Algorithms

Lecture 3
Sorting with Divide and Conquer

Prantik Paul [PNP]
Lecturer
Department of Computer Science and Engineering
BRAC University

Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(n-i-1):
            if arr[j] > arr[j+1]:
                temp = arr[j]
                arr[j] = arr[j+1]
                arr[j+1] = temp
```

Worst Case Complexity?

Best Case Complexity?

Space Complexity?

Optimization?

Selection Sort

```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

Worst Case Complexity?
Best Case Complexity?
Space Complexity?
Optimization?

Insertion Sort

```
def insertion_sort(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr [j+1] = arr[j]
             j = j -1
        arr[j+1] = key
```

Worst Case Complexity?
Best Case Complexity?
Space Complexity?
Optimization?

Stable and Unstable Sort

[Stable and Unstable Sorting Algorithms]()

$(Dave, A)$	$(Alice, B)$	$(Carol, A)$	$(Dave, A)$
$(Alice, B)$	$(Carol, A)$	$(Dave, A)$	(Ken, A)
(Ken, A)	$(Dave, A)$	(Ken, A)	$(Carol, A)$
$(Eric, B)$	$(Eric, B)$	$(Eric, B)$	$(ALice, B)$
$(Carol, A)$	(Ken, A)	$(Alice, B)$	$(Eric, B)$

Divide and Conquer Strategy

Divide	the problem (instance) into subproblems.
Conquer	the subproblems by solving these recursively (or iteratively).
Combine	the solutions to the subproblems.

Merge Sort

The problem

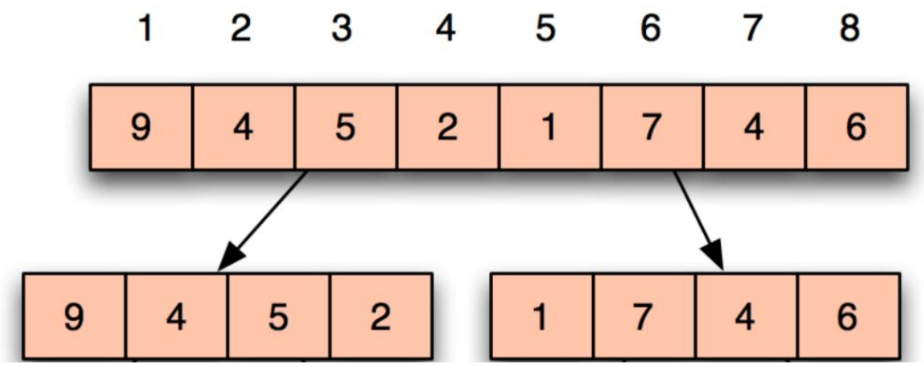
Sort an Array:

1. *Divide:* Trivial.
2. *Conquer:* Recursively sort 2 subarrays.
3. *Combine:* Merge the sorted subarrays in $\Theta(n)$ time.

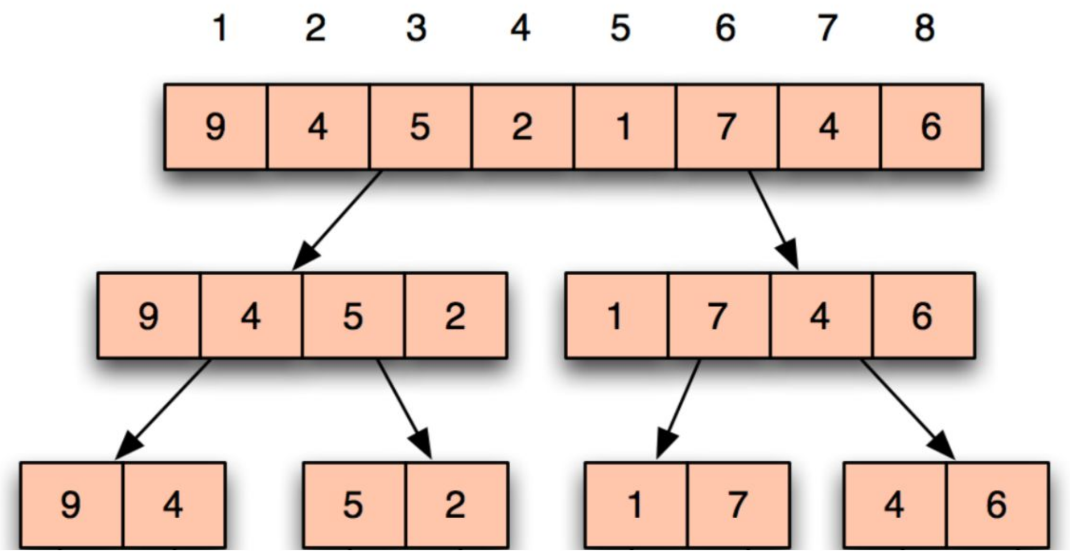
Merge Sort (Simulation)

