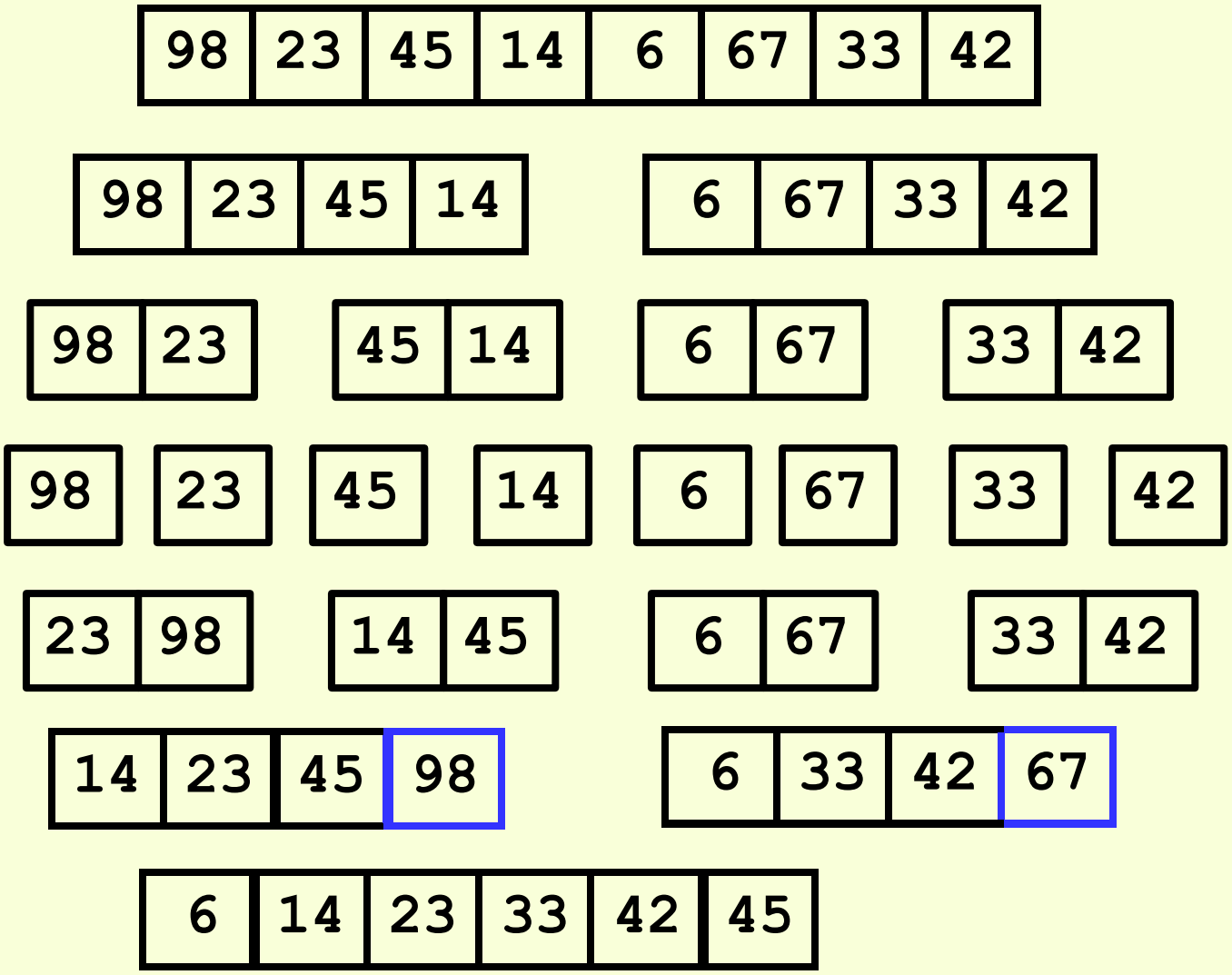

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6
Merge

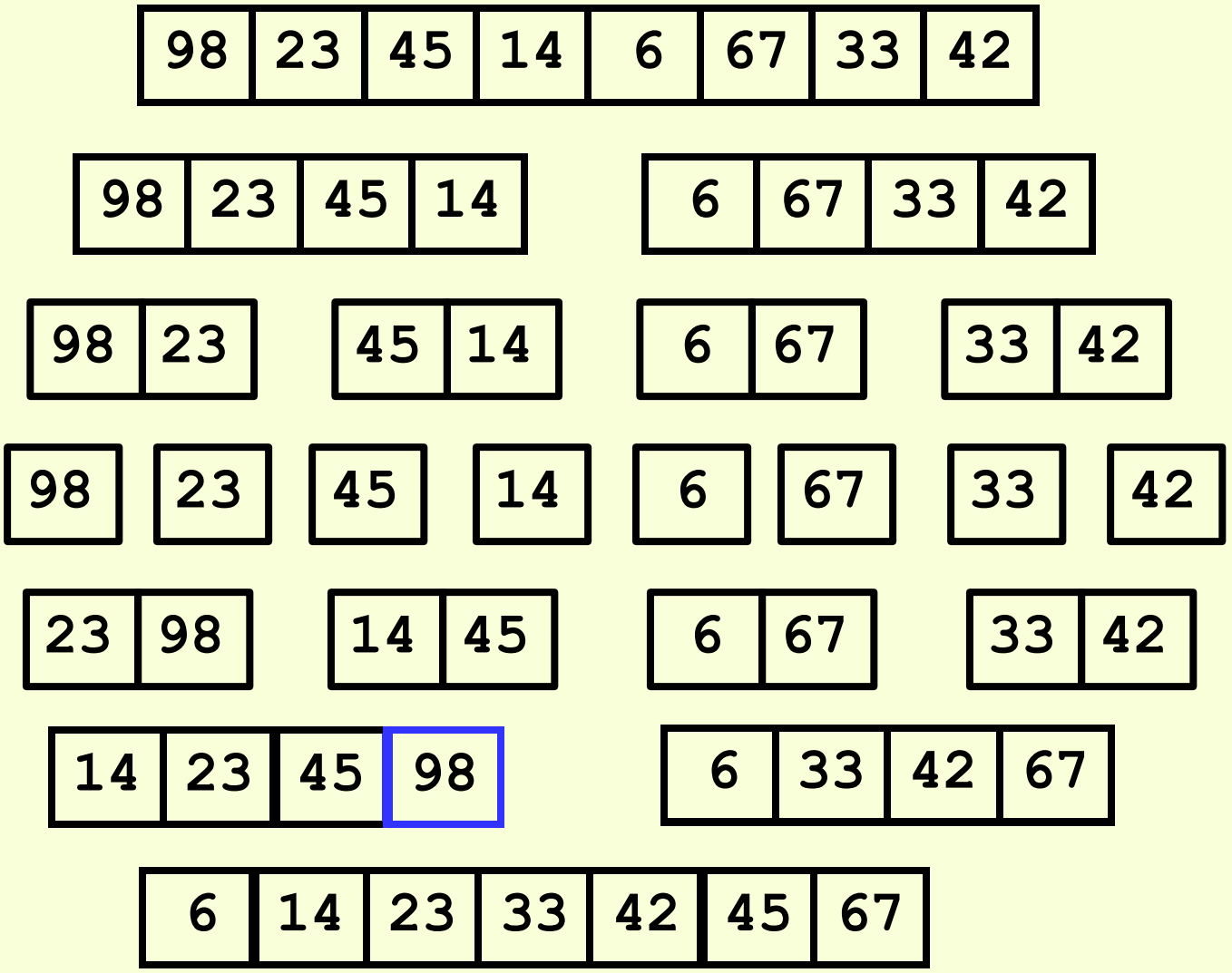

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14
Merge

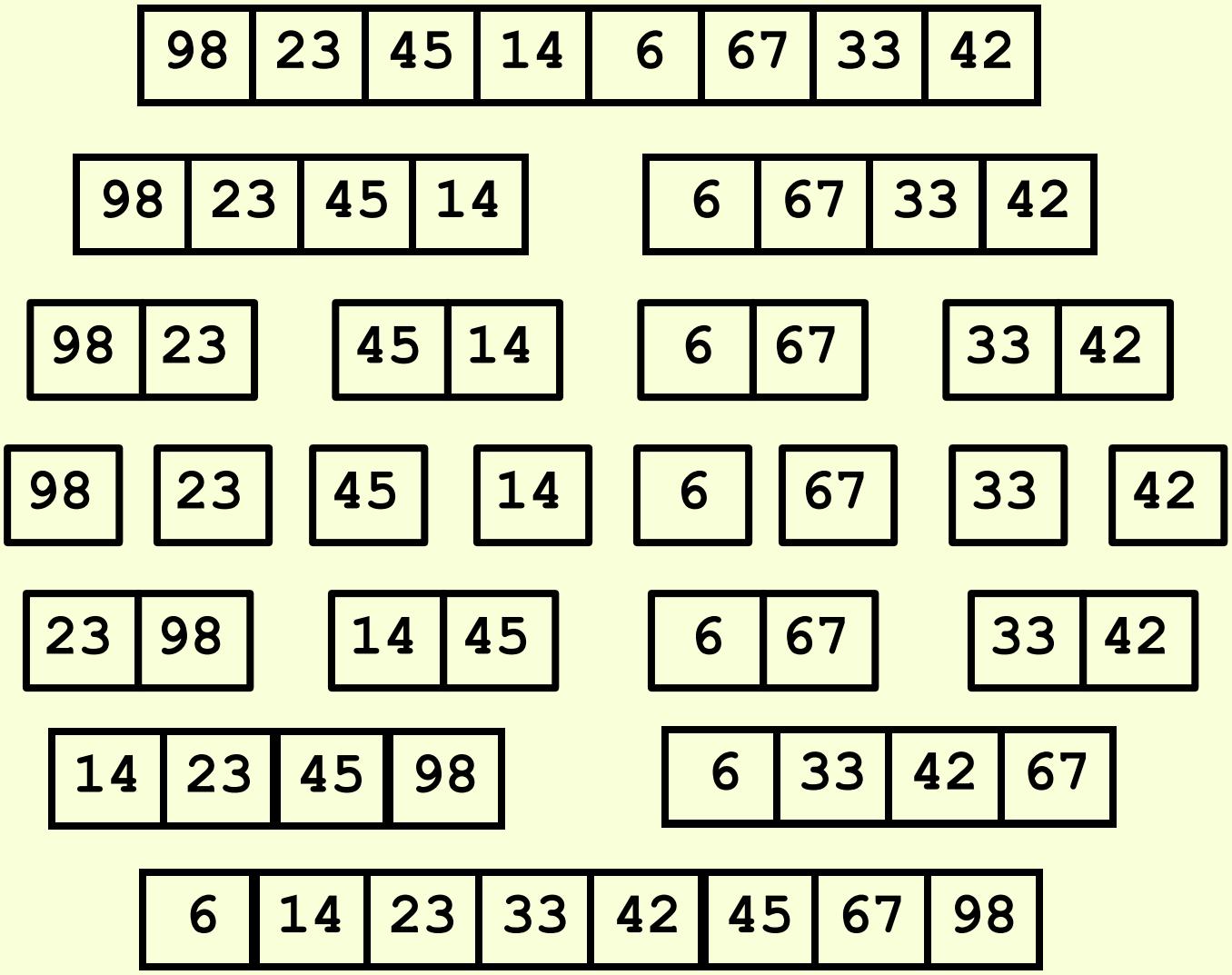

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23
Merge

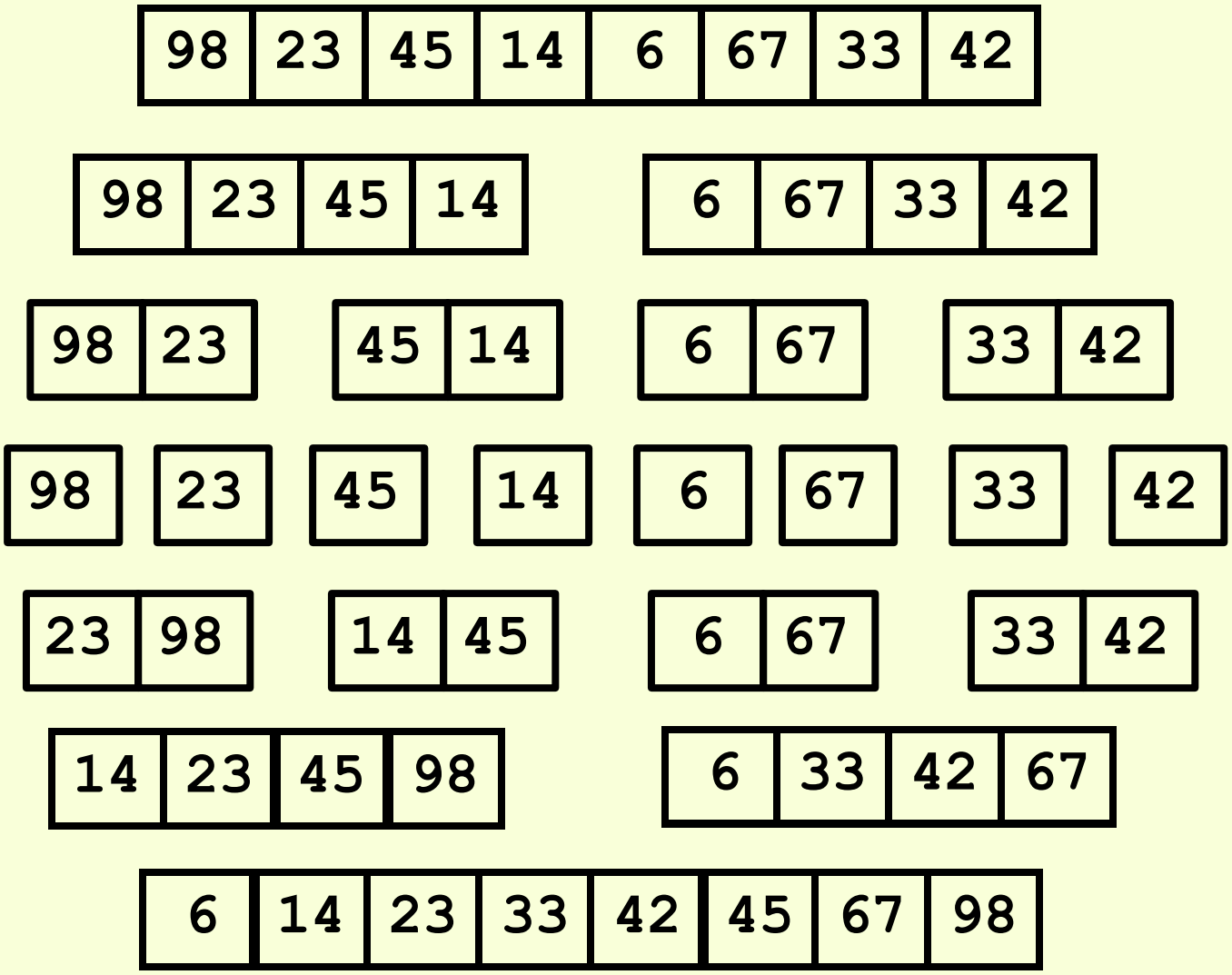

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33
Merge

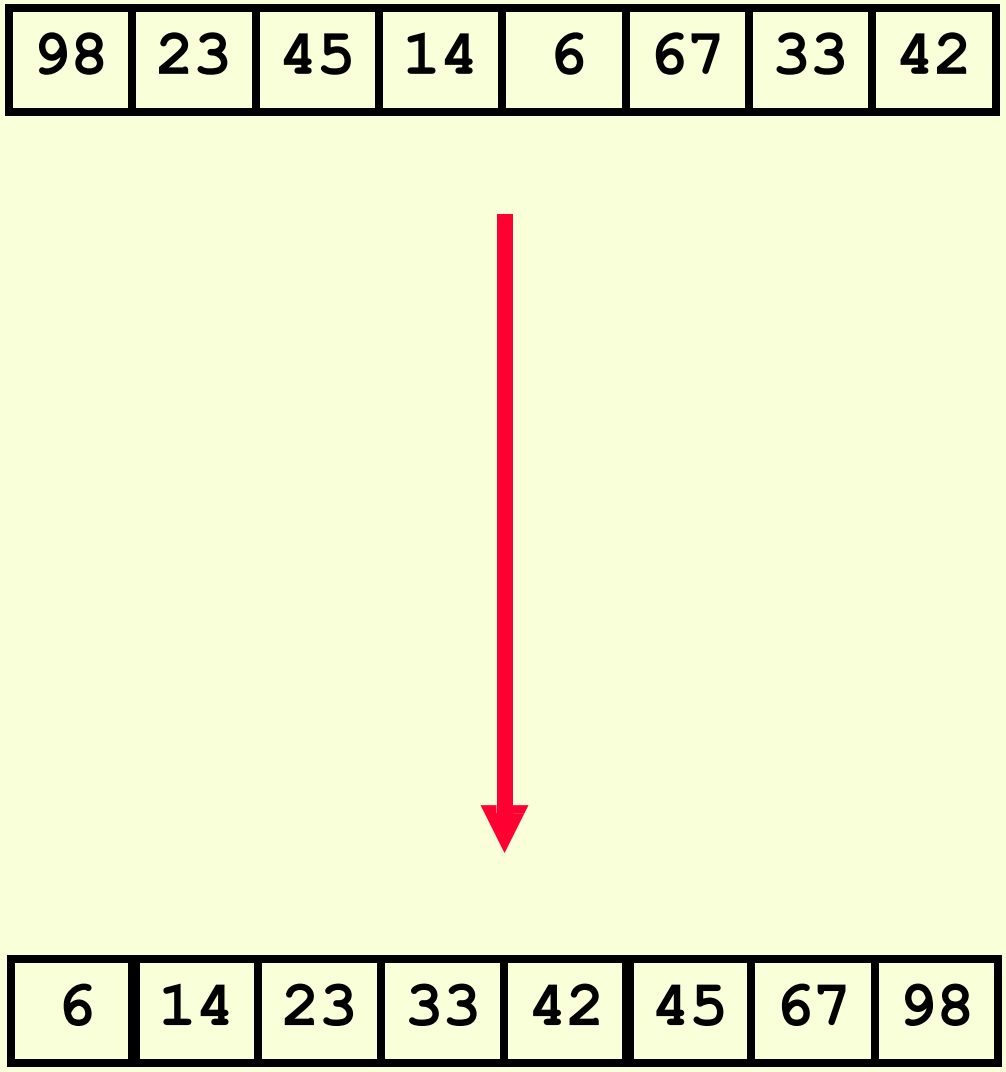

98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45 67
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45 67 98
Merge


98 23 45 14 6 67 33 42
98 23 45 14
6 67 33 42
98 23
45 14
6 67
33 42
98
23
45
14
6
67
33
42
23 98
14 45
6 67
33 42
14 23 45 98
6 33 42 67
6 14 23 33 42 45 67 98


98 23 45 14 6 67 33 42
6 14 23 33 42 45 67 98

Summary

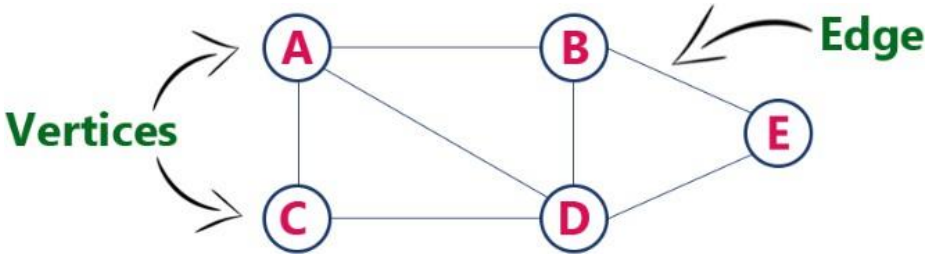
- Divide the unsorted collection into two
- Until the sub-arrays only contain one element
- Then merge the sub-problem solutions together

Questions?

Graphs

What is a graph?

- Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges that relate the nodes to each other. الرسم البياني هو بنية بيانات غير خطية. ويحتوي على مجموعة من النقاط تعرف بالعقد (أو القمم) ومجموعة من الروابط تعرف بالحواف التي تربط العقد ببعضها البعض.
- The set of edges describes relationships among the vertices. مجموعة الحواف تصف العلاقات بين القمم.



Formal definition of Graphs التعريف الرسمي للرسوم البيانية

Generally, a graph **G** is represented as **G = (V , E)**, where **V is set of vertices** القمم and **E is set of edges** حواف .. In the above figure the following is a graph with 5 vertices and 7 edges.

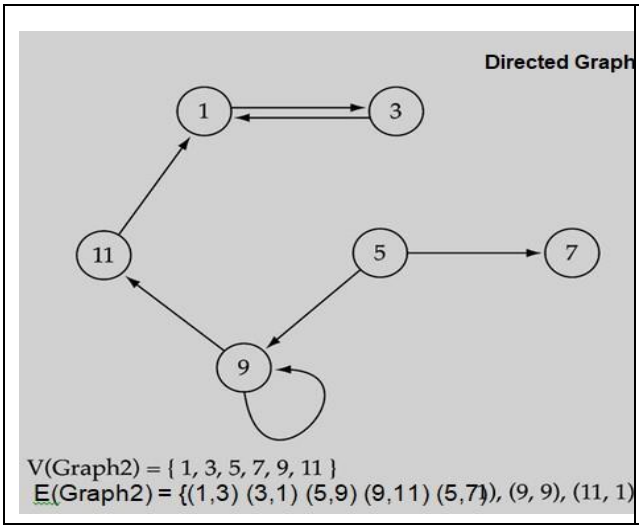
هي مجموعة الرؤوس القمم و V ، حيث(V , E) = G :على النحو التالي G بشكل عام، يتم تمثيل الرسم البياني

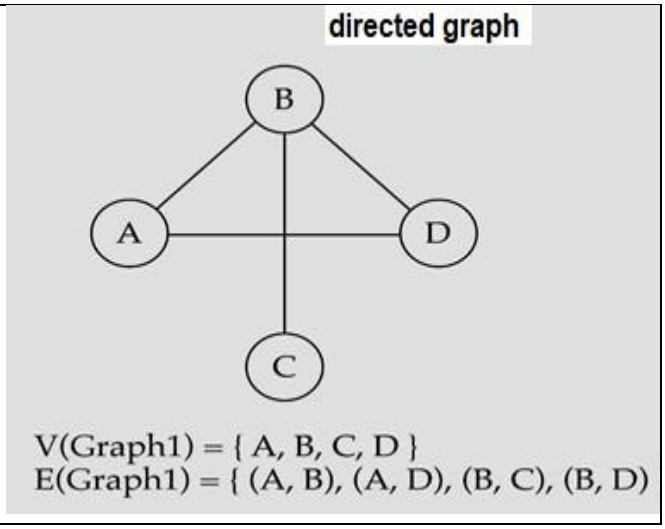
E هي مجموعة الحواف الحادة.. في الشكل أعلاه، يوجد رسم بياني يحتوي على 5 رؤوس و7 حواف.

V = {A,B,C,D,E} and E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.

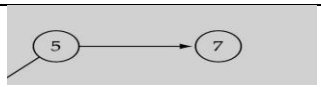
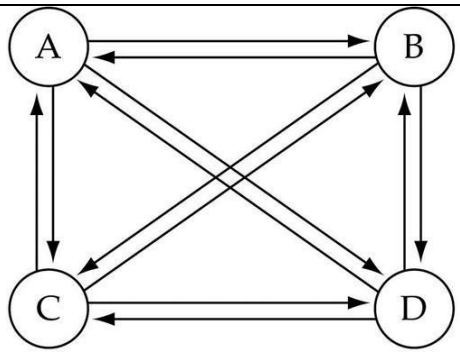
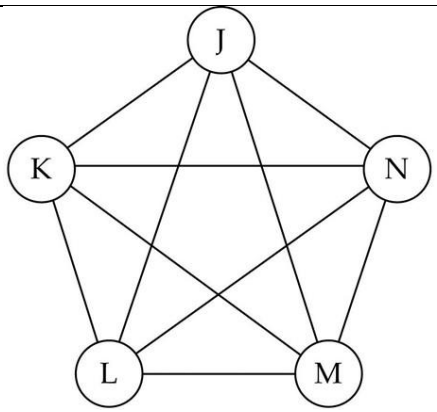
Directed vs. undirected graphs

Directed	Undirected graphs
When the edges in a graph have a direction, the graph is called directed (or *digraph*) عندما يكون لحواف الرسم البياني اتجاه، يسمى الرسم البياني موجهًا (أو ديغراف) **Warning**: if the graph is directed, the order of the vertices in each edge is important تحذير: إذا كان الرسم البياني موجهًا، فإن ترتيب القمم في كل حافة مهم	When the edges in a graph have no direction, the graph is called *undirected* عندما لا يكون لحواف الرسم البياني أي اتجاه، يسمى الرسم البياني غير موجه

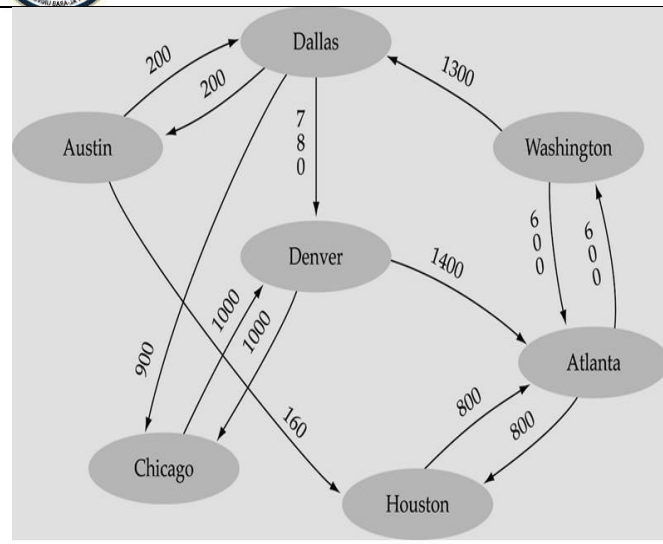

Directed Graph
1
3
11
5
7
9
V(Graph2) = { 1, 3, 5, 7, 9, 11 }
E(Graph2) = {(1,3) (3,1) (5,9) (9,11) (5,7), (9, 9), (11, 1)


directed graph
B
A
D
C
V(Graph1) = { A, B, C, D }
E(Graph1) = { (A, B), (A, D), (B, C), (B, D)

Graph terminology مصطلحات الرسم البياني

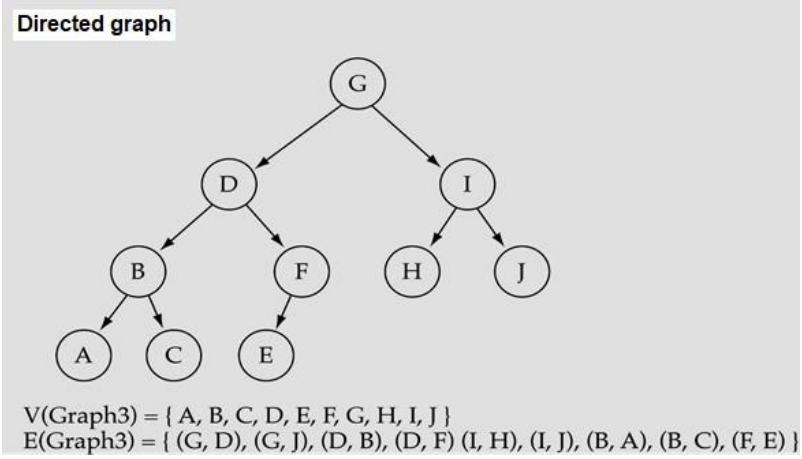
Adjacent node العقدة المجاورة : two nodes are adjacent if they are connected by an edge. تكون العقدتان متجاورتين إذا كانتا متصلتين بواسطة حافة 5 is adjacent to 7 7 is adjacent from 5	
Path: a sequence of edges connect a sequence of vertices. سلسلة من الحواف تربط سلسلة من القمم.	
Complete graph: a graph in which every vertex is directly connected to every other vertex.رسم بياني ترتبط فيه كل قمة مباشرة بكل قمة أخرى .	
The number of edges in a complete directed graph with N vertices are: *N * (N-1)*	
The number of edges in a complete undirected graph with N vertices are: عدد الحواف في الرسم البياني غير الموجه الكامل مع القمم هو N:*N * (N-1) /2*	

Weighted graph: a graph in which each edge carries a value. الرسم البياني الموزون: رسم بياني تحمل فيه كل حافة قيمة.



Trees vs graphs

Trees are special cases of graphsالأشجار هي حالات خاصة من الرسوم البيانية



Graph Representingتمثيل الرسم البياني

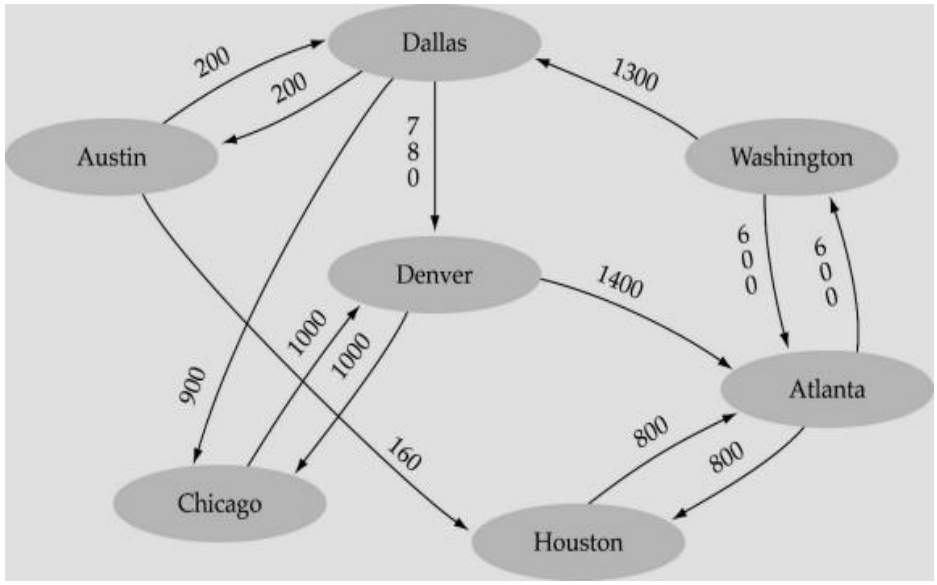
The two most common ways of representing graphs are:

1. Adjacency matrix 1. مصفوفة الجوار

2- Adjacency Listقائمة الجوار

In Adjacency matrix way use array.

- A 1D array is used to represent the vertices. يتم استخدام مصفوفة أحادية الأبعاد لتمثيل القمم.
- A 2D array (adjacency matrix) is used to represent the edges. • يتم استخدام مصفوفة ثنائية الأبعاد (مصفوفة الجوار) لتمثيل الحواف


Dallas
Austin
Washington
Denver
Atlanta
Chicago
Houston
200
200
1300
780
600
600
1400
1000
1000
900
160
800
800

[0]	"Atlanta "
[1]	"Austin "
[2]	"Chicago "
[3]	"Dallas "
[4]	"Denver "
[5]	"Houston "
[6]	"Washington"
[7]	
[8]	
[9]	

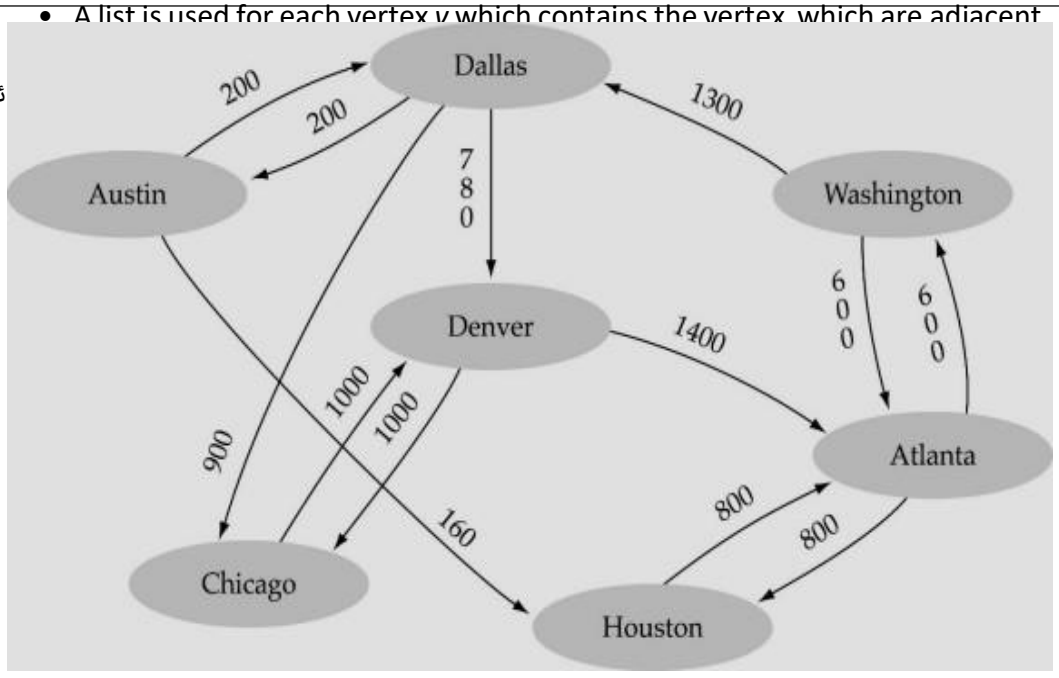
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
[0]	0	0	0	0	0	800	600	•	•	•
[1]	0	0	0	200	0	160	0	•	•	•
[2]	0	0	0	0	1000	0	0	•	•	•
[3]	0	200	900	0	780	0	0	•	•	•
[4]	1400	0	1000	0	0	0	0	•	•	•
[5]	800	0	0	0	0	0	0	•	•	•
[6]	600	0	0	1300	0	0	0	•	•	•
[7]	•	•	•	•	•	•	•	•	•	•
[8]	•	•	•	•	•	•	•	•	•	•
[9]	•	•	•	•	•	•	•	•	•	•

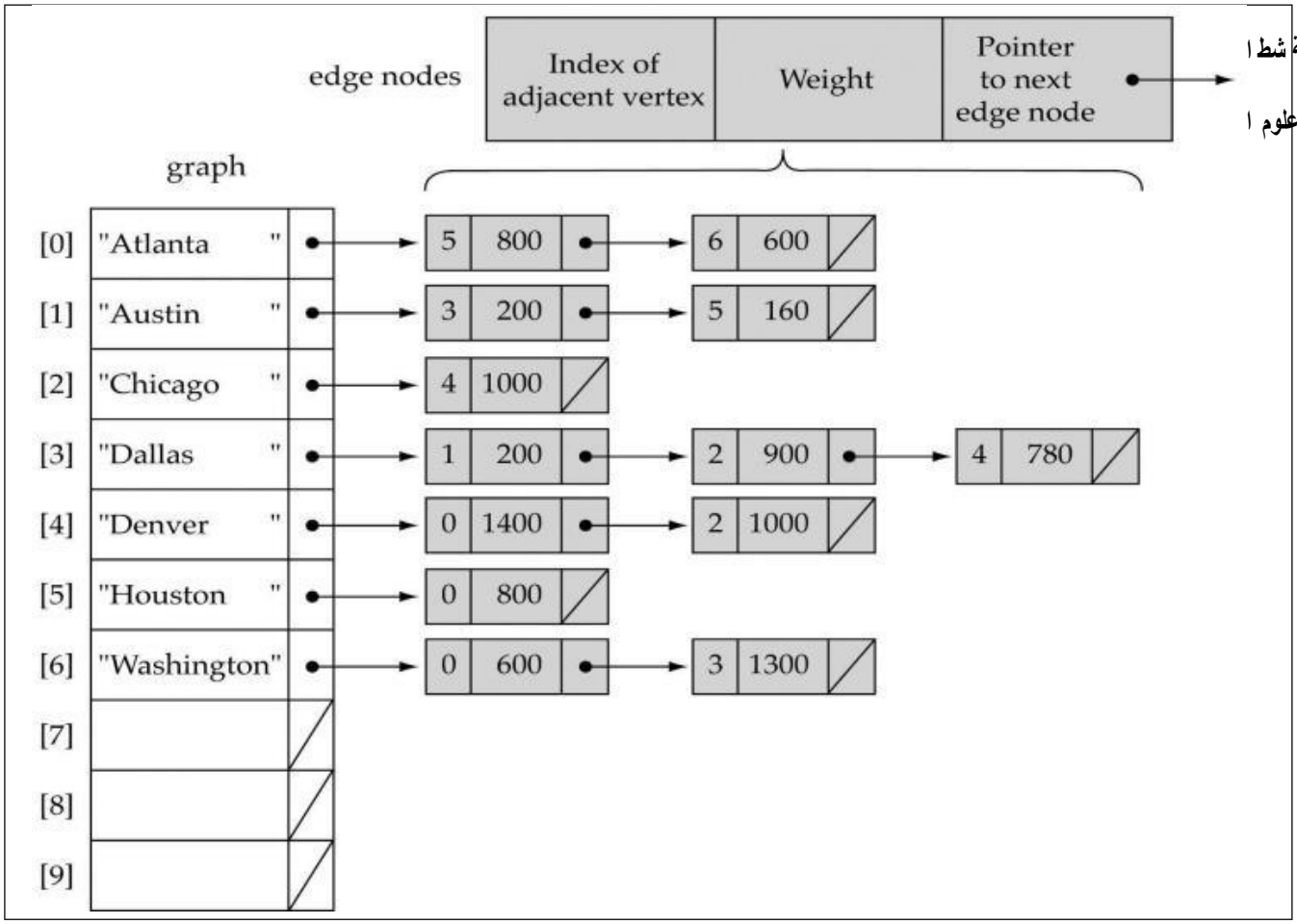
(Array positions marked '•' are undefined)

In Adjacency list way use Linked-list

- A 1D array is used to represent the vertices • يتم استخدام مصفوفة أحادية الأبعاد لتمثيل القمم
- A list is used for each vertex v which contains the vertex which are adjacent

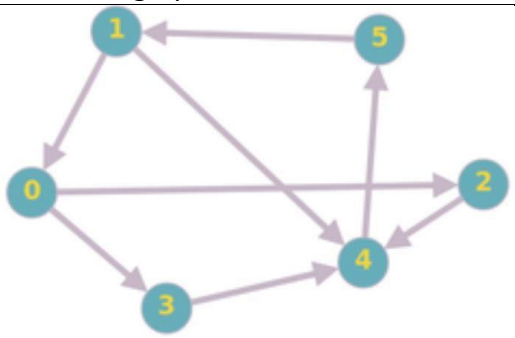
• ئمة لكل قمة



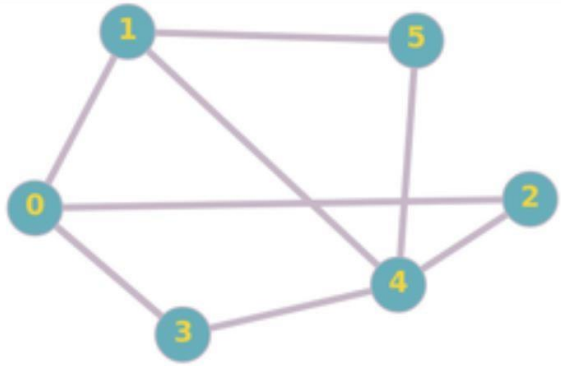


Example

Consider the graphs below.



Graph1



Graph2

1. Use an adjacency list to represent this graph.
2. Use an adjacency matrix to represent this graph

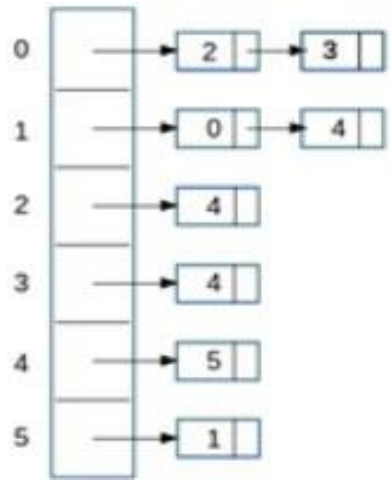
Answer:

Graph1

Adjacency matrix

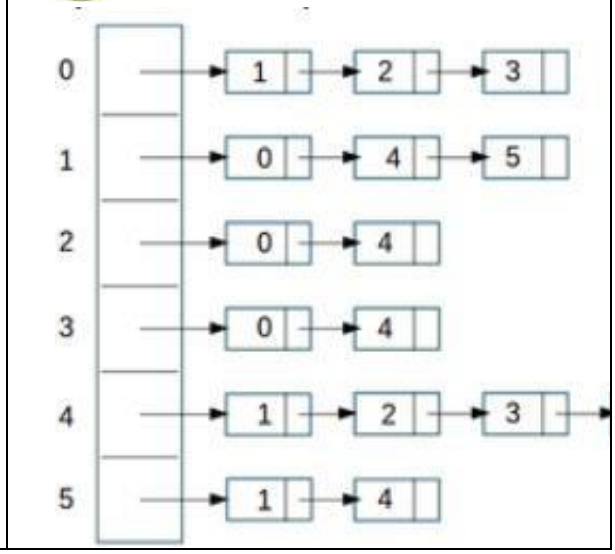
	0	1	2	3	4	5
0	0	0	1	1	0	0
1	1	0	0	0	1	0
2	0	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	0	0	1
5	0	1	0	0	0	0

Adjacency list



	0	1	2	3	4	5
0	0	1	1	1	0	0
1	1	0	0	0	1	1
2	1	0	0	0	1	0
3	1	0	0	0	1	0
4	0	1	1	1	0	1
5	0	1	0	0	1	0

Graph2



Graph traversal عبور الرسم البياني

Graph traversal (also known as graph search) refers to the process of visiting (checking and/or updating) each vertex in a graph. Such traversals are classified by the order in which the vertices are visited. يشير اجتياز الرسم البياني (المعروف أيضًا باسم بحث الرسم البياني) إلى عملية زيارة (فحص و/أو تحديث) كل قمة في الرسم البياني. يتم تصنيف عمليات الاجتياز هذه حسب الترتيب الذي تمت به زيارة القمم.