1	2	3	4	5	6	7	8
9	4	5	2	1	7	4	6

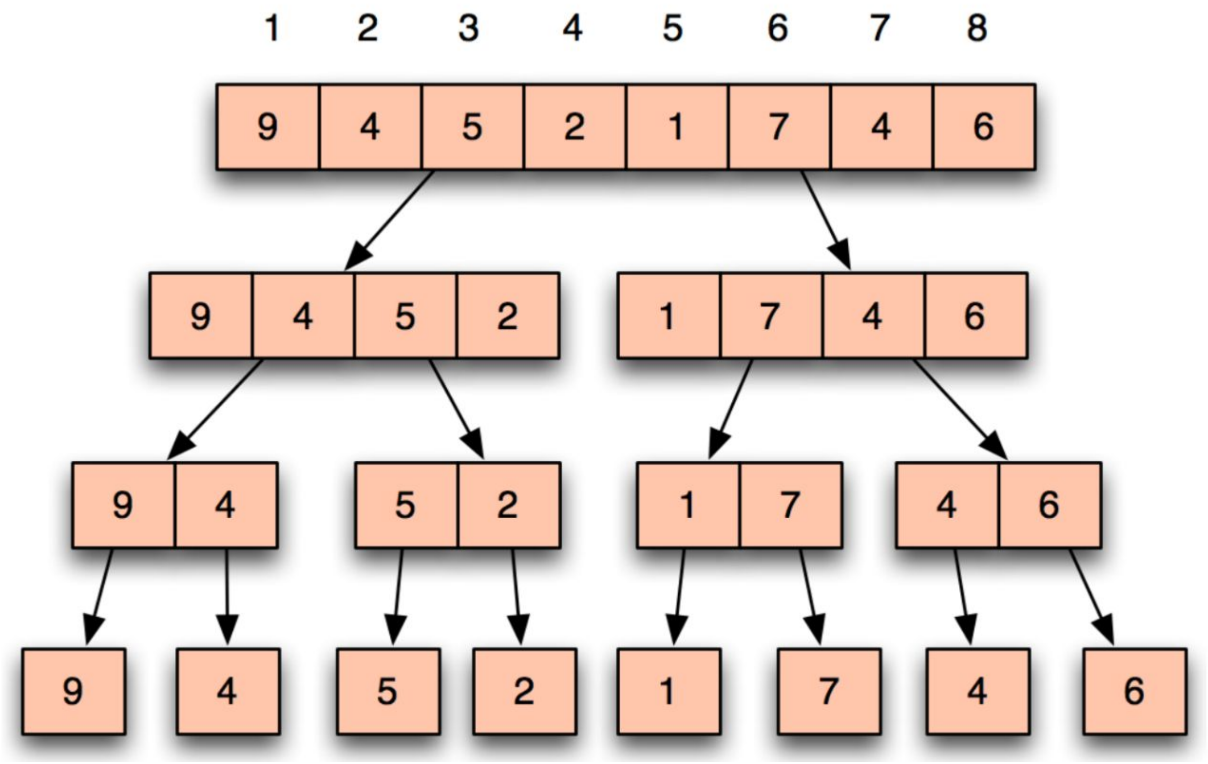
Merge Sort (Simulation)



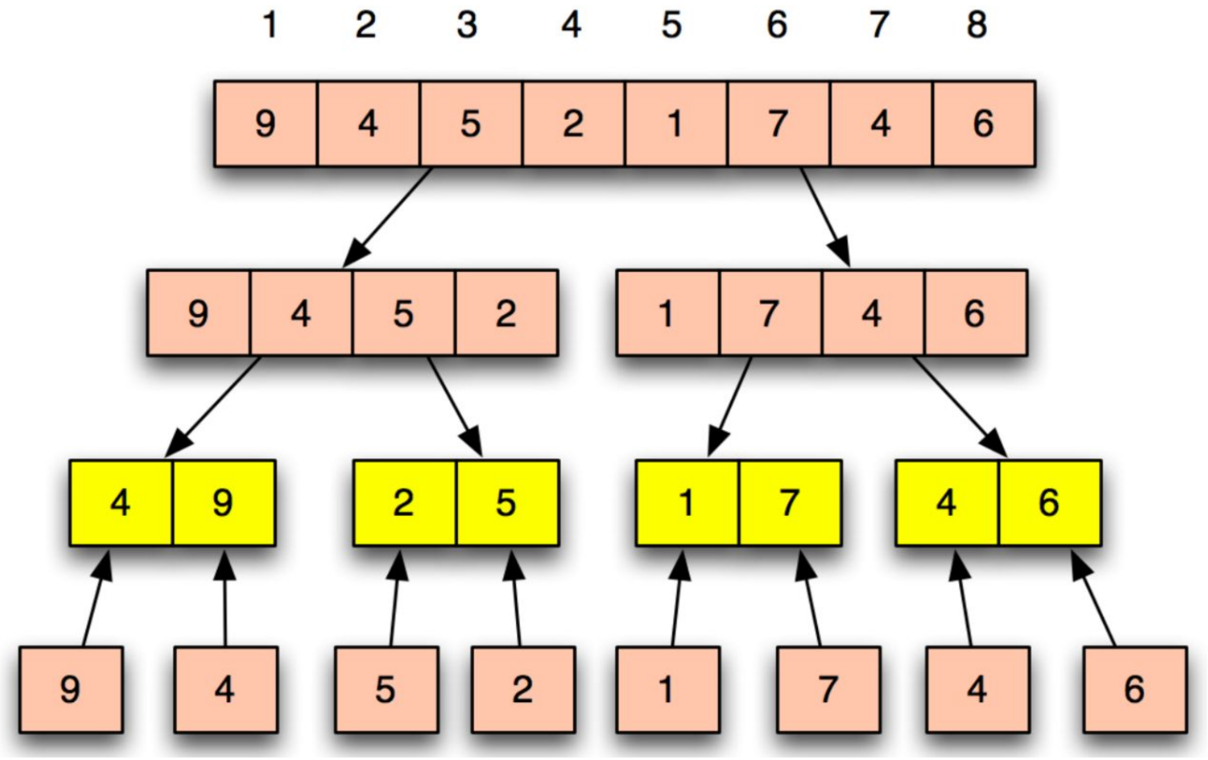
Merge Sort (Simulation)



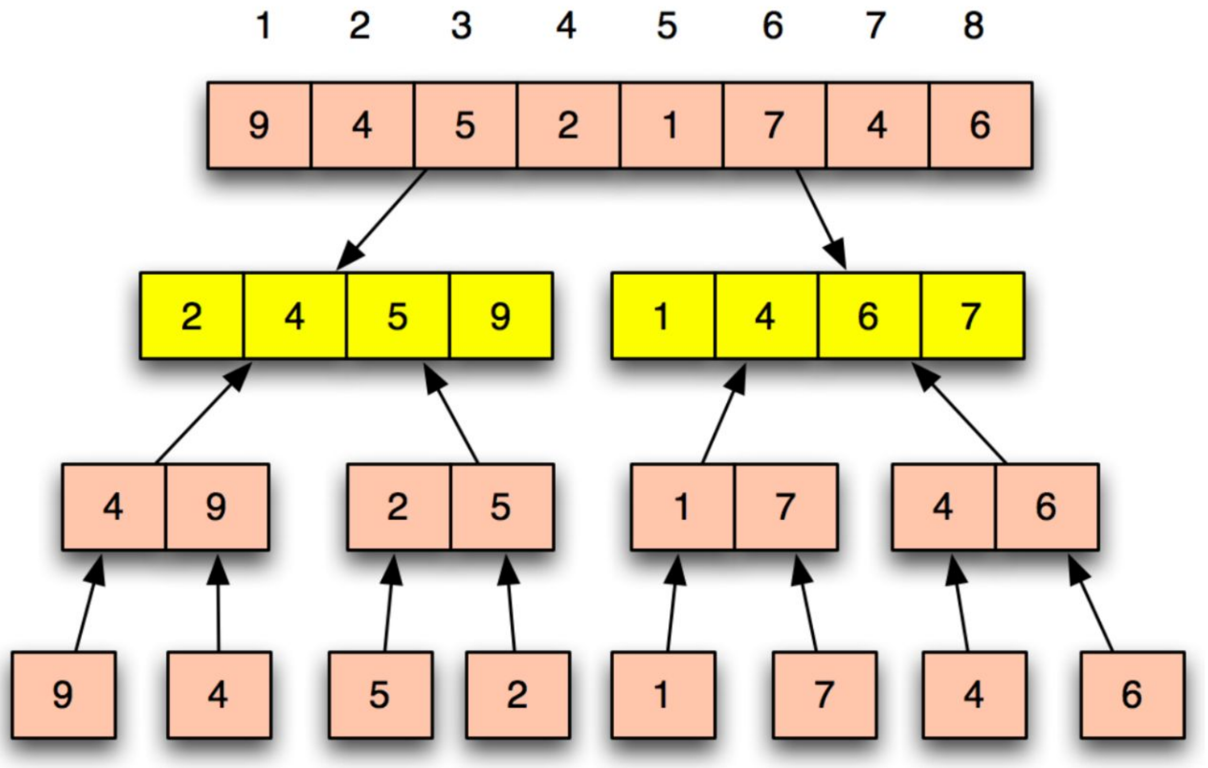
Merge Sort (Simulation)