There are two graph traversal techniques and they are as follows:

1. **DFS (Depth First Search) 1. DFS (**بحث العمق الأول**)**
2. **BFS (Breadth First Search) 2. BFS (**بحث العرض الأول**)**

DFS (Depth First Search)

In DFS traversal, the **stack data structure** is used, which works on the LIFO (Last In First Out) principle. In DFS, traversing can be started from any node, or we can say that any node can be considered as a root node.

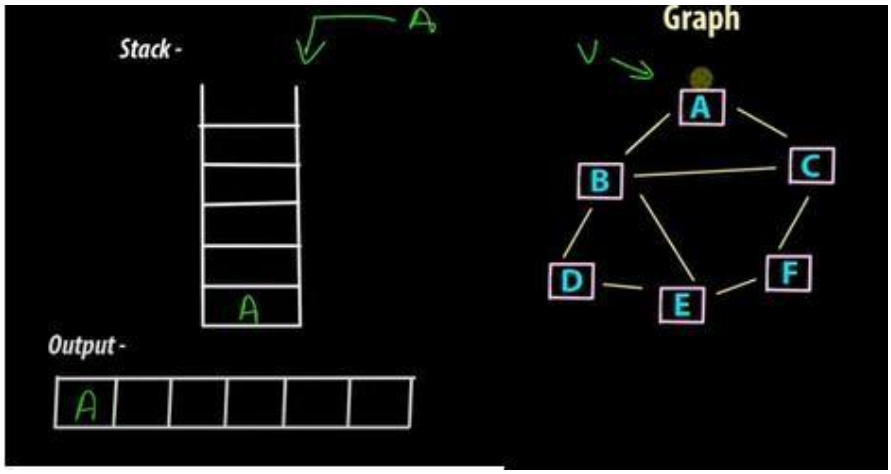
In this algorithm one starting vertex is given, and when an adjacent vertex is found, it moves to that adjacent vertex first and try to traverse in the same manner. في اجتياز DFS ، يتم استخدام بنية بيانات المكدس، والتي تعمل على مبدأ LIFO في DFS (آخر ما يدخل أولاً يخرج). يمكننا DFS ، يمكن بدء العبور من أي عقدة، أو يمكن القول أن أي عقدة يمكن اعتبارها عقدة جذر.

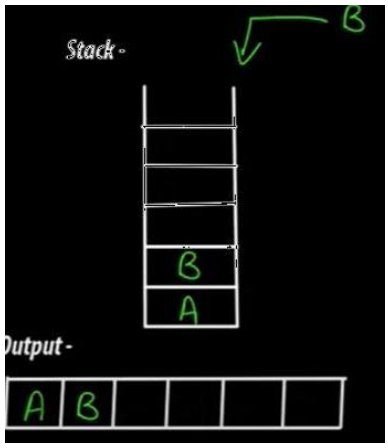
في هذه الخوارزمية يتم إعطاء قمة بداية واحدة، وعندما يتم العثور على قمة مجاورة، فإنها تنتقل إلى تلك القمة المجاورة أولاً وتحاول الاجتياز بنفس الطريقة

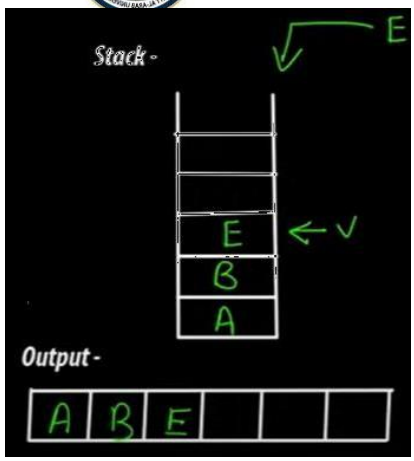
Steps:

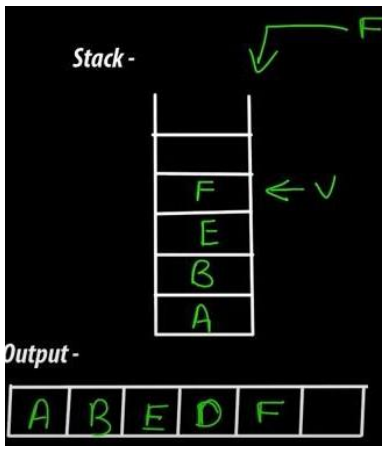
1. **Define a Stack 1.** تحديد المكدس
2. **Set current vertex V 2.** قم بتعيين الرأس الحالي **V**
3. **Add current vertex V to stack 3.** أضف قمة **V** الحالية إلى المكدس
4. **Print current vertex V 4.** طباعة الرأس الحالي **V**
5. **Add any 1 neighbor (unvisited i.e. not in stack previously) of V to stack 5.** أضف أي جار واحد (لم تتم زيارته، أي لم يكن موجودًا في المكدس سابقًا) لـ **V** للمكدس
6. **If current vertex has all its neighbors already visited, pop it from stack & backtrack 6.** إذا كانت القمة الحالية قد زارها جميع جيرانها بالفعل، فقم بإخراجها من المكدس والتراجع
7. **Check remaining vertices in the stack for any unvisited vertices. 7.** تحقق من القمم المتبقية في المكدس بحثًا عن أي قمم لم تتم زيارتها
8. **Repeat from step 4 till stack empty 8.** كرر ذلك من الخطوة 4 حتى تصبح المجموعة فارغة

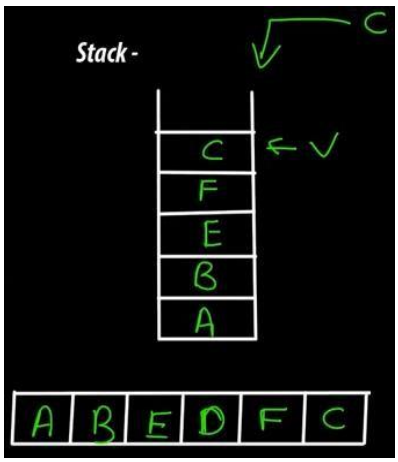
Example 1 : What is the possible DFS traversal for this graph? (Start with A). : ما هو اجتياز DFS المحتمل لهذا الرسم البياني؟ (ابدأ بـ أ).


Stack -
A
A
Graph
V
A
B
C
D
E
F
Output -
A

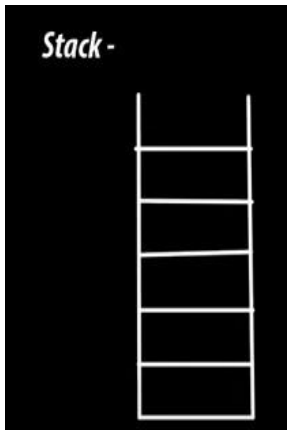




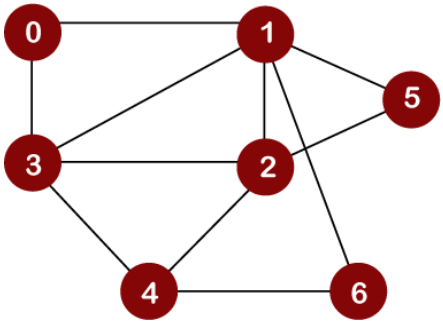


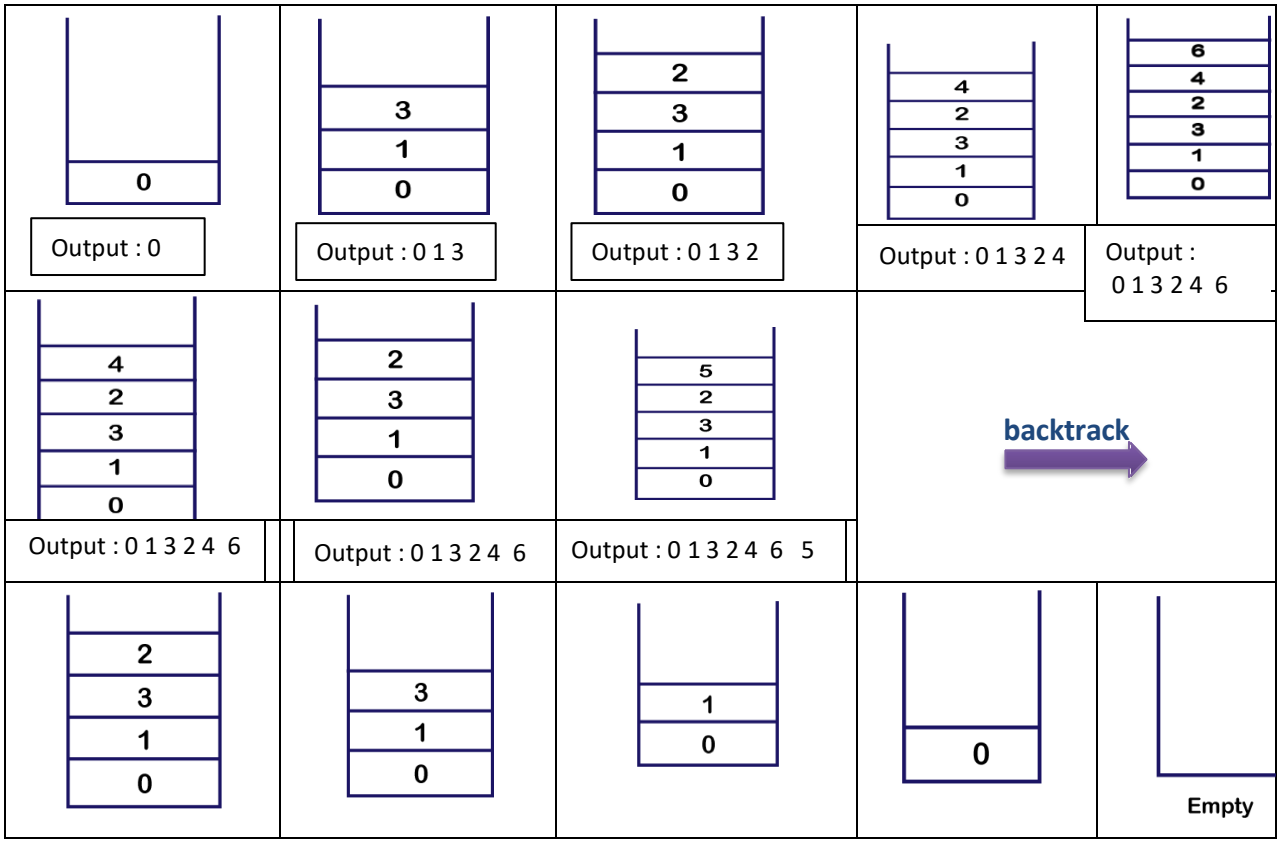


Stack After backtrack



Example 2: What is the possible DFS traversal for this graph? (Start with 0). : المحتمل لهذا DFS ما هو اجتياز
الرسم البياني؟ (ابدأ بـ 0).





BFS (Breadth First Search)

Stands for **BFS**. It is also known as **level order traversal**. The Queue data structure is used for the Breadth First Search traversal.

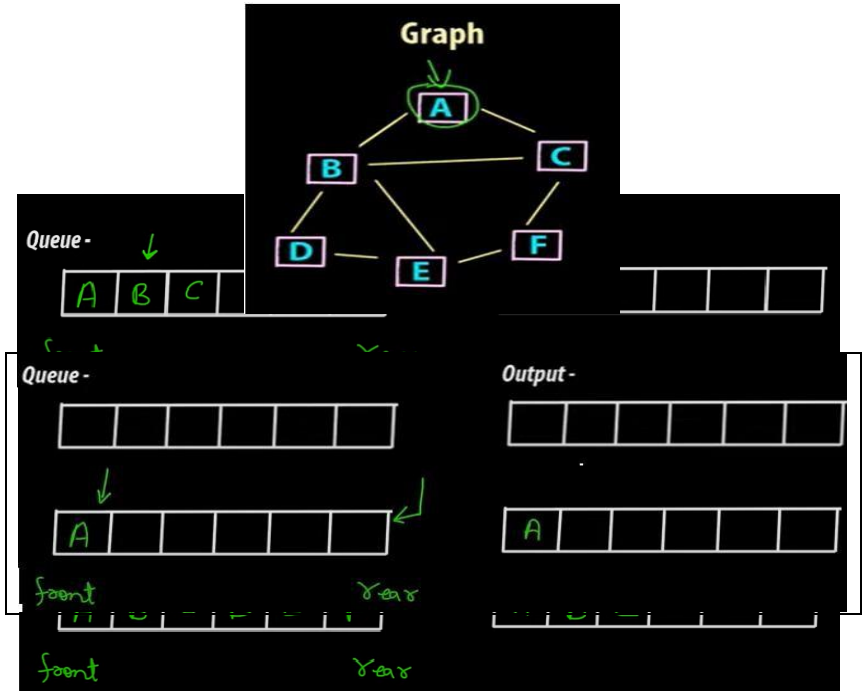
In this traversal algorithm one node is selected and then all of the adjacent nodes are visited one by one. After completing all of the adjacent vertices, it moves further to check another vertex and checks its adjacent vertices again.لتقف على BFS. يُعرف أيضًا باسم Broadth First Search. اجتياز ترتيب المستوى. يتم استخدام بنية بيانات قائمة الانتظار لاجتياز
في خوارزمية الاجتياز هذه، يتم تحديد عقدة واحدة ثم تتم زيارة جميع العقد المجاورة واحدة تلو الأخرى.
بعد الانتهاء من جميع القمم المجاورة، فإنه يتحرك أبعد للتحقق من رأس آخر والتحقق من القمم المجاورة له مرة أخرى

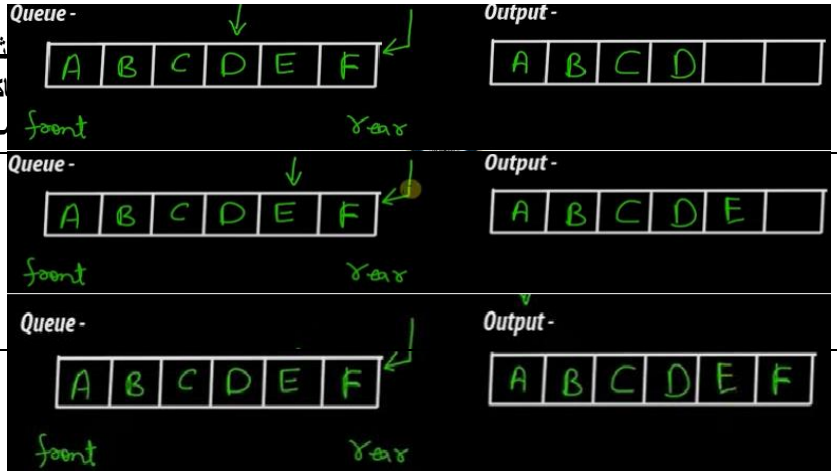
Steps:

1. **Define a Queue 1.** تحديد قائمة الانتظار
2. **Set current vertex V 2.** قم بتعيين الرأس الحالي **V**
3. **Add current vertex V to queue 3.** أضف قمة الرأس **V** الحالية إلى قائمة الانتظار
4. **Print current vertex V 4.** طباعة الرأس الحالي **V**
5. **Add all neighbors (unvisited i.e. previously not in queue) of V to queue (in any order) 5.** أضف جميع جيران **V** (بأي ترتيب) (لم تتم زيارتهم، أي لم يكونوا في قائمة الانتظار سابقًا) إلى قائمة الانتظار
6. **Repeat from step 4 till queue empty 6.** كرر ذلك من الخطوة 4 حتى تصبح قائمة الانتظار فارغة

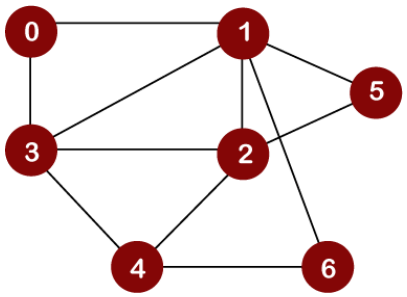
Example 1: What is the possible Breadth First traversal for this graph? (Start with A).

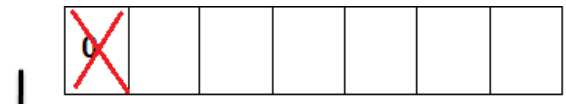
ما هو اجتياز العرض الأول المحتمل لهذا الرسم البياني؟ (ابدأ بـ أ).



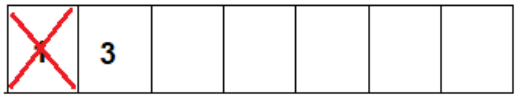


Example 2: What is the possible Breadth First traversal for this graph? (Start with 0).

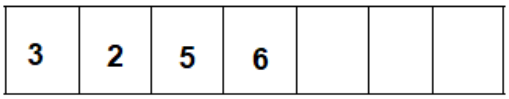


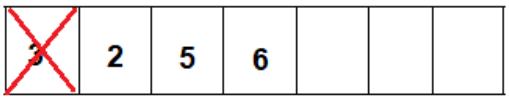


output : 0

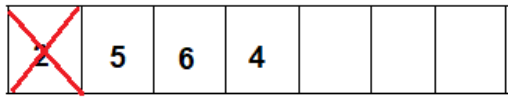


output : 0 1





output : 0 1 3



output : 0 1 3 2

output : 0 1 3 2 5

output : 0 1 3 2 5 6

output : 0 1 3 2 5 6 4

Maps and Hash Tables

What is Maps?

Maps (sometimes called associative arrays) are an abstract data structure (ADT)

- It stores a collection of (key-value) pairs.
- Each key is unique and allows for quick access to values.
- There cannot be duplicate keys.
- The key is used to decide where to store the value in the structure. In other words, the key associated with value can be viewed as the address for that value.
- Maps provide an alternative approach to searching.
- خرائط (والتي تسمى أحيانًا بالمصفوفات الترابطية) عبارة عن بنية بيانات مجردة
- ▪ تخزن مجموعة من أزواج (المفتاح والقيمة)
- ▪ كل مفتاح فريد من نوعه ويسمح بالوصول السريع إلى القيم
- ▪ لا يمكن أن تكون هناك مفاتيح مكررة
- ▪ يتم استخدام المفتاح لتحديد مكان تخزين القيمة في البنية. بعبارة أخرى، يمكن عرض المفتاح المرتبط بالقيمة كعنوان لتلك القيمة
- ▪ توفر الخرائط نهجًا بديلاً للبحث
-

Operations on map

- get(k) : return the value associated with key k if exits in the map
- put(k,k): map the key k to the value v.
- remove(k): remove key k and its value from the map
- size()Returns the number of elements present in the map
- isEmpty()method return True if no key-value is present in the map else return false.

Example: suppose key=integer, value=letter

Operation	M={}
put(5,A)	M={(5,A)}
put(7,B)	M={(5,A), (7,B)}
put(2,C)	M={(5,A), (7,B), (2,C)}
put(8,D)	M={(5,A), (7,B), (2,C), (8,D)}
put(2,E)	M={(5,A), (7,B), (2,E), (8,D)}
get(7)	return B
get(4)	return null
get(2)	return E

remove(5)	M={(7,B), (2,E), (8,D)}
size()	return 3
isEmpty()	return false
remove(2)	M={(7,B), (8,D)}
get(2)	return null

Hashing

The purpose of hashing is to achieve search, insert and delete an element in complexity O(1) [in constant time]. In this technique, convert a range of key values into a range of indexes in the hash table by using a hashing function. الغرض من التجزئة هو تحقيق البحث عن عنصر وإدراجه وحذفه في [في وقت ثابت]. في هذه التقنية، يتم تحويل مجموعة من قيم المفتاح إلى مجموعة من الفهارس في جدول تعقيد O(1) التجزئة باستخدام دالة التجزئة.

Main concepts of hashing

- The hash table
- The hash function

المفاهيم الرئيسية للتجزئة

- جدول التجزئة
- دالة التجزئة

Hash Table

A hash table is a data structure that provides a mapping from keys to values (also called hash values) using a technique called hashing.

- Keys must be unique but the values may be repeated.
- We refer to these values as key-values pairs.
- key-values pairs can be any type as integer, string .., they are need to hashed by using hash function.
- جدول التجزئة هو بنية بيانات توفر تعيينًا من المفاتيح إلى القيم (وتسمى أيضًا قيم التجزئة) باستخدام تقنية تسمى التجزئة .
- يجب أن تكون المفاتيح فريدة ولكن يمكن تكرار القيم.
- نشير إلى هذه القيم على أنها أزواج مفتاح-قيم.
- يمكن أن تكون أزواج المفتاح-القيم من أي نوع مثل عدد صحيح أو سلسلة...، ويجب تجزئتها باستخدام دالة التجزئة.
-

Hash function

A hash function (h(k)) is a function that maps a key ‘k’ to number in a fixed range.