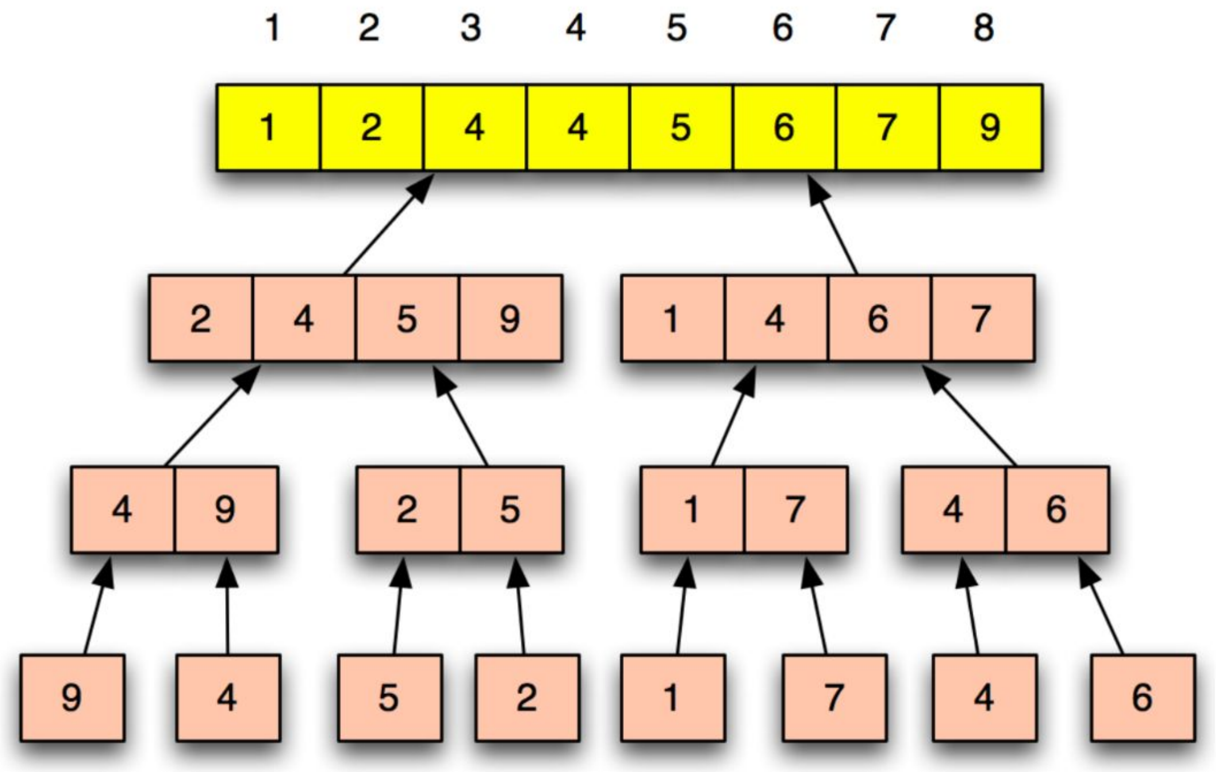
Merge Sort (Simulation)



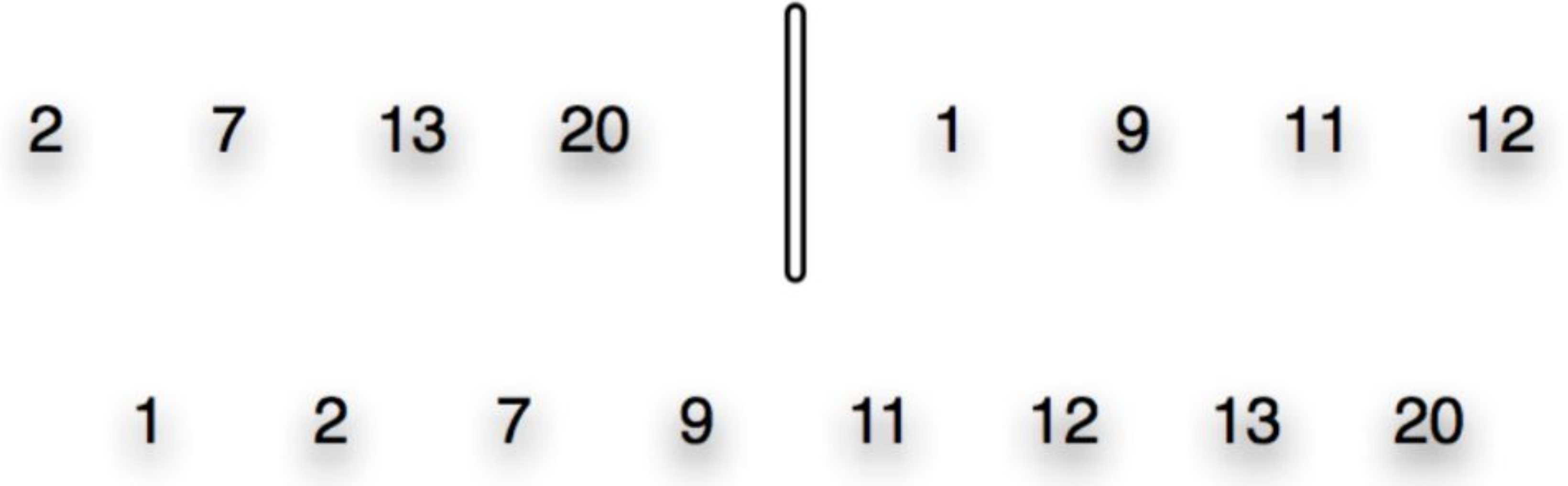
Merge Sort (Simulation)



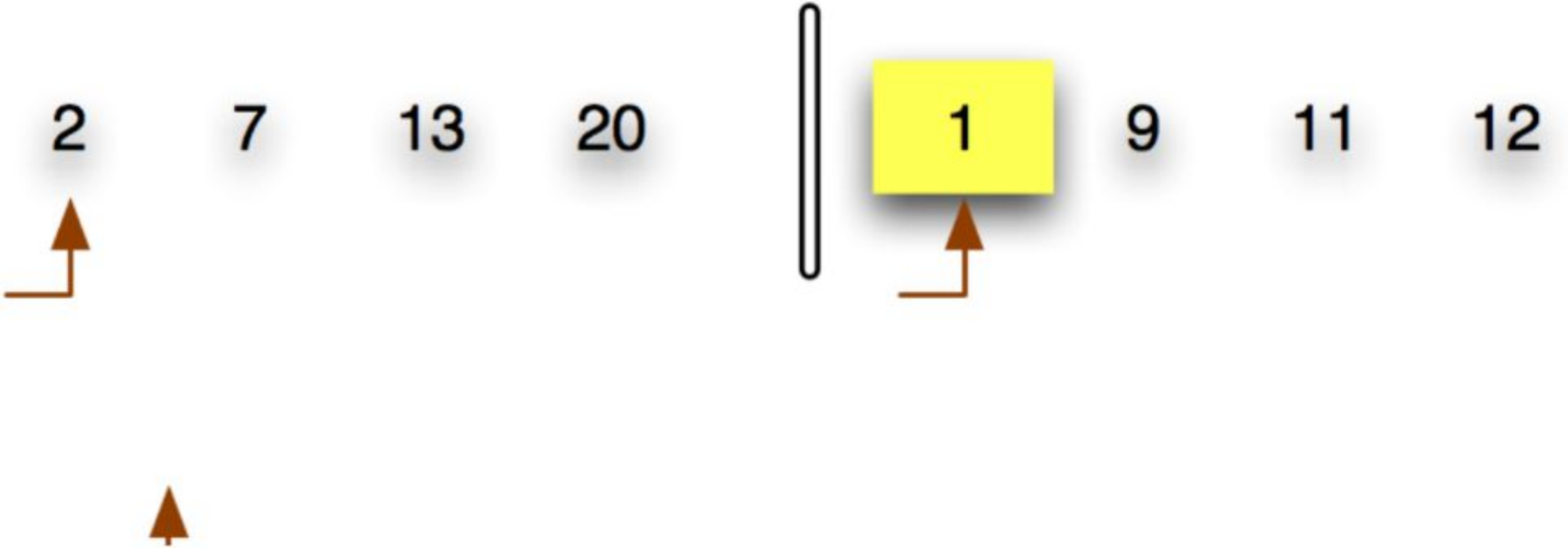
Merge Sort (Simulation)