A good hash function satisfies two basic properties:

1) It should be very fast to compute.
2) It should reduce (not prevent) the number of collisions.

Notes: The size of the table N and the hash function are decided by the user
دالة التجزئة (h(k)) هي دالة تقوم بربط المفتاح "k" برقم في نطاق ثابت.

تلبي دالة التجزئة الجيدة خاصيتين أساسيتين:
1) يجب أن تكون سريعة جدًا في الحساب.
2) يجب أن تقلل (وليس تمنع) عدد التصادمات.

ودالة التجزئة بواسطة المستخدم N ملاحظات: يتم تحديد حجم الجدول

In the following different methods to find a good hash function:

1. Division Method

If k is a key and m is the size of the hash table, the hash function h() is calculated as:

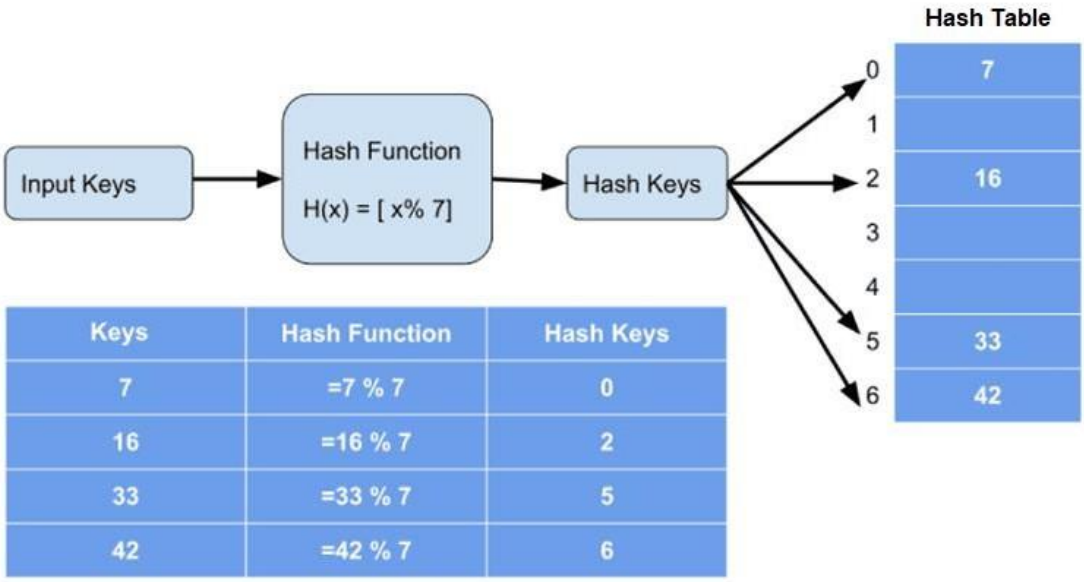
h(k) = k mode m (**best for m is prime**)

2. Multiplication Method

h(k) = ⌊m(kA mod 1)⌋ , (**best for m = 2^n**)where

- k is a key and m is the size of the hash table
- ⌊ ⌋ gives the floor value
- A lies between 0 and 1

How does Hashing in Data Structure Work?



Keys	Hash Function	Hash Keys
7	=7 % 7	0
16	=16 % 7	2
33	=33 % 7	5
42	=42 % 7	6

Hash table size

- By "size" of the hash table we mean how many slots or buckets it has.
- Choice of hash table size depends in part on choice of hash function, and collision resolution strategy.
- But a good general is:
  [The hash table should be an array with length about 1.3 times the maximum number of keys that will actually be in the table, and Size of hash table array should be a prime number.]
- If you underestimate the number of keys, you may have to create a larger table and rehash the entries when it gets too full; if you overestimate the number of keys, you will be wasting some space
- نعني بـ "حجم" جدول التجزئة عدد الفتحات أو الدلاء التي يحتوي عليها.
- يعتمد اختيار حجم جدول التجزئة جزئيًا على اختيار دالة التجزئة واستراتيجية حل التصادم.
- ولكن القاعدة العامة الجيدة هي:
- [يجب أن يكون جدول التجزئة عبارة عن مصفوفة بطول حوالي 1.3 مرة من الحد الأقصى لعدد المفاتيح التي ستكون في الجدول بالفعل، ويجب أن يكون حجم مصفوفة جدول التجزئة عددًا أوليًا.]
- إذا قللت من تقدير عدد المفاتيح، فقد تضطر إلى إنشاء جدول أكبر وإعادة تجزئة الإدخالات عندما يصبح ممتلئًا للغاية؛ إذا بالغت في تقدير عدد المفاتيح، فسوف تهدر بعض المساحة

Why prefer table size is prime number?لماذا يفضل حجم الجدول أن يكون عددا أوليا؟

If size of hash table is prime number will produce the most wide-spread distribution of keys in hash table.

But if it not prime, every key that shares a common factor with the table size will be hashed into a value that is a multiple of this factor

إذا كان حجم جدول التجزئة عددًا أوليًا، فسيؤدي ذلك إلى أكبر توزيع للمفاتيح في جدول التجزئة.

ولكن إذا لم يكن أوليًا، فسيتم تجزئة كل مفتاح يشترك في عامل مشترك مع حجم الجدول إلى قيمة مضاعفة لهذا العامل.

Why use mod in hash function?لماذا استخدام mod في دالة التجزئة؟

Generally, hash functions calculate an integer value from the key, to ensure this integer value is

within the length of the hash table. The result will range somewhere from 0 to the table_size-1.
بشكل عام، تحسب وظائف التجزئة قيمة عددية صحيحة من المفتاح، وذلك لضمان أن تكون هذه القيمة الصحيحة ضمن طول جدول التجزئة. ستتراوح النتيجة بين 0 و table_size-1.

Hash Collisionتصادم التجزئة

Hashing in data structure falls into a collision when the hash function generates the same location (hash value) for two keys. The collision creates a problem because each location in a hash table is supposed to store only one value.

يقع التجزئة في بنية البيانات في تصادم عندما تولد دالة التجزئة نفس الموقع (قيمة التجزئة) لمفتاحين. يخلق التصادم مشكلة لأن كل موقع في جدول التجزئة من المفترض أن يخزن قيمة واحدة فقط.

Collision Resolution Techniquesتقنيات حل التصادم

- It is process of finding an alternate location.
- The collision resolution techniques can be named as-

1. Open Hashing (Closed Addressing)
   Separate Chaining)
2. Closed Hashing (Open Addressing)
   A. Linear Probing
   B. Quadratic Probing
   C. Double Hashing

إنها عملية البحث عن موقع بديل.
يمكن تسمية تقنيات حل التصادم على النحو التالي-
1. التجزئة المفتوحة (التوجيه المغلق)

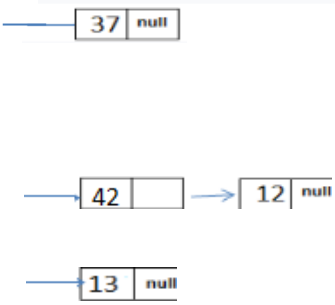
(التسلسل المنفصل
2. التجزئة المغلقة (التوجيه المفتوح)
أ. الفحص الخطي
ب. الفحص التربيعي
ج. التجزئة المزدوجة

Open Hashing - Separate Chaining

In chaining, if a hash function produces the same location for multiple elements, these elements are stored in the same location by using a linked list.
Example 1: Use division method and opened hashing (closed addressing)(**chaining**) to insert the elements below into a hash table of size **10** .

3, 2, 42, 4, 12, 14,17,13,37

Hash Table



Key(k)	Location(h(key))	probe
3	3 % 10 = 3	1
2	2 % 10 = 2	1
42	42 % 10 = 2	2
12	12 % 10 = 2	3
14	14 % 10 = 4	1
17	17 % 10 = 7	1
13	13 % 10 = 3	2
37	37 % 10 = 7	2

0	
1	
2	2
3	3
4	14
5	
6	
7	17
8	
9	

Example 2:
Use opened hashing (**chaining**) to insert the elements below into a hash table of size 9.
7, 42, 25, 70, 14, 38, 8, 21, 34, 11, 48, 26, 93, 125

Open Addressing

Unlike chaining, open addressing doesn't store multiple elements into the same location. Here, each location is either filled with a single key or left **null**.

A. Linear Probing

In this technique search the next empty location in the hash table by looking into the next location until we find an empty location.

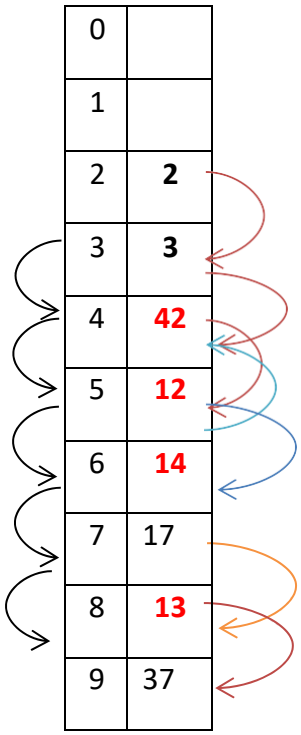
Probe: The list of locations which a method for open addressing produces as alternatives in case of a collision.

[next location = (collision location) % table_size , i=0..table_size-1]

Example 1: Use division method and closed hashing (opened addressing)(**Linear Probing**) to insert the elements below into a hash table of size **10**.

3, 2, 42, 4, 12, 14,17,13,37

Hash Table



Key(K)	Location(h(key))	probe
3	3 % 10 = 3	1
2	2 % 10 = 2	1
42	42 % 10 = 2	3
12	12 % 10 = 2	4
14	14 % 10 = 4	3
17	17 % 10 = 7	1
13	13 % 10 = 3	6
37	37 % 10 = 7	3

Example 2:
Use closed hashing (**Linear Probing**) to insert the elements below into a hash table of size 9.
7, 42, 25, 70, 14, 38, 8, 21, 34, 11, 48, 26, 93, 125

B. Quadratic Probing

It works similar to linear probing but the spacing between the location is increased (greater than one) by using the following relation:

[next location = (collision location+i^2) % table_size , i=0..table_size-1]

Example 1: Use division method and closed hashing (opened addressing)(**Quadratic Probing**) to insert the elements below into a hash table of size **10**.

3, 2, 42, 4, 12, 14,17,13,37

Hash Table

0	**12**
1	
2	**2**
3	**3**
4	**14**
5	
6	**42**
7	17
8	**37**
9	**13**

Key(k)	Location(h(key))	probe
3	3 % 10 = 3	1
2	2 % 10 = 2	1
42	42 % 10 = 2	3
12	12 % 10 = 2	4
14	14 % 10 = 4	1
17	17 % 10 = 7	1
13	13 % 10 = 3	5
37	37 % 10 = 7	2

Example 2:
Use closed hashing (**Linear Probing**) to insert the elements below into a hash table of size 9.
7, 42, 25, 70, 14, 38, 8, 21, 34, 11, 48, 26, 93, 125

Non-linear Data Structure (Abstract Data Structure-ADT):

Data structures where data elements are not arranged sequentially or linearly are called non-linear data structures. In a non-linear data structure, single level is not involved. Therefore, we can't traverse all the elements in single run only. Whereas in non-linear data structure, multiple levels are involved, then, we can traverse all the elements.

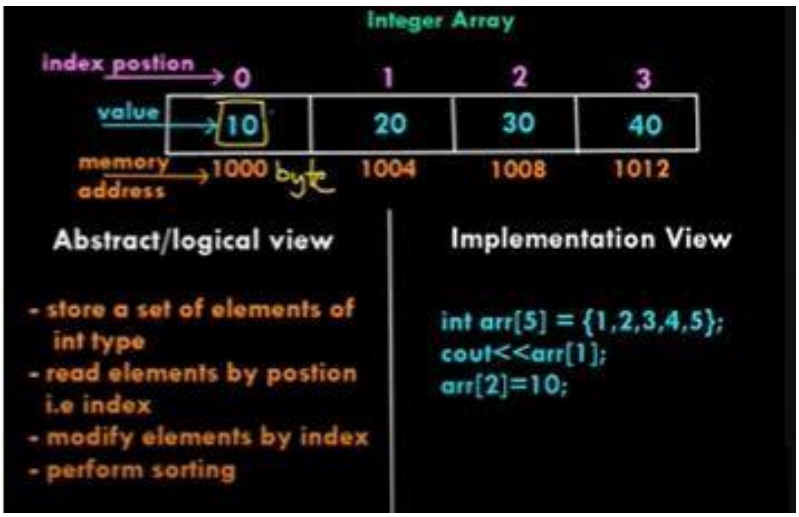
Examples of none-linear data structure are trees and graphs.

تُسمى هياكل البيانات التي لا يتم ترتيب عناصر البيانات فيها بشكل تسلسلي أو خطي بهياكل البيانات غير الخطية. في هيكل البيانات غير الخطي، لا يتم إشراك مستوى واحد. لذلك، لا يمكننا عبور جميع العناصر في تشغيل واحد فقط. بينما في هيكل البيانات غير الخطي، يتم إشراك مستويات متعددة، وبالتالي، يمكننا عبور جميع العناصر.

ومن أمثلة هياكل البيانات غير الخطية الأشجار والرسوم البيانية

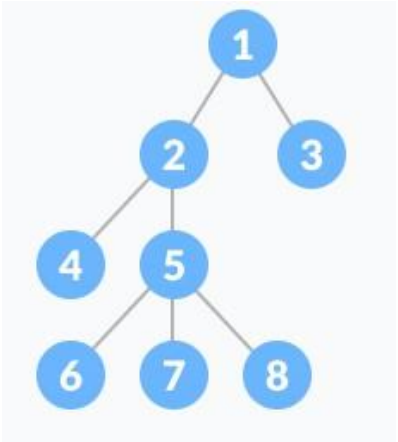
ADTs are entities that are definitions of data and operation but not have implements details.

ADTs عبارة عن كيانات تمثل تعريفات للبيانات والتشغيل ولكنها لا تحتوي على تفاصيل التنفيذ



Tree Data Structure

A tree (upside down) is a nonlinear hierarchical data structure that consists of nodes connected by edges with parent-child relationship. الشجرة (المقلوبة) عبارة عن بنية بيانات هرمية غير خطية تتكون من عقد متصلة بواسطة حواف بعلاقة الوالد بالطفل



- Each node (except the top node) has a parent and zero or more children nodes. تحتوي كل عقدة (باستثناء العقدة العلوية) على عقدة رئيسية وصفر أو أكثر من العقد الفرعية.
- Each node has data in any type as char, number, and string. تحتوي كل عقدة على

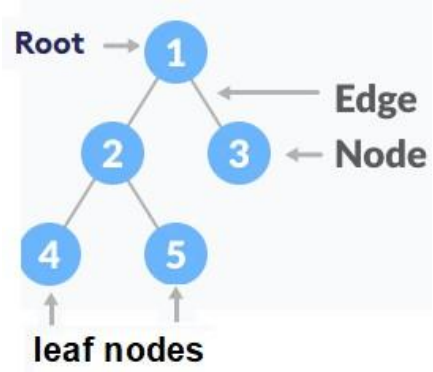
بيانات بأي نوع مثل string.وnumber وchar

- First node that starts with the tree called root.أول عقدة تبدأ بالشجرة تسمى الجذر .

Tree Terminologies

Node is an entity that contains a value and pointers to its child nodes. العقدة هي كيان يحتوي على قيمة ومؤشرات إلى العقد الفرعية الخاصة به

و



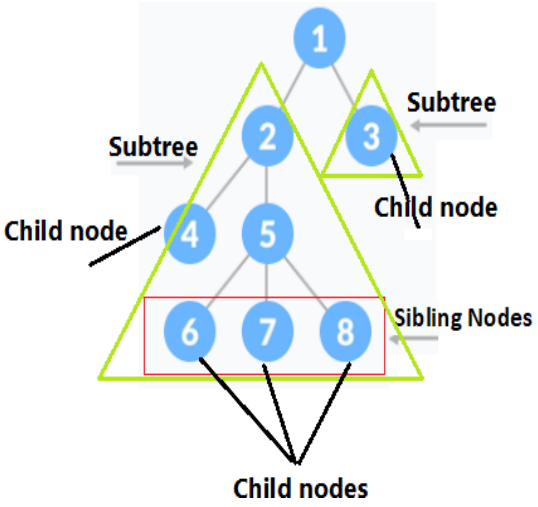
Leaf node (also called as external nodes) is the node which does not have a child.

Non-leaf (also called as internal node) is a node with at least one child.
[The root node is also said to be Internal Node]

Edge is the link between any two nodes.الحافة هي الرابط بين أي عقدتين .

Root is the topmost node of a tree (without any parent).هي العقدة العلوية في الشجرة (بدون أي أب)

Sibling Nodes: nodes that have the same parent.العقد الشقيقة: العقد التي لها نفس الوالد .



Ancestor node (N): any predecessor node on a path from node N to root. [The root node doesn't have any ancestors. **[2,1 are ancestor of 5]**
(N): العقدة السلفية إلى **N** أي عقدة سابقة على مسار من العقدة الجذر. [لا تحتوي العقدة الجذرية على أي أسلاف. [2،1 هي سلف لـ 5**]**

Descendant node (N): any successor node on a path from node N to leaf **[6, 7, 8 are descendant of 5]**
(N): العقدة المنحدرة إلى **N** أي عقدة خليفة على المسار من العقدة الورقة [6، 7، 8 هي من نسل 5]

Child node: If the node is a descendant of any node, then the node is known as a child node.
العقدة الفرعية: إذا كانت العقدة من نسل أي عقدة، فإن العقدة تُعرف باسم العقدة الفرعية.

Parent: If the node contains any sub-node, then that node is said to be the parent of that sub-node. **[5 is parent of 6,7,8]** إذا كانت العقدة تحتوي على أي عقدة فرعية، فيقال أن هذه العقدة هي العقدة الأم لتلك العقدة الفرعية. [5 هي العقدة الأم لـ 6،7،8]

Subtree : tree consisting of a node and its descendants شجرة فرعية: شجرة تتكون من عقدة وأحفاده

ا

Path: the sequence of nodes from one node (source node) to another node(destination node).

مسار: هو تسلسل العقد من عقدة واحدة (عقدة المصدر) إلى عقدة أخرى (عقدة الوجهة).

Height of a Node N: is the longest path from node N to leaf. **[Height of leaf node = 0]**

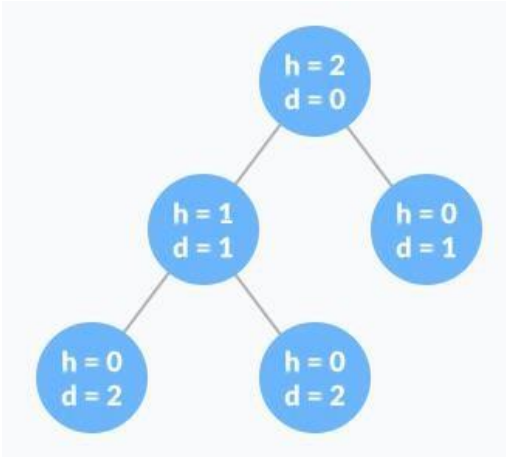
Depth of a Node N: is the number of edges from the root to the node N. In other words: it is the number of nodes it passes from root through down to node N. **[depth of root node = 0]**

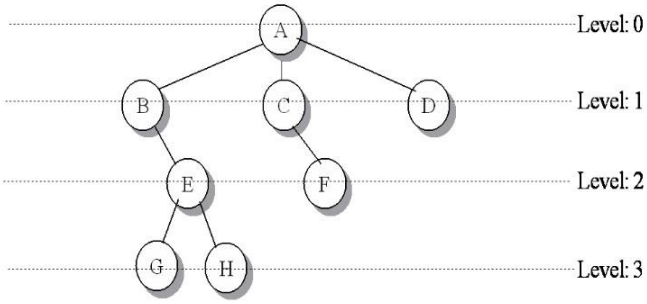
Height of a Tree: is height of the root node.

Depth of a Tree: the total number of edges from root node to a leaf node in the longest path.[maximum depth of any node]

Degree of a Node N: **number of children of that node**. **[degree of leaf =0]**

Degree of a Tree: maximum degree among of nodes.





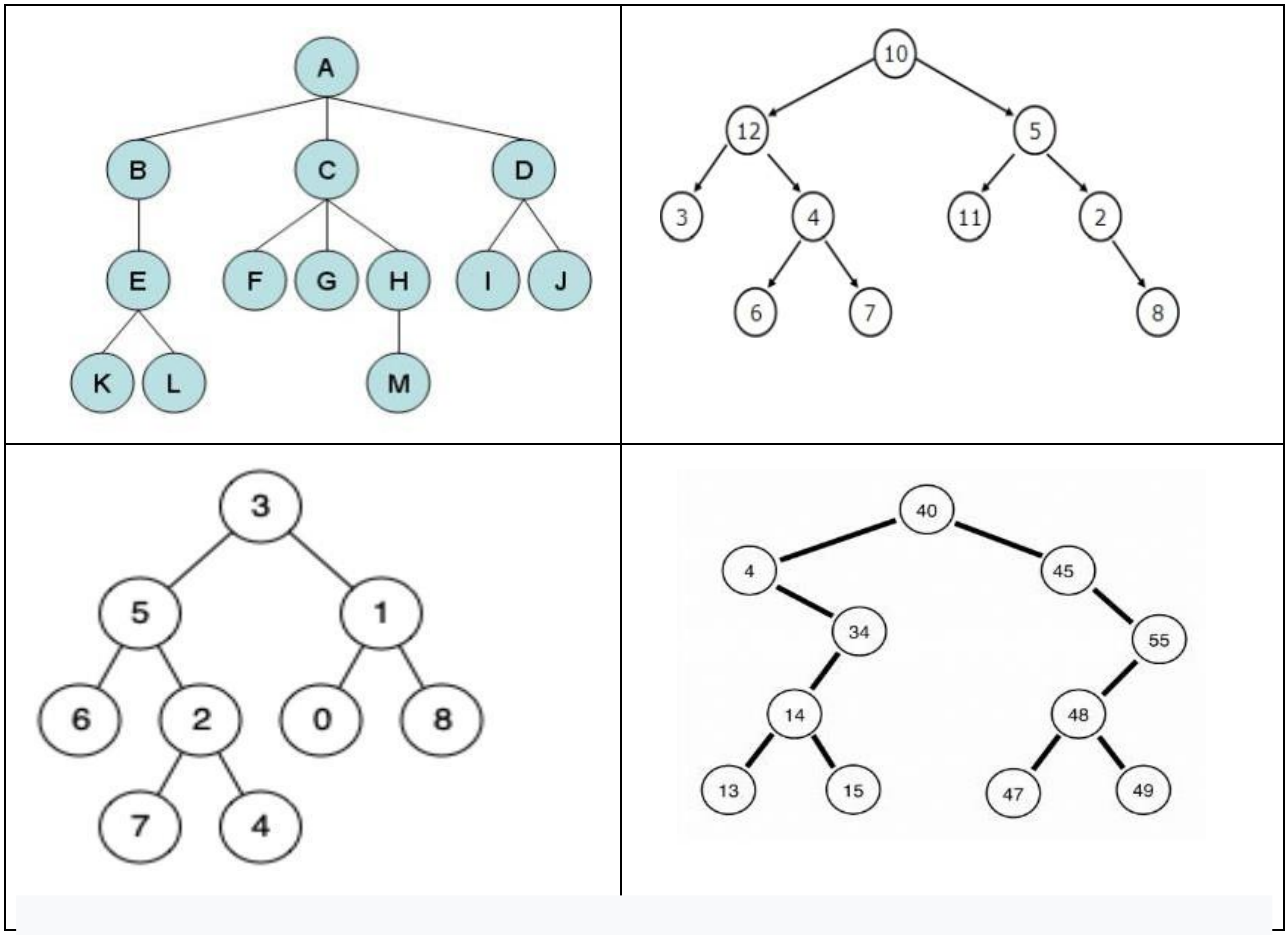
Level

the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on...

Depth of each node in any level equal to that level number.

Question: Consider the trees below.

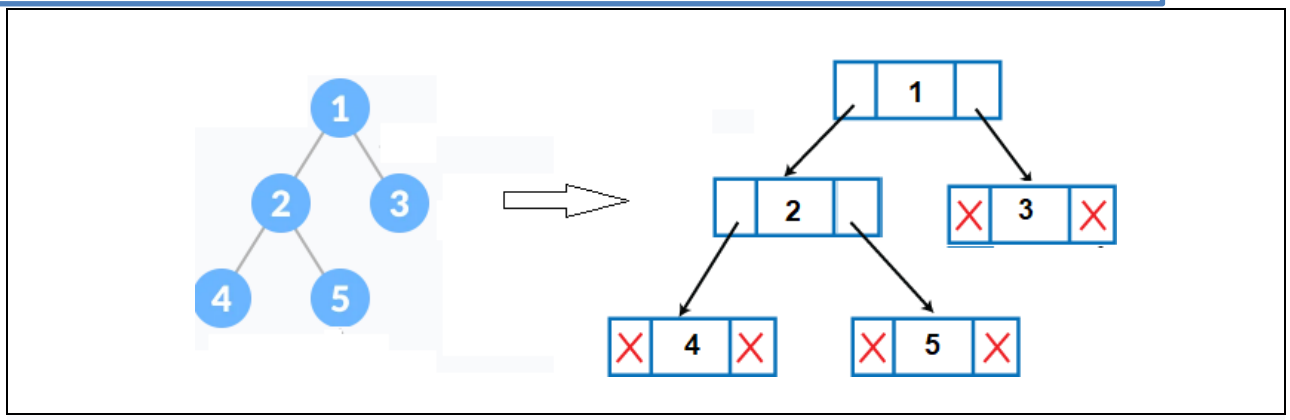
1. Which node is the root? 1. أي عقدة هي الجذر؟
2. What are the internal nodes? 2. ما هي العقد الداخلية؟
3. How many descendants does node (?) have? 3. كم عدد أحفاد العقدة (؟)؟
4. What is the depth of node (?)?4. ما هو عمق العقدة (؟)؟
5. What are the internal nodes? 5. ما هي العقد الداخلية
6. How many descendants does node (?) have? 6. كم عدد أحفاد العقدة (؟).
7. How many ancestors does node (?) have?كم عدد أسلاف العقدة
8. What is the depth of node (?)?
9. What are the siblings of node (? ما هم أشقاء العقدة
10. What are the siblings of node (?)?
11. Which nodes are in the subtree rooted at node (?)?11. ما هي العقد الموجودة في الشجرة الفرعية المتجذرة في العقدة
12. What is the height of the tree? 12. ما هو ارتفاع الشجره



A Linked Structure for Tree

The tree data structure can be created by creating the nodes dynamically with linked list.
Atree node is represented by an object storing يمكن إنشاء بنية بيانات الشجرة عن طريق إنشاء العقد ديناميكيًا باستخدام القائمة المرتبطة.
بواسطة كائن يخزن Atree يتم تمثيل عقدة

- Element
- A parent node
- Asequence of children nodes ▪ تسلسل العقد الأبناء

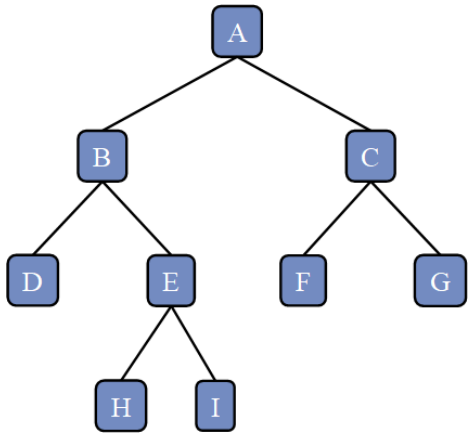


Binary Tree

Binary tree is a tree data structure in which each parent node can have at most two children. [Children ordered pair (left and right].

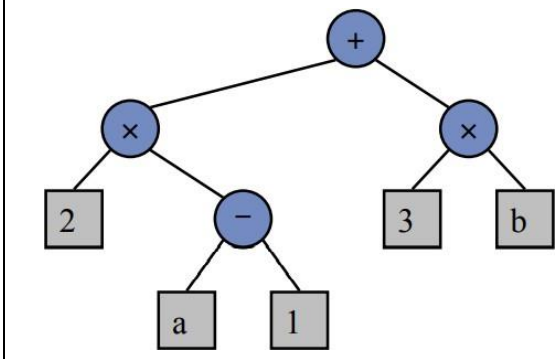
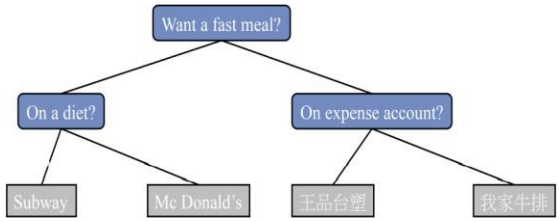
Abinary tree is a tree with the following properties:

- Each internal node has at most two children (left child and right child)
- The children of a node are an ordered pair (left and right)

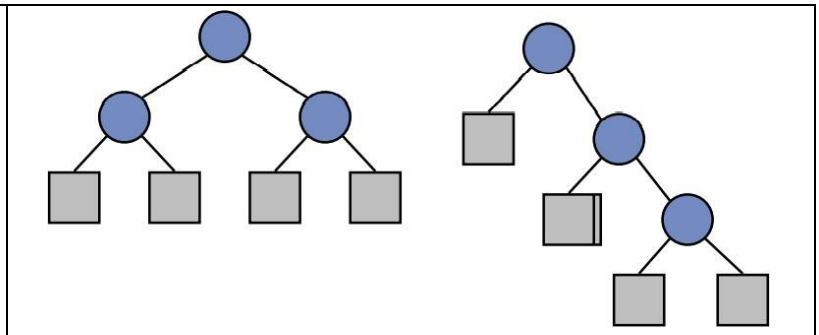


Applications of Binary Tree

1. To build Arithmetic Expression Tree
2. To build Decision Tree

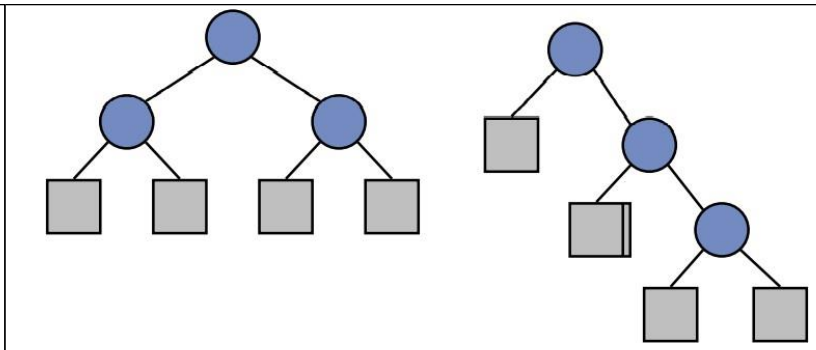
Binary tree associated with an arithmetic expression ▪ internal nodes: operators ▪ external nodes: operands Example : arithmetic tree for the expression : (2 X (a - 1) + (3 X b))	
Binary tree associated with a decision process ▪ internal nodes: questions with yes/no answer ▪ external nodes: decisions Example: dining decision	

Proper Binary Trees الأشجار الثنائية الصحيحة

Each internal node has exactly 2 children كل عقدة داخلية لديها طفلين بالضبط	
▪ n :number of total nodes ▪ e :number of external nodes ▪ i :number of internal nodes ▪ h :height (maximum depth of a node)	Properties: ***1. e = i +*** 1 ***2. n =*** 2***e*** - 1 ***3. h <=i*** ***4. h <=*** (***n*** - 1)/2 ***5. e <=*** 2^{h} 6. ***h >=*** $\log_2 e$ ***7. h >=*** $\log_2$ (***n +*** 1) - 1

from the two tress:
n = 7 , e = 4 , i = 3, h=2

- ✓ ***e = i +*** 1
- ✓ ***n =*** 2***e*** - 1
- ✓ ***h <=i***
- ✓ ***h*** <= (***n*** - 1)/2
- ✓ ***5. e*** <= 2^{h}
- ✓ 6. ***h >***= $\log_2$ ***e=2***
- ✓ ***7. h >***= $\log_2$ (***n +*** 1) – 1=2



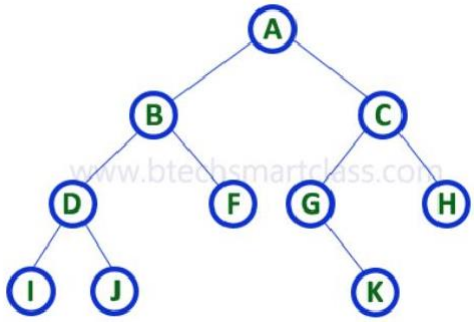
Binary Tree Representations

A binary tree data structure is represented using two methods. Those methods are as follows:يتم تمثيل بنية بيانات الشجرة الثنائية باستخدام طريقتين. وتلك الأساليب هي كما يلي

1. **Array Representation**تمثيل المصفوفة .1
2. **Linked List Representation**تمثيل القائمة المرتبطة

Array Representation

Consider the following binary tree:



In array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree. في تمثيل المصفوفة لشجرة ثنائية، نستخدم مصفوفة أحادية البعد (مصفوفة ذات أبعاد واحدة) لتمثيل شجرة ثنائية

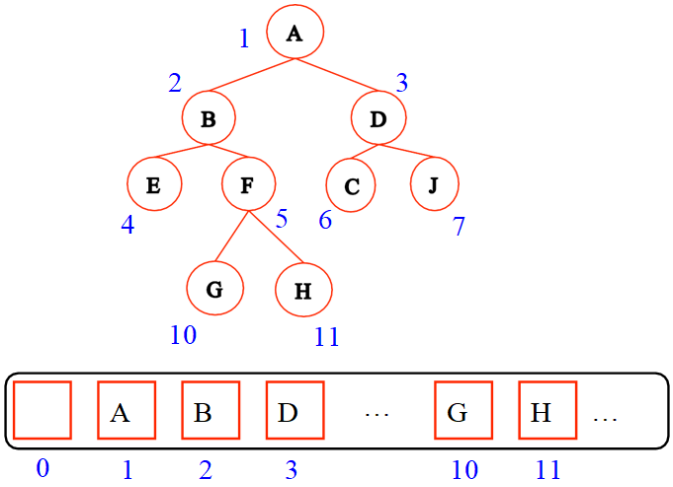
Consider the above example of a binary tree and it is represented as follows: خذ بعين الاعتبار المثال أعلاه للشجرة الثنائية ويتم تمثيله على النحو التالي:

A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

- A[0] is always empty
- A[i] is empty if there is no node in the ith position i فارغًا إذا لم تكن هناك عقدة في الموضع A[i] يكون

- حجم المصفوفة N 2 هو(h+1)
- The array size N is 2(h+1)

Consider the following binary tree:



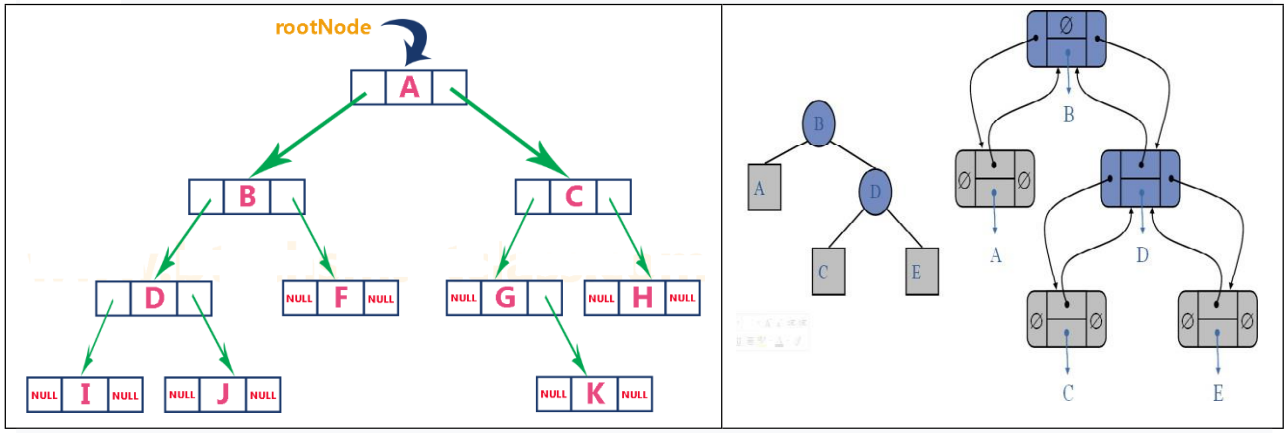
Linked List Representation

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address.

In this linked list representation, a node has the following structure... نستخدم قائمة مرتبطة مزدوجة لتمثيل شجرة ثنائية. في القائمة المرتبطة المزدوجة، تتكون كل عقدة من ثلاثة حقول. الحقل الأول لتخزين عنوان الطفل الأيسر، والثاني لتخزين البيانات الفعلية والثالث لتخزين عنوان الطفل الأيمن.

في تمثيل القائمة المرتبطة هذا، تحتوي العقدة على البنية التالية

The above example of the binary tree represented using Linked list representation is shown as follows:

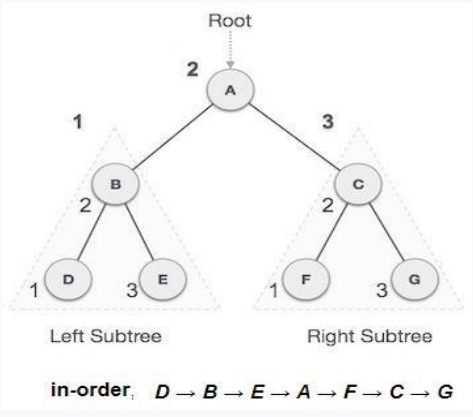
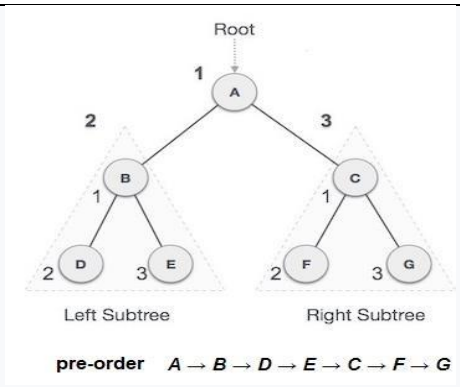
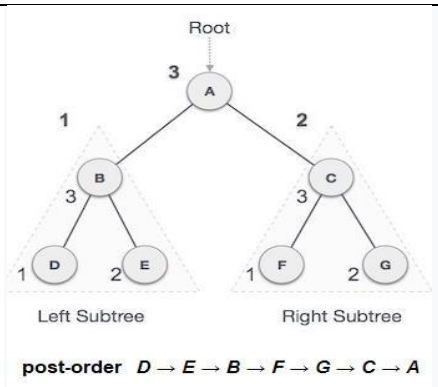


Operations of Tree

1. Insert
2. Delete
3. Search
4. Traversal

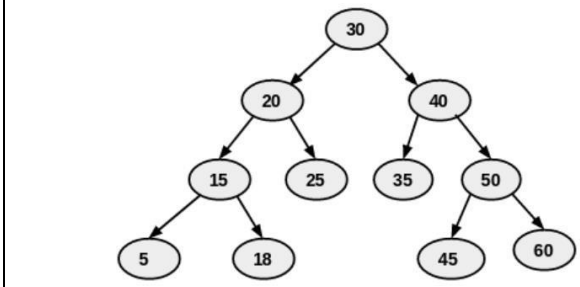
Tree Traversal

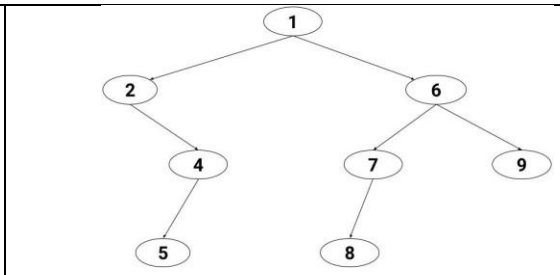
- In order to perform any operation on a tree, you need to reach to the specific node.
- Traversing a tree means visiting every node in the tree. ▪ من أجل إجراء أي عملية على شجرة، عليك الوصول إلى العقدة المحددة.
  1. **inorder**
  2. **preorder**
  3. **postorder.**

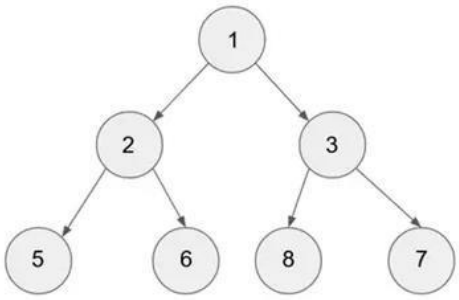
Inorder traversal	Pre-order traversal	Post-order traversal
1. visit all the nodes in the left subtree 2. Visit the root node 3. Visit all the nodes in the right subtree vc	1. Visit the root node 2. Visit all the nodes in the left subtree 3. Visit all the nodes in the right subtree	1. Visit all the nodes in the left subtree 2. Visit all the nodes in the right subtree 3. Visit the root node
		

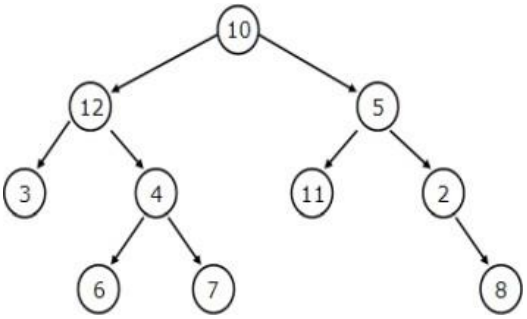
Consider the trees below.