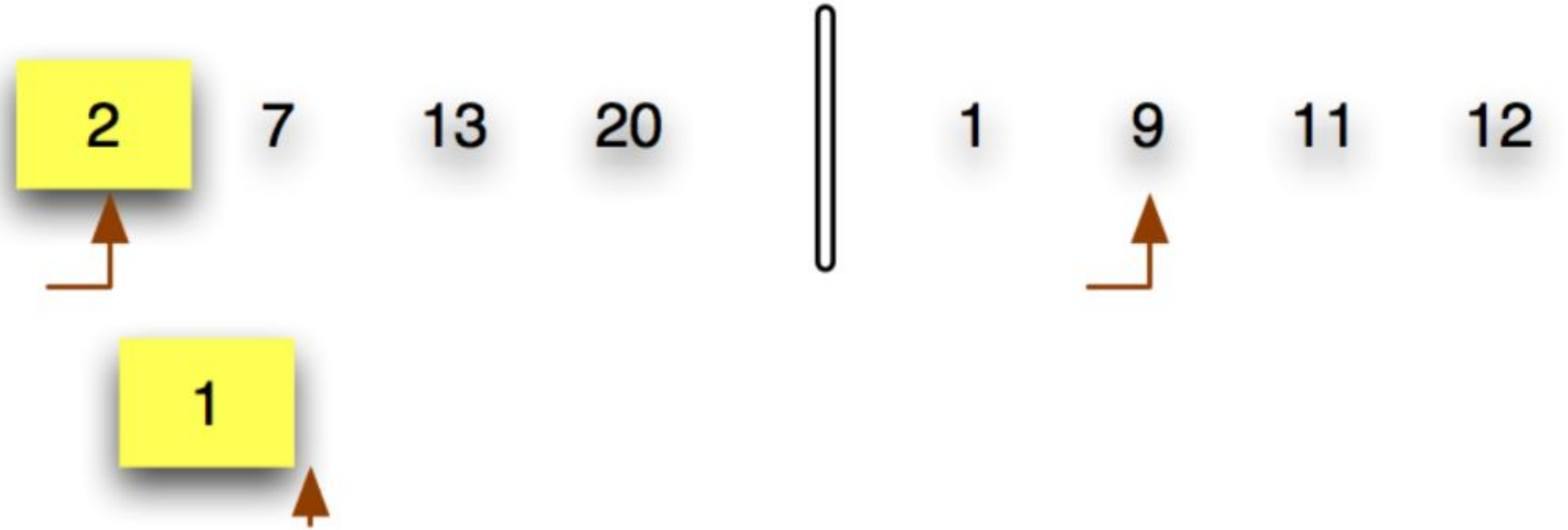
Merge (Simulation)