1. What is the preorder traversal of the tree?
2. What is the inorder traversal of the tree?
3. What is the postorder traversal of the tree?


30
20
40
15
25
35
50
5
18
45
60


1
2
6
4
7
9
5
8

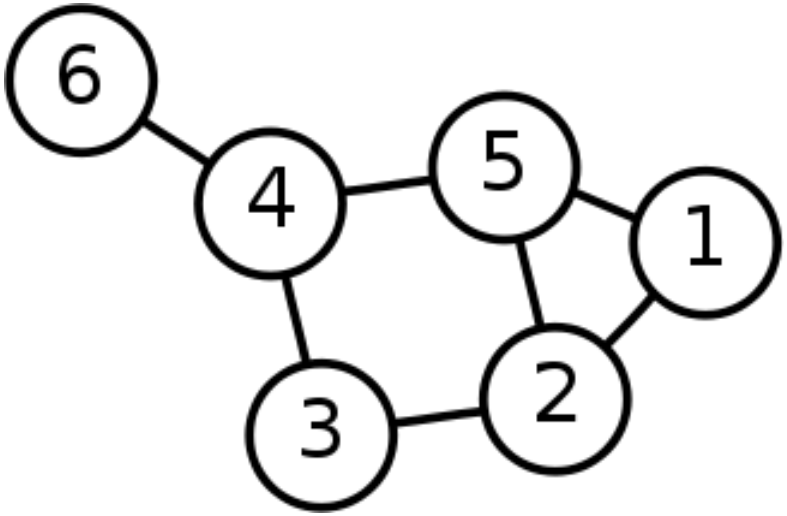

1
2
3
5
6
8
7


10
12
5
3
4
11
2
6
7
8

Dijkstra’s Algorithm

Single-Source Shortest Path Problem

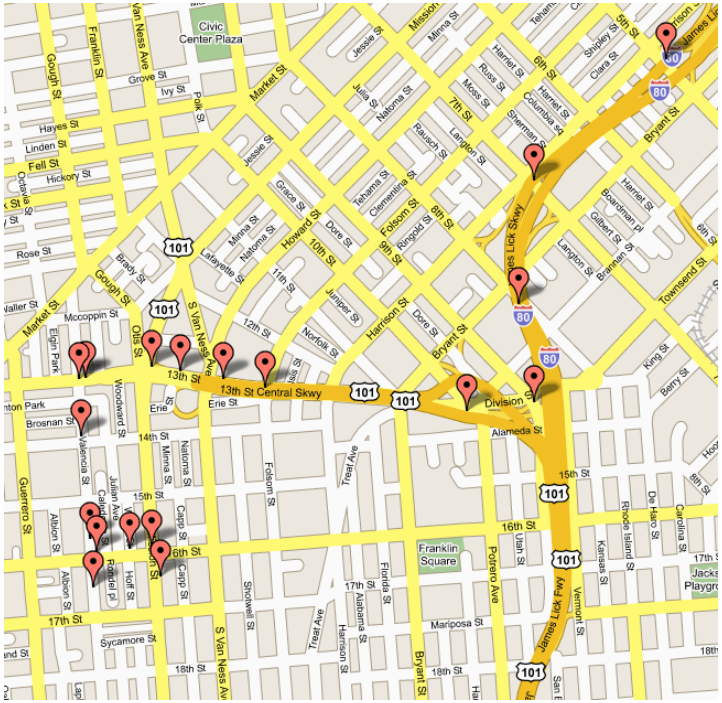
Single-Source Shortest Path Problem - The problem of finding shortest paths from a source vertex v to all other vertices in the graph.



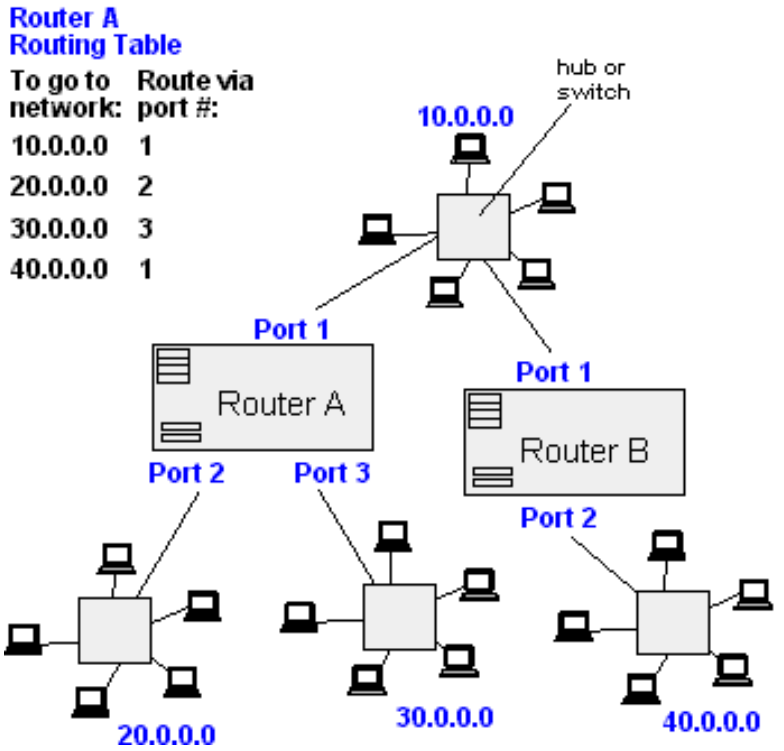
جامعة شط العرب كلية
العلوم
قسم علوم الحاسبات

Applications

- Maps (Map Quest, Google Maps)
- Routing Systems



From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.



Dijkstra's algorithm

Dijkstra's algorithm - is a solution to the single-source shortest path problem in graph theory.

Works on both directed and undirected graphs. However, all edges must have nonnegative weights.

Input: Weighted graph G={E,V} and source vertex v∈V, such that all edge weights are nonnegative

Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex v∈V to all other vertices

Approach

- The algorithm computes for each vertex u the distance to u from the start vertex v, that is, the weight of a shortest path between v and u.
- the algorithm keeps track of the set of vertices for which the distance has been computed, called the cloud C
- Every vertex has a label D associated with it. For any vertex u, D[u] stores an approximation of the distance between v and u. The algorithm will update a D[u] value when it finds a shorter path from v to u.
- When a vertex u is added to the cloud, its label D[u] is equal to the actual (final) distance between the starting vertex v and

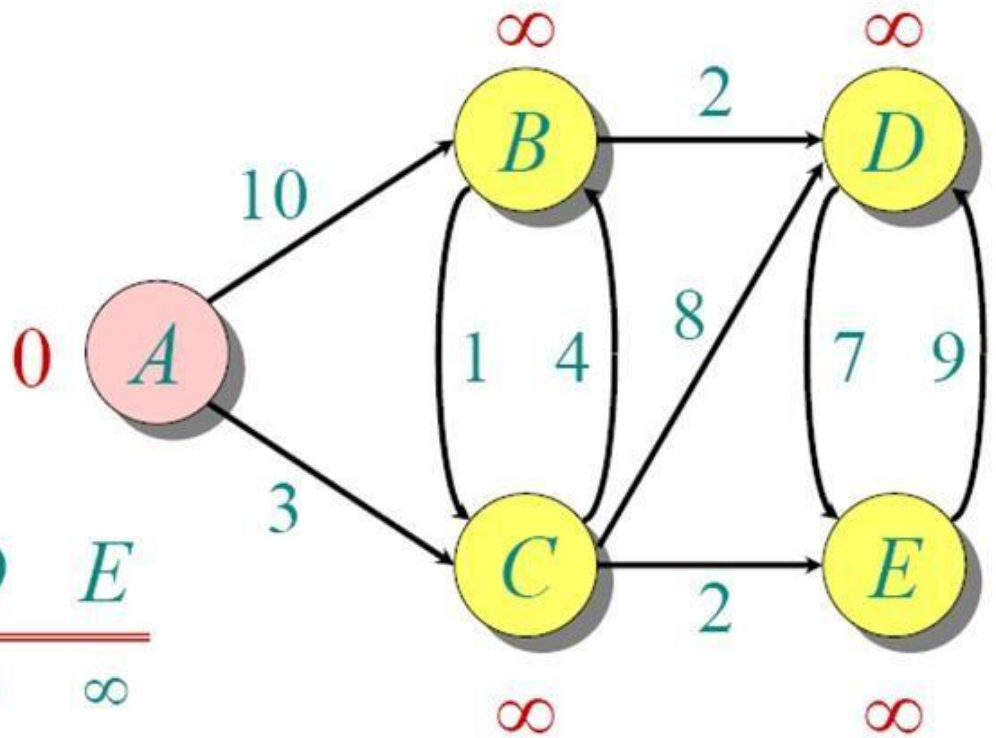
vertex u.

Dijkstra pseudocode

```
Dijkstra(v1, v2):
   for each vertex v:                    // Initialization
      v's distance := infinity.
      v's previous := none.
   v1's distance := 0.
   List := {all vertices}.

   while List is not empty:
      v := remove List vertex with minimum distance.
      mark v as known.
      for each unknown neighbor n of v:
         dist := v's distance + edge (v, n)'s weight.

         if dist is smaller than n's distance:
            n's distance := dist.
            n's previous := v.
```

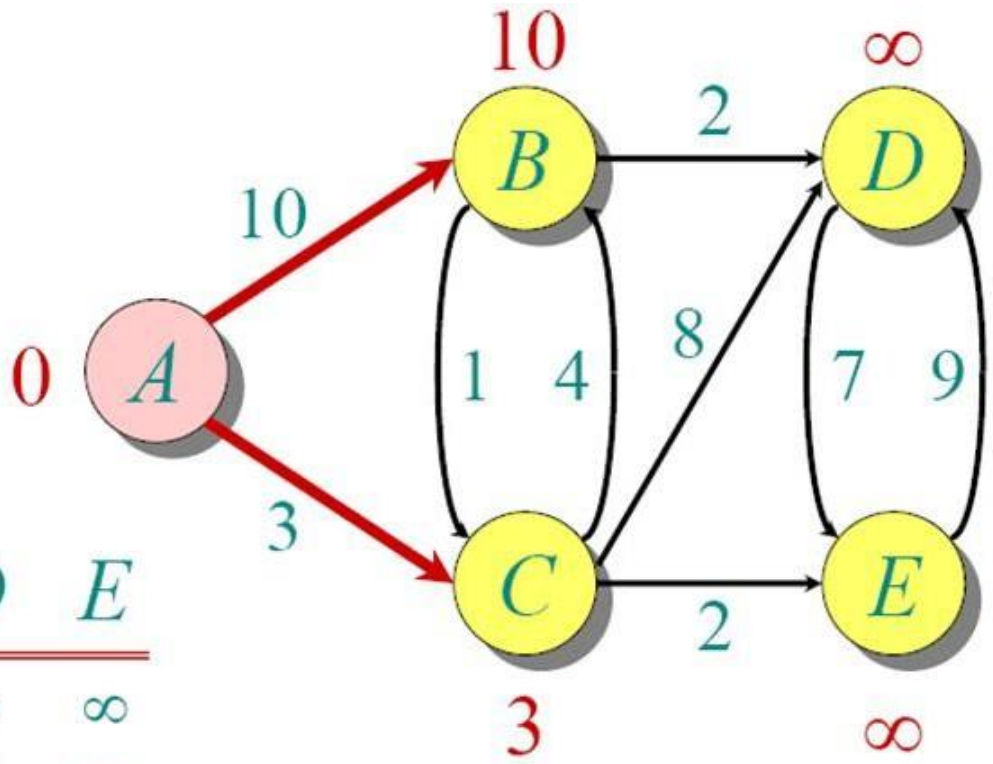
reconstruct path from v2 back to v1, following previous pointers.

Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

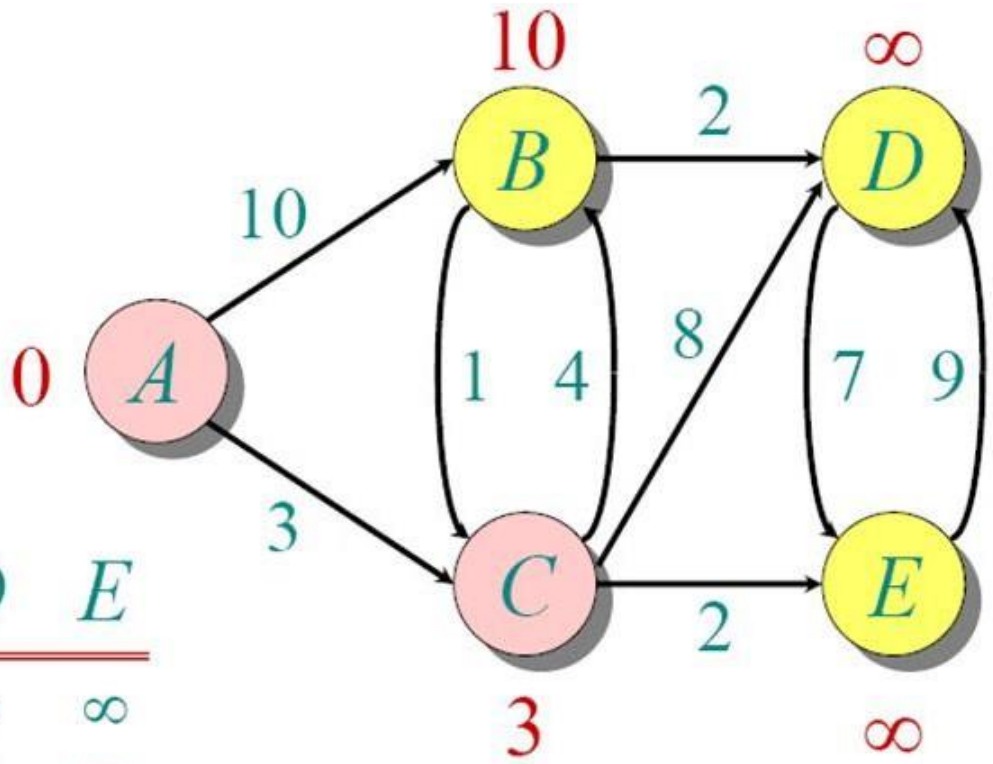
Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

$S: \{ A \}$

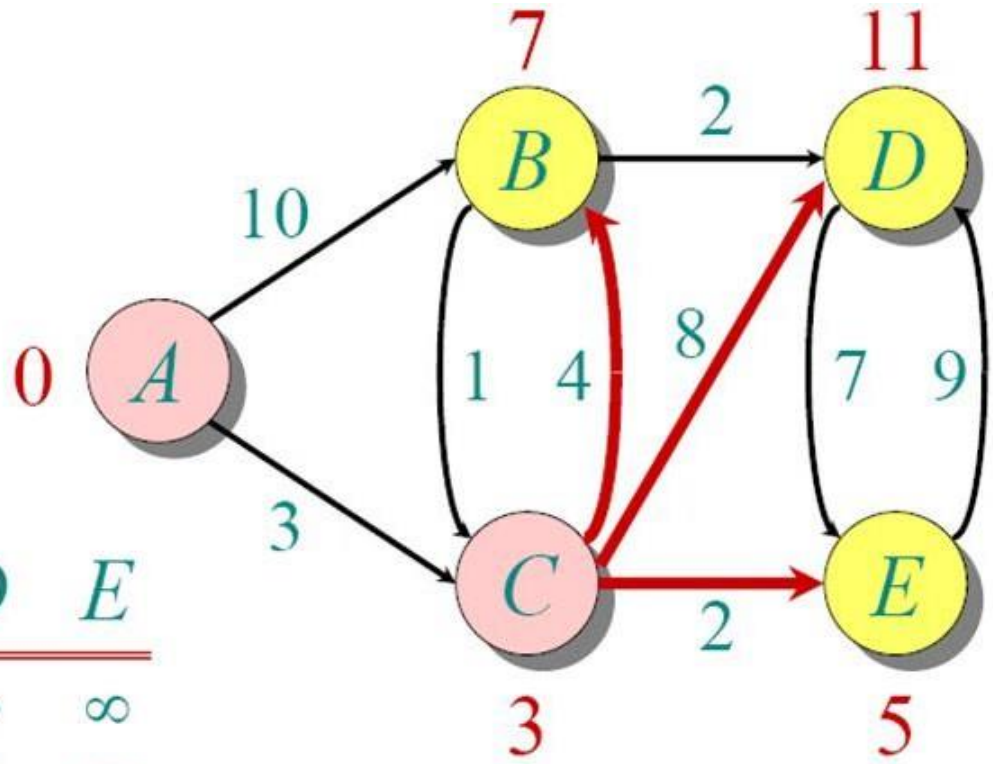
Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

$S: \{ A, C \}$

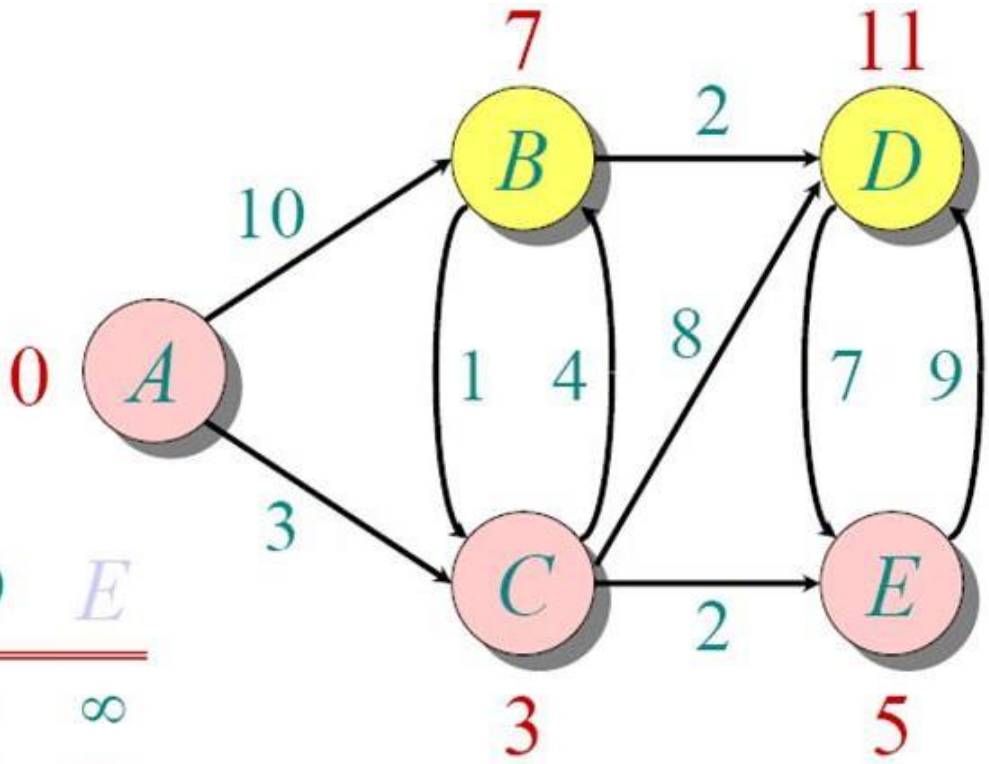
Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

$S: \{ A, C \}$

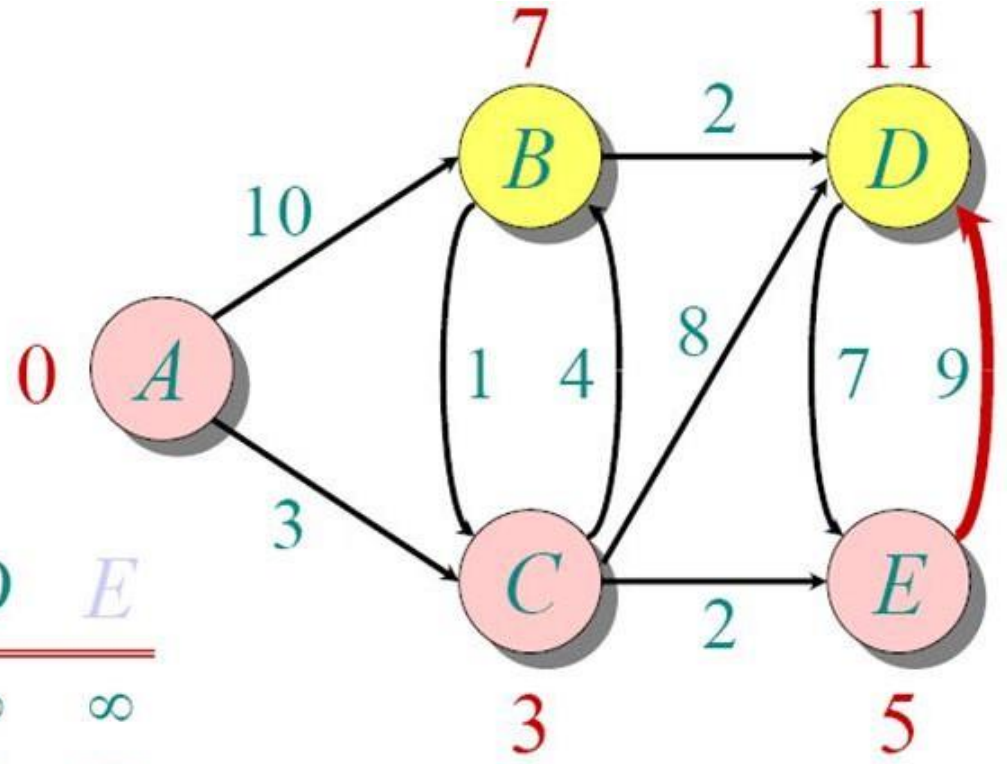
Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C, E }

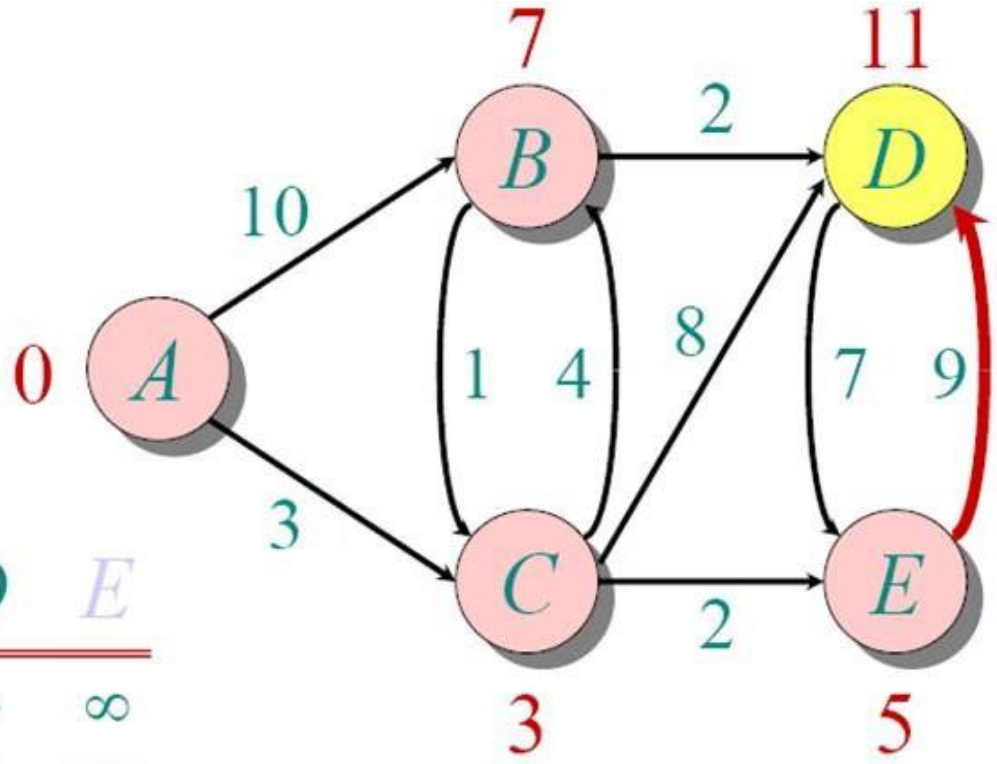
Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

$S: \{ A, C, E \}$

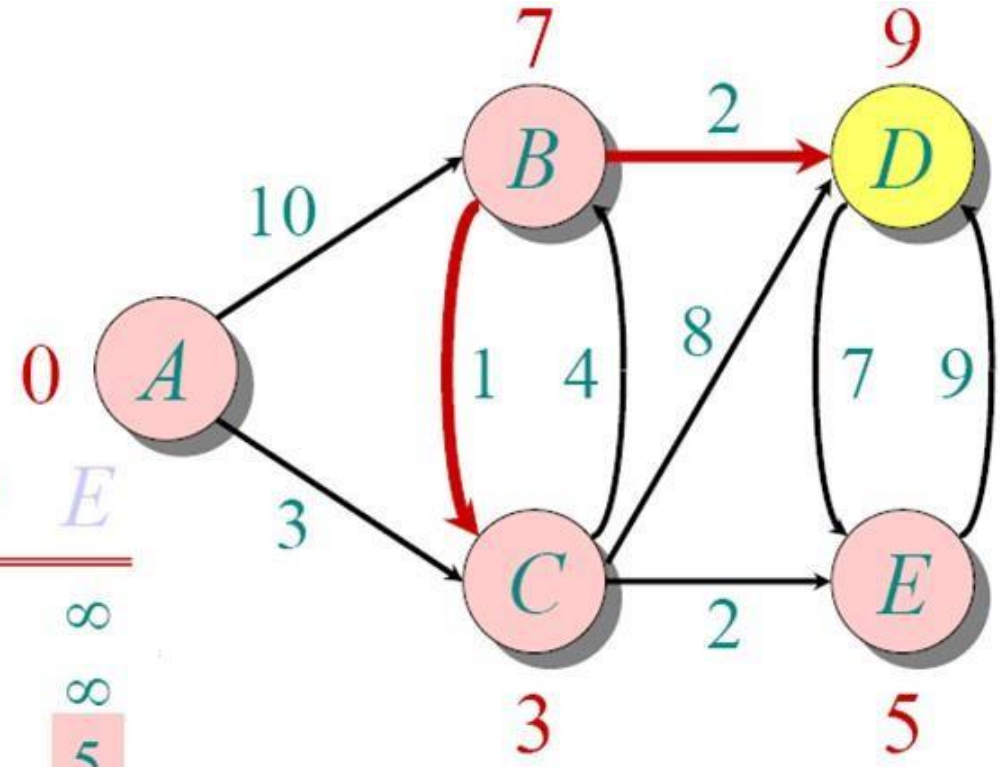
Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

$S: \{ A, C, E, B \}$

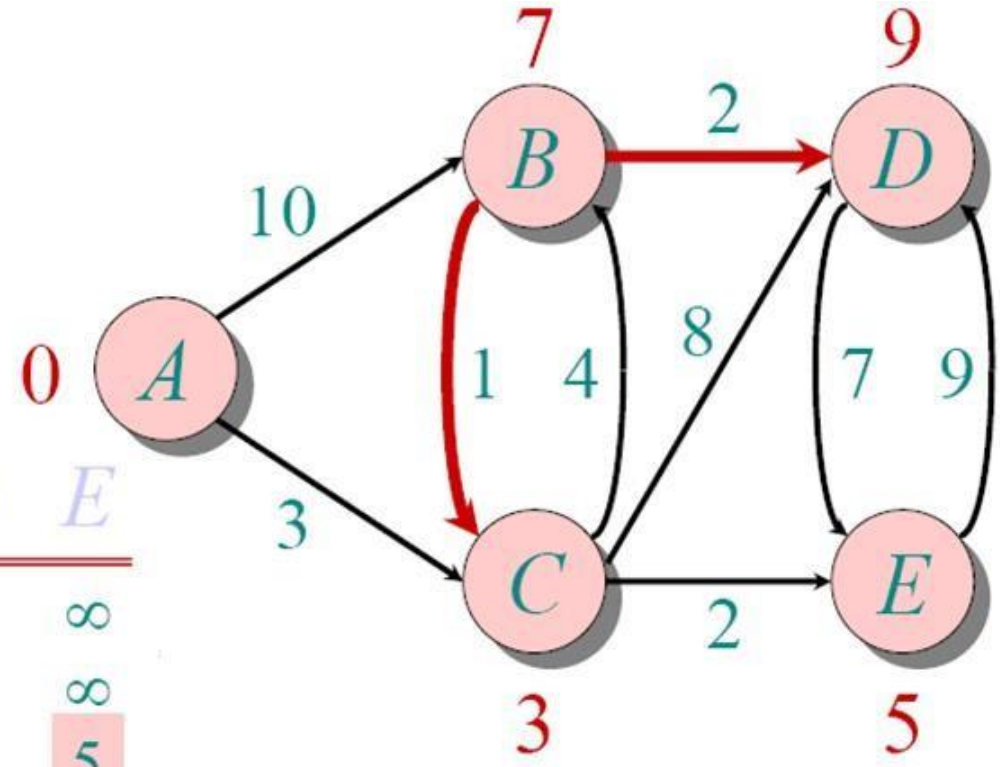
Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

$S: \{ A, C, E, B \}$

Another Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

$S: \{ A, C, E, B, D \}$

Dijkstra's Pseudo Code

- Graph *G*, weight function *w*, root *s*

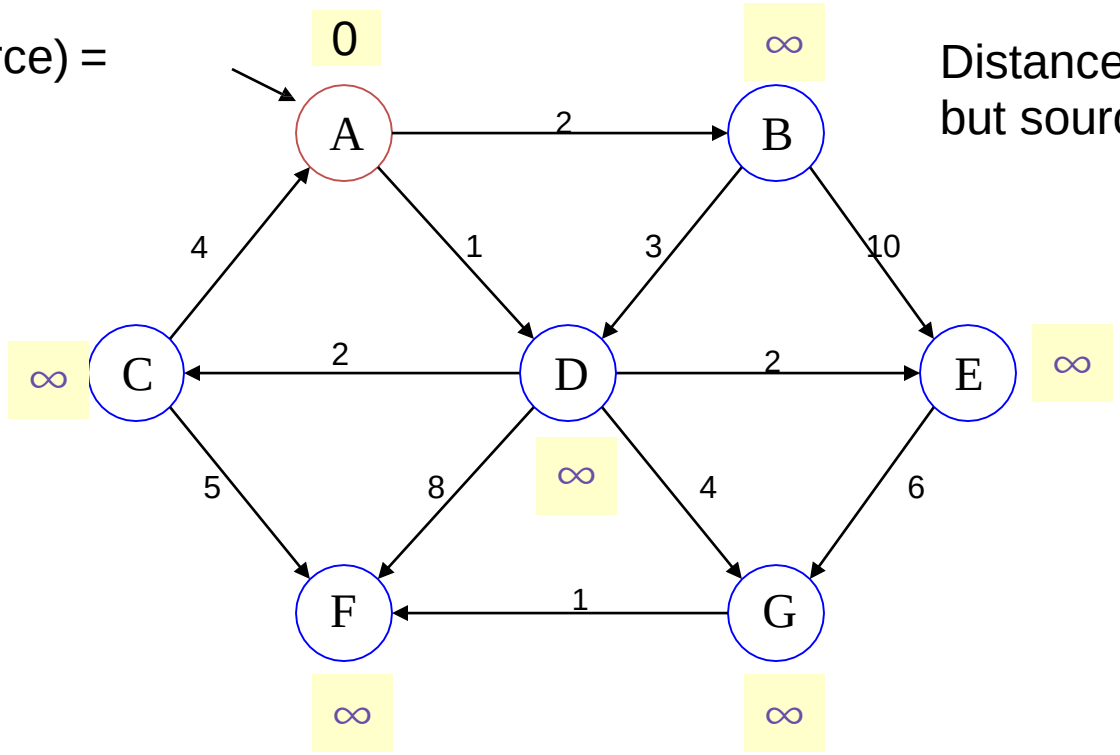
```
DIJKSTRA(G, w, s)
 1 for each v ∈ V
 2       do d[v] ← ∞
 3 d[s] ← 0
 4 S ← ∅   ▷ Set of discovered nodes
 5 Q ← V
 6 while Q ≠ ∅
 7       do u ← EXTRACT-MIN(Q)
 8          S ← S ∪ {u}
 9          for each v ∈ Adj[u]
10              do if d[v] > d[u] + w(u, v)
11                    then d[v] ← d[u] + w(u, v)
```

relaxing edges

Example: Initialization

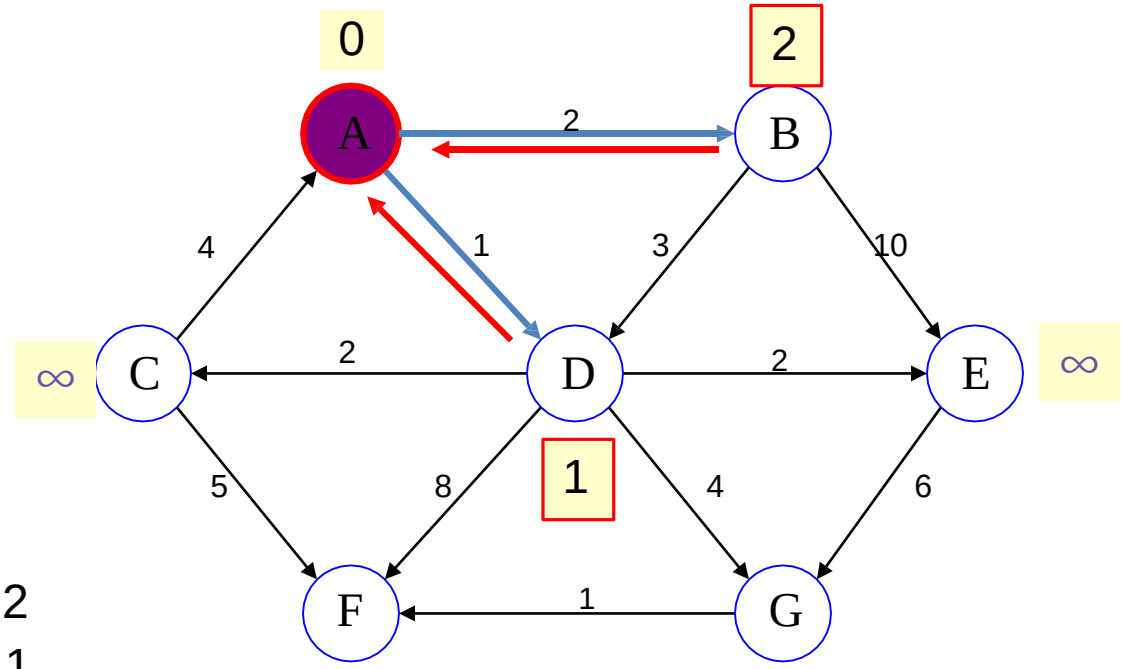
Distance(source) = 0

Distance (all vertices but source) = ∞



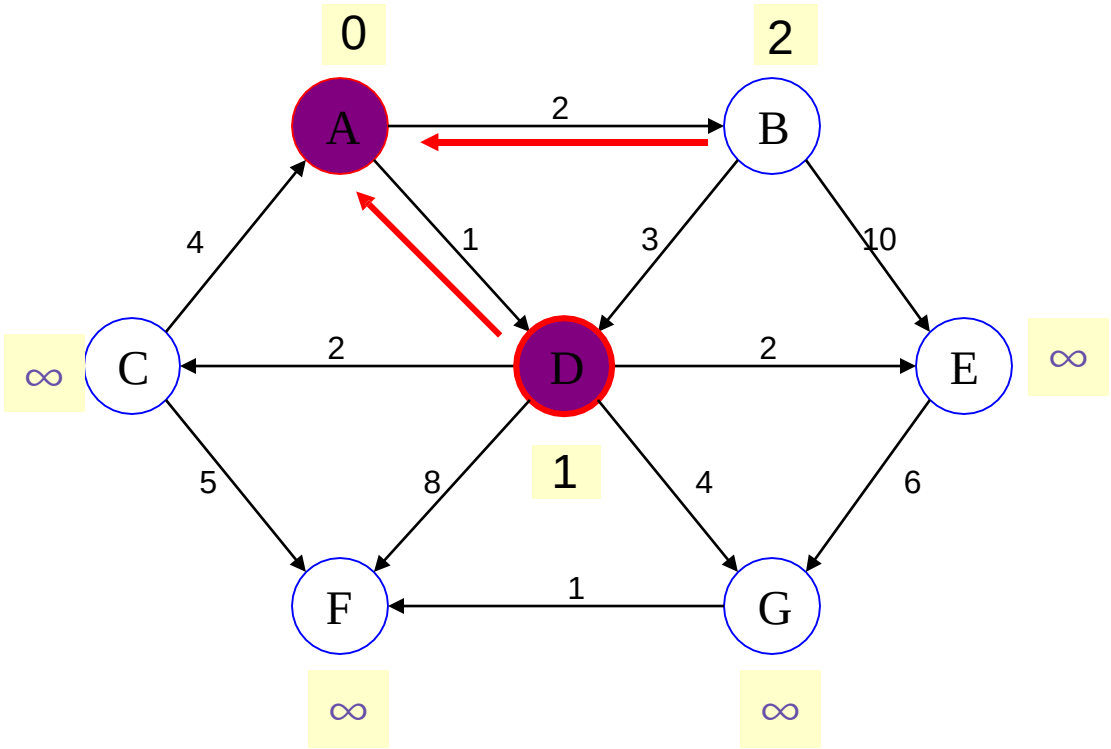
Pick vertex in List with minimum distance.

Example: Update neighbors' distance



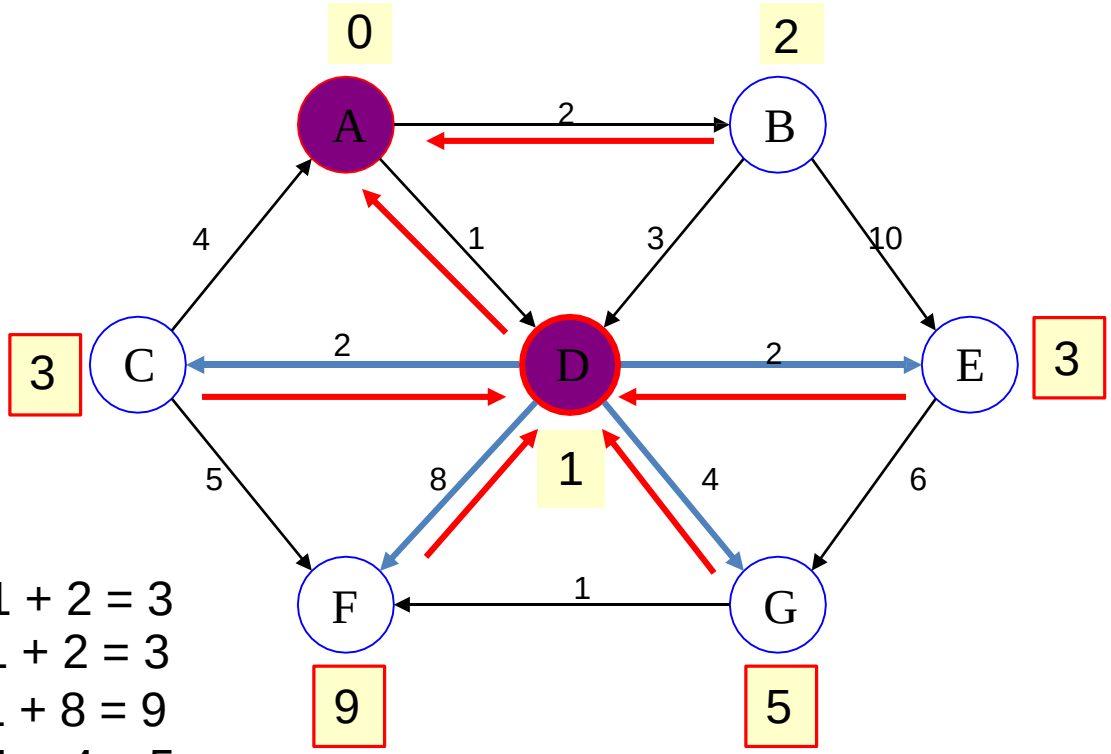
Distance(B) = 2
Distance(D) = 1

Example: Remove vertex with minimum distance



Pick vertex in List with minimum distance, i.e., D

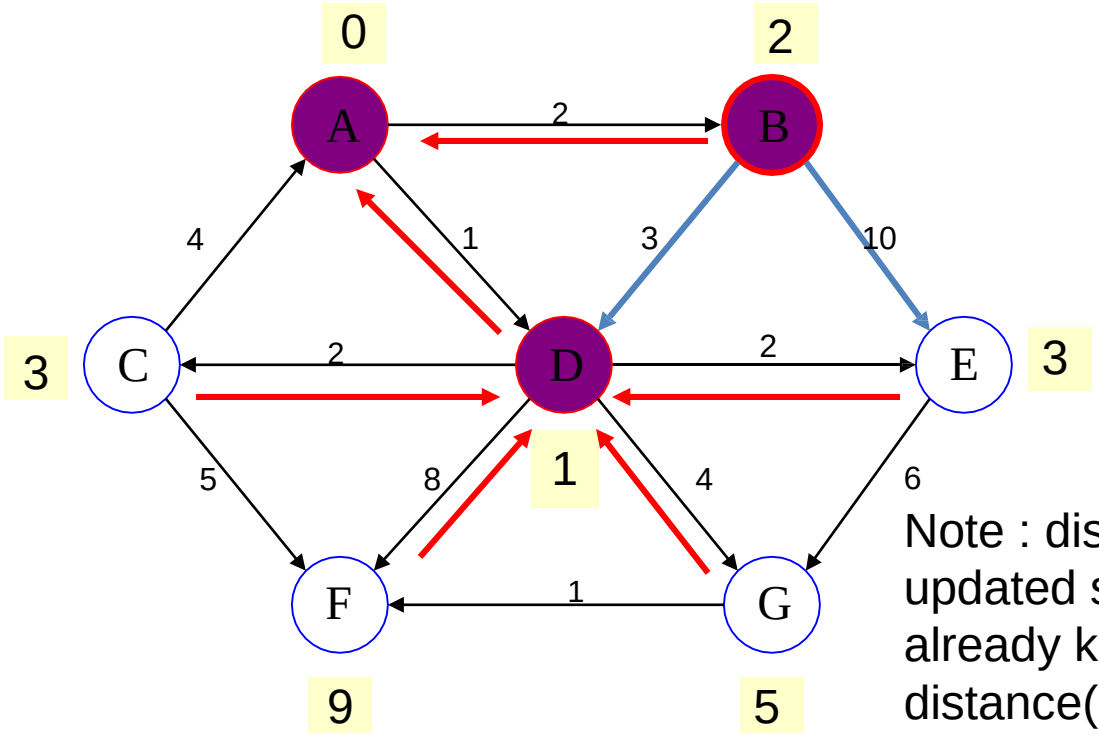
Example: Update neighbors



Distance(C) = 1 + 2 = 3
Distance(E) = 1 + 2 = 3
Distance(F) = 1 + 8 = 9
Distance(G) = 1 + 4 = 5

Example: Continued...