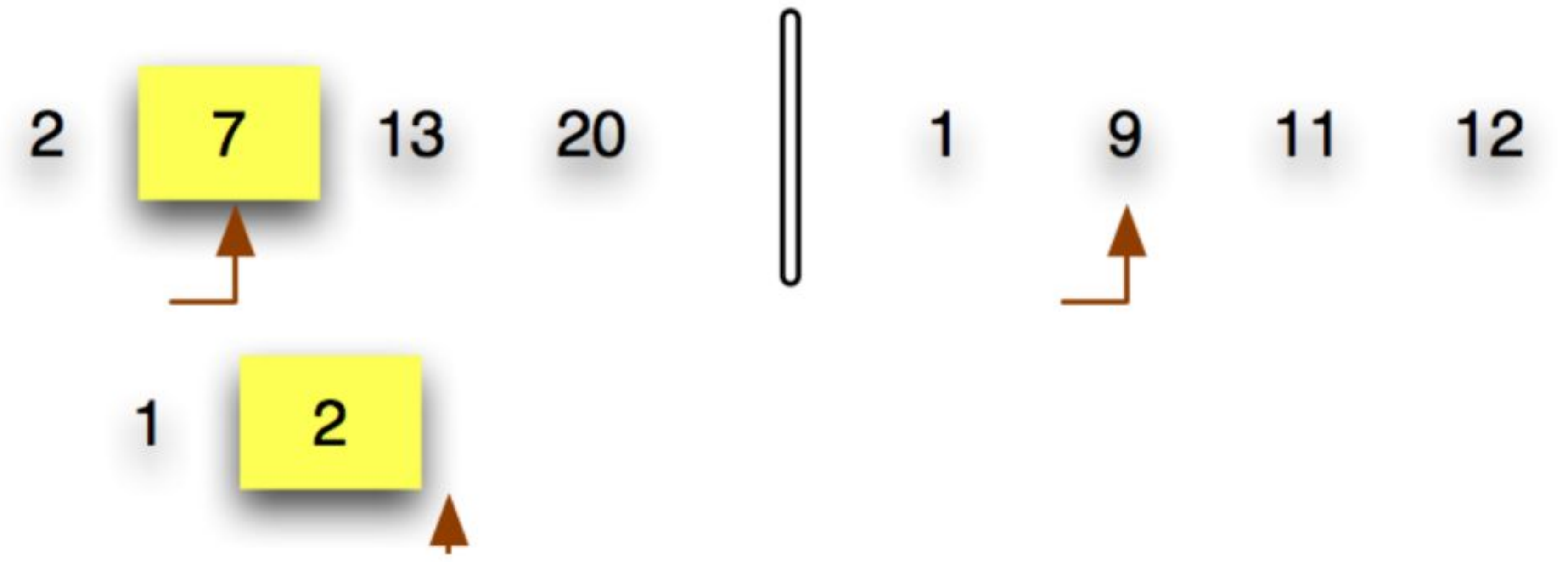
Merge (Simulation)



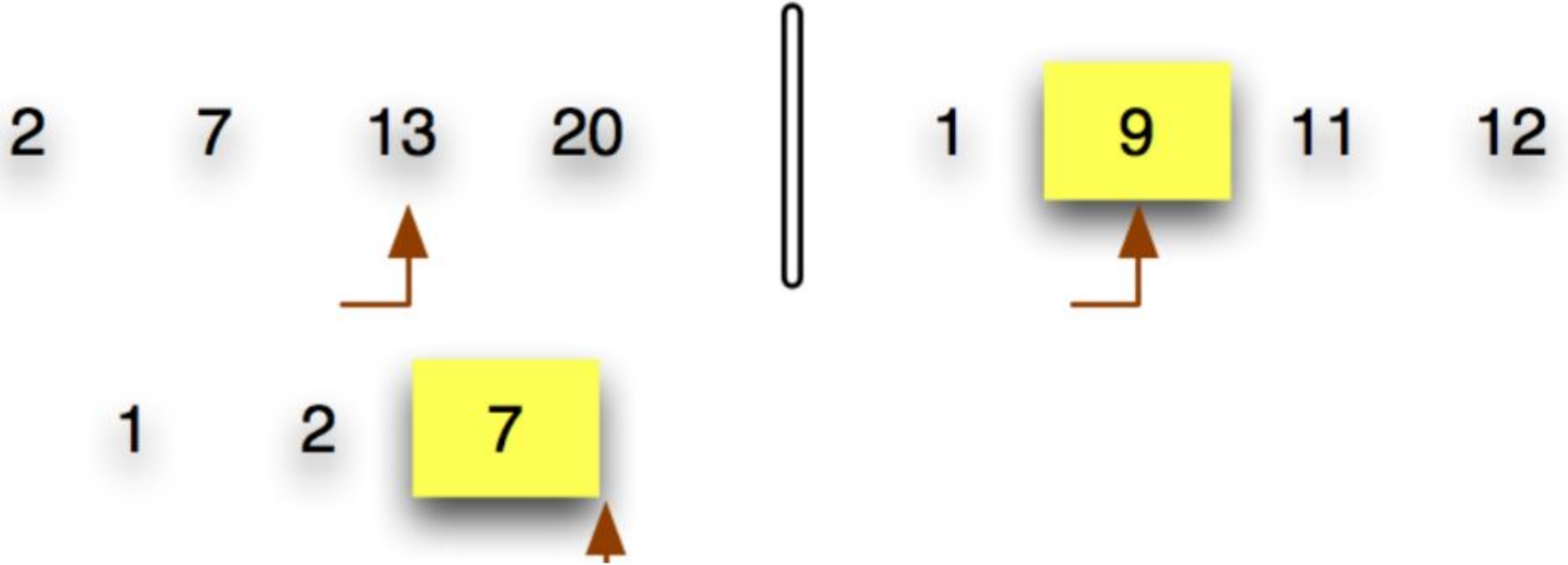
Merge (Simulation)