Pick vertex in List with minimum distance (B) and update neighbors

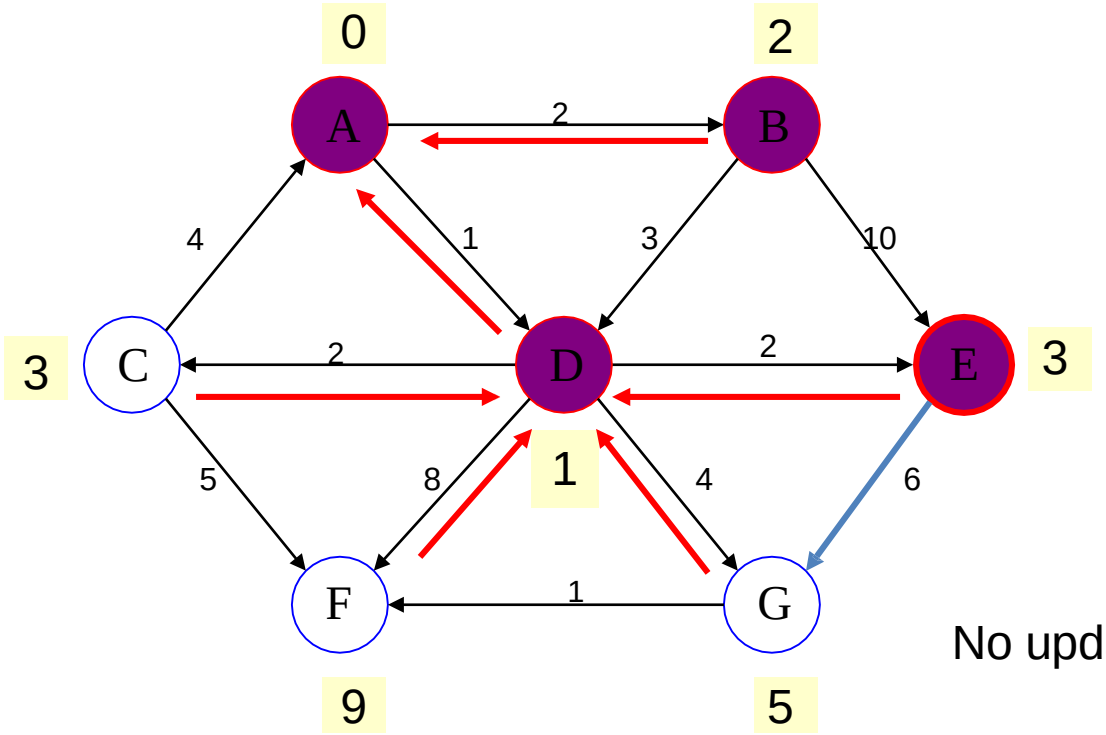


Note : distance(D) not updated since D is already known and distance(E) not updated since it is larger than

previously computed

Example: Continued...

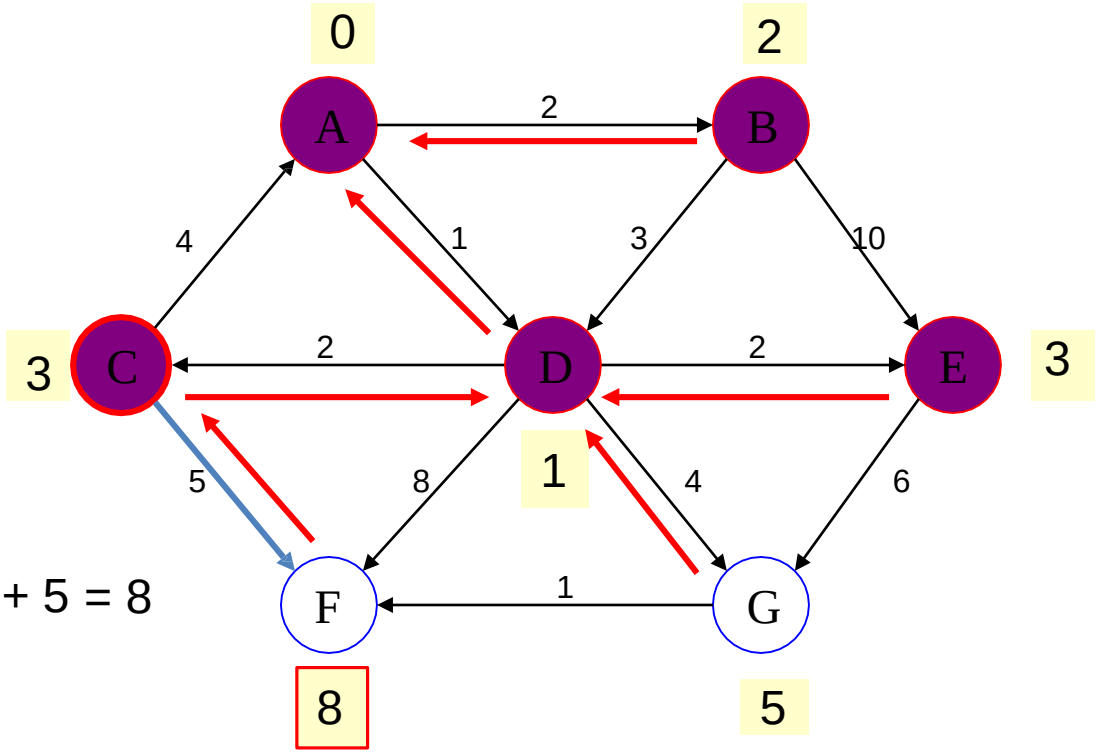
Pick vertex List with minimum distance (E) and update neighbors



No updating

Example: Continued...

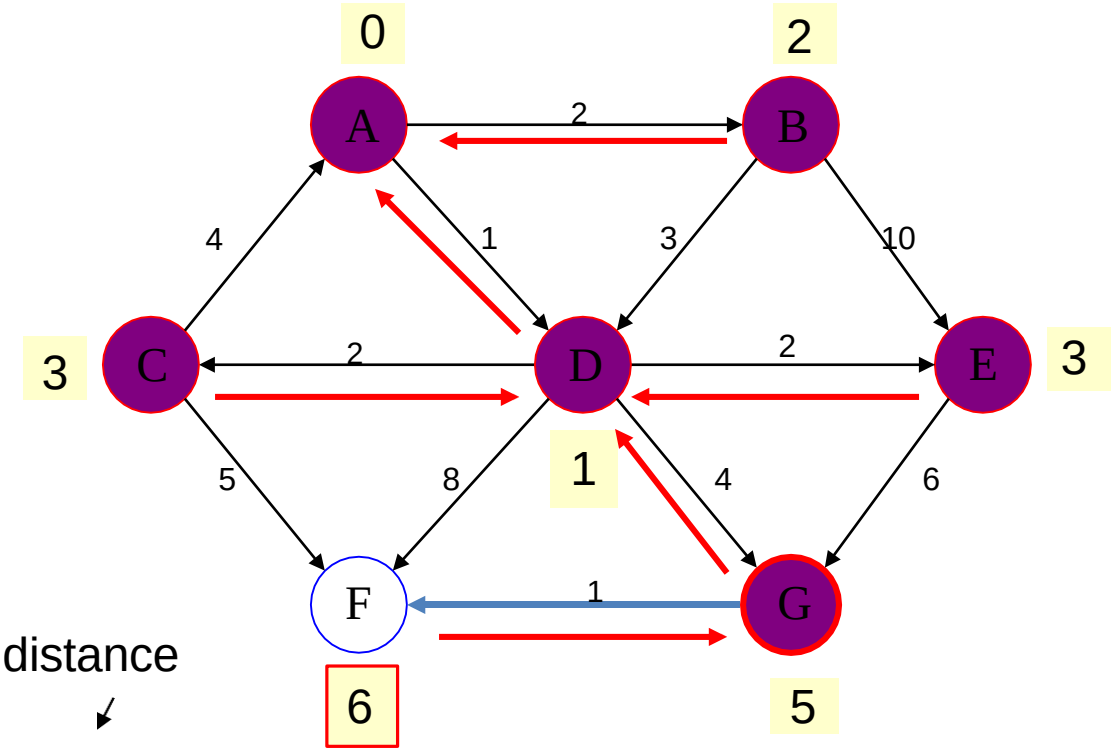
Pick vertex List with minimum distance (C) and update neighbors



Distance(F) = 3 + 5 = 8

Example: Continued...

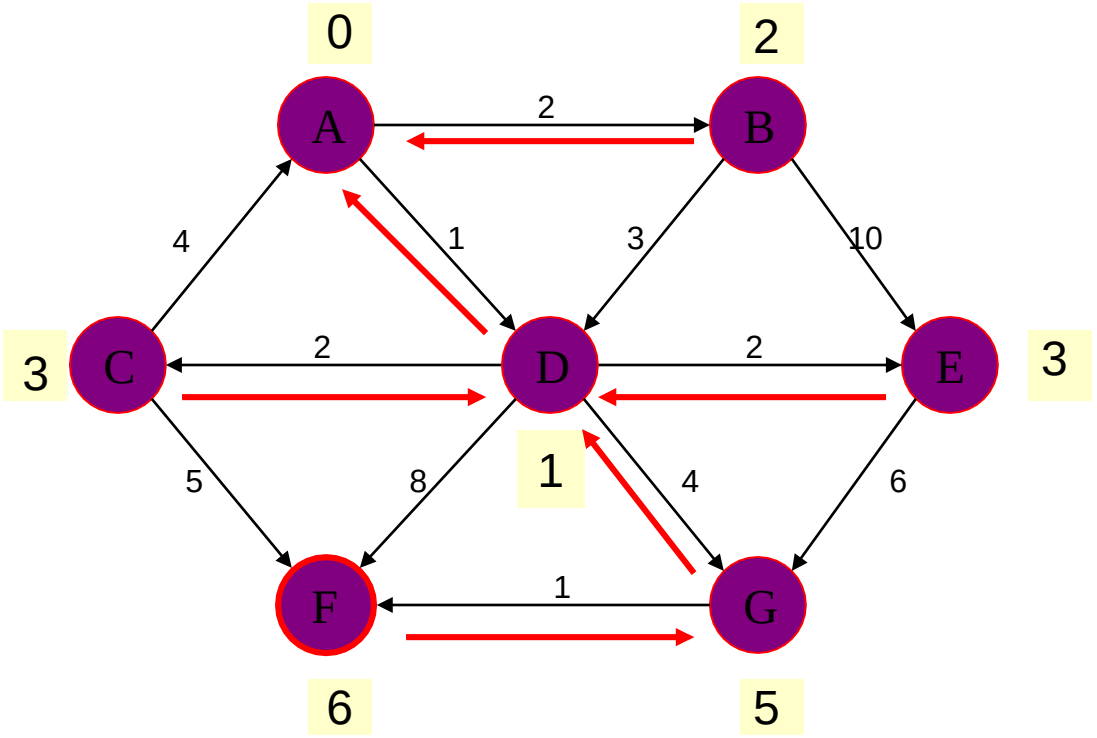
Pick vertex List with minimum distance (G) and update neighbors



Previous distance

Distance(F) = min (8, 5+1) = 6

Example (end)



Pick vertex not in S with lowest cost (F) and update neighbors

Time Complexity: Using List

The simplest implementation of the Dijkstra's algorithm stores vertices in an ordinary linked list or array

- Good for dense graphs (many edges)

- |V| vertices and |E| edges
- Initialization O(|V|)
- While loop O(|V|)
  - Find and remove min distance vertices O(|V|)
- Potentially |E| updates
    - Update costs O(1)

Total time $O(|V^2| + |E|) = O(|V^2|)$

Time Complexity: Priority Queue

For sparse graphs, (i.e. graphs with much less than $|V^2|$ edges)
Dijkstra's implemented more efficiently by *priority queue*

- Initialization O(|V|) using O(|V|) buildHeap
- While loop O(|V|)
  - Find and remove min distance vertices O(log |V|) using O(log |V|) deleteMin

- Potentially |E| updates
  - Update costs O(log |V|) using decreaseKey

Total time O(|V|log|V| + |E|log|V|) = O(|E|log|V|)

- |V| = O(|E|) assuming a connected graph

Heap Data Structure بنية البيانات الكومة

A heap is a tree-based data structure that allows access to the minimum and maximum element in the tree in constant time. The constant time taken is O(1)). This is regardless of the data stored in the heap.

Heap is a special case of balanced binary tree data structure where the root-node key is compared with its children and arranged accordingly. If **α** has child node **β** then :

الكومة عبارة عن بنية بيانات قائمة على الشجرة تسمح بالوصول إلى الحد الأدنى والحد الأقصى للعنصر في الشجرة في وقت ثابت. الوقت الثابت المستغرق هو O(1)). هذا بغض النظر عن البيانات المخزنة في الكومة.

الكومة هي حالة خاصة من بنية بيانات الشجرة الثنائية المتوازنة حيث تتم مقارنة مفتاح العقدة الجذرية مع أبنائه وترتيبهم وفقًا لذلك. إذا كان α لديه عقدة فرعية β، فعندئذٍ:

key(α) ≥ key(β)

A heap can be of two types: **Min-heap** and **Max-heap**.

- In a min-heap, the value of the parent (root) node is less than or equal to its children.
- In a max-heap, the value of the parent (root) node is greater than or equal to its children.

يمكن أن تكون الكومة من نوعين: Min-heap وMax-heap.

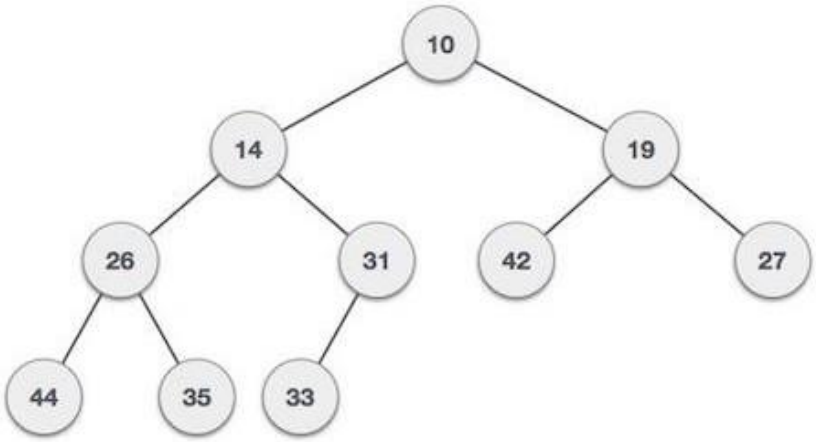
- في الكومة الصغيرة، تكون قيمة العقدة الأصلية (الجذر) أقل من أو تساوي أبنائها.
- في الكومة القصوى، تكون قيمة العقدة الأصلية (الجذر) أكبر من أو تساوي أبنائها.

Q: Consider the elements given below.

35, 33, 42, 10, 14, 19, 27, 44, 26, 31

1. Construct a min-heap (as a tree) from the above elements. Then show how the heap will be stored in the array. 1. أنشئ كومة صغيرة (على شكل شجرة) من العناصر المذكورة أعلاه. ثم أظهر كيف سيتم تخزين الكومة في المصفوفة.

Tree:

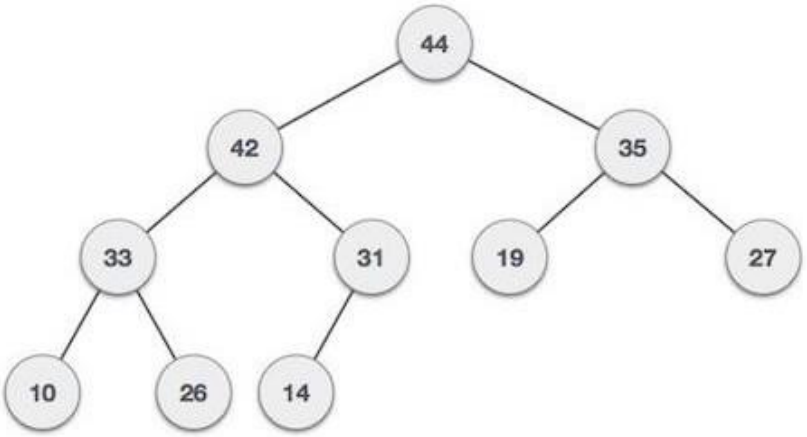


Array:

10	14	19	26	31	42	27	44	35	33

2. Construct a max-heap (as a tree) from the above elements. Then show how the heap will be stored in the array. 2. قم ببناء الحد الأقصى للكومة (على شكل شجرة) من العناصر المذكورة أعلاه. ثم أظهر كيف سيتم تخزين الكومة في المصفوفة.

Tree

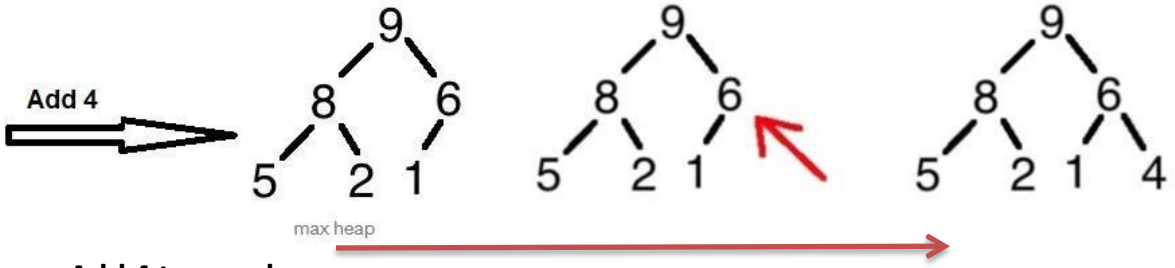


Array:

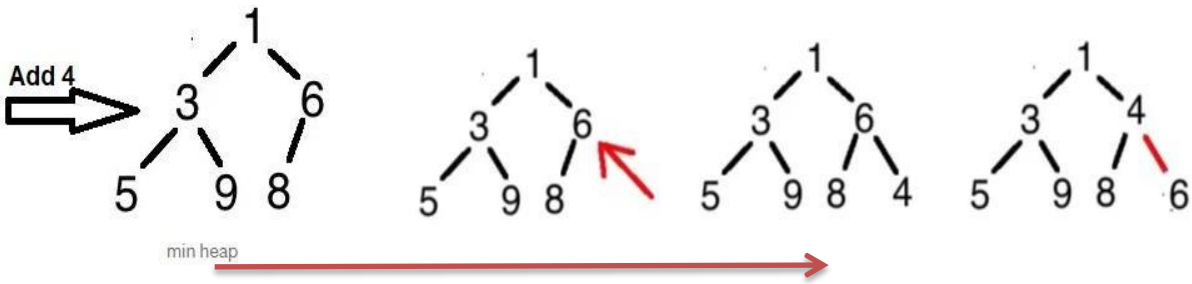
44	42	35	33	31	19	27	10	26	14

Add and remove element to heap

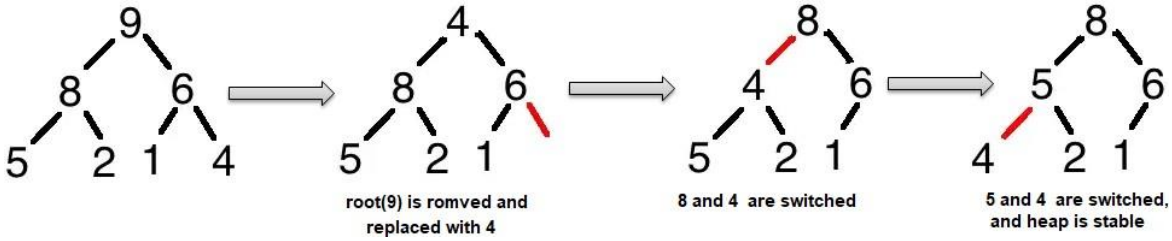
- **Add 4 to max heap**



- **Add 4 to max heap**



- **Remove from max heap**



- **Remove from min heap**