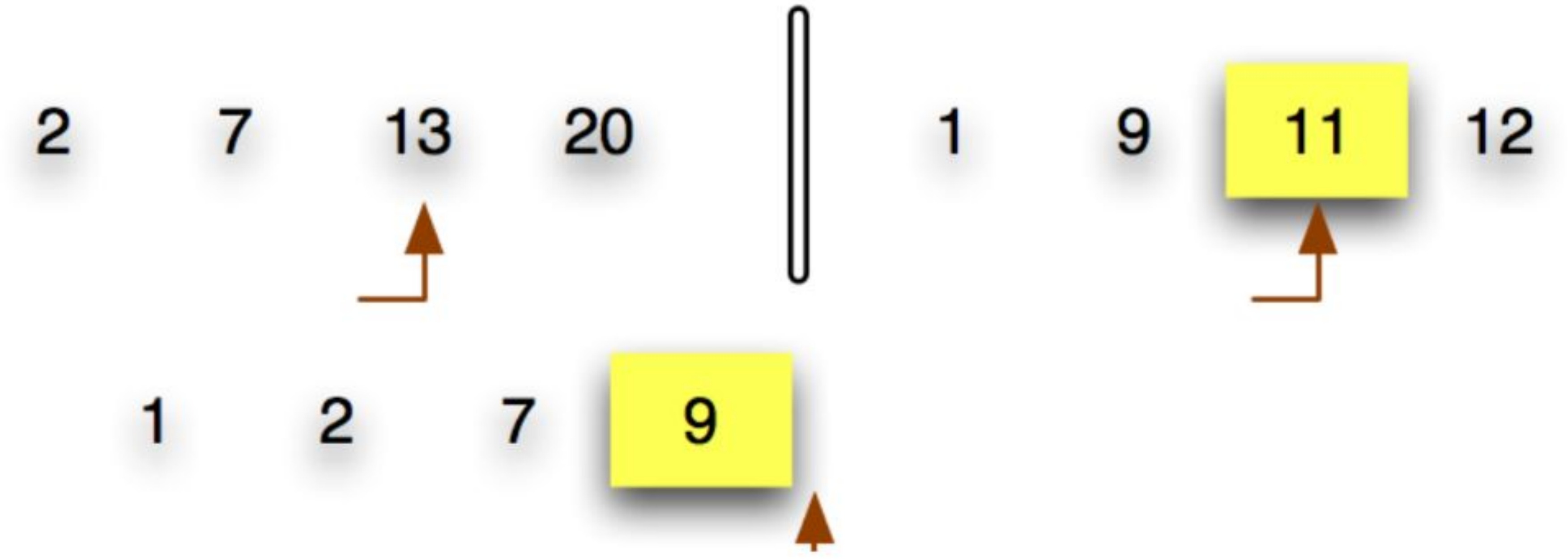
Merge (Simulation)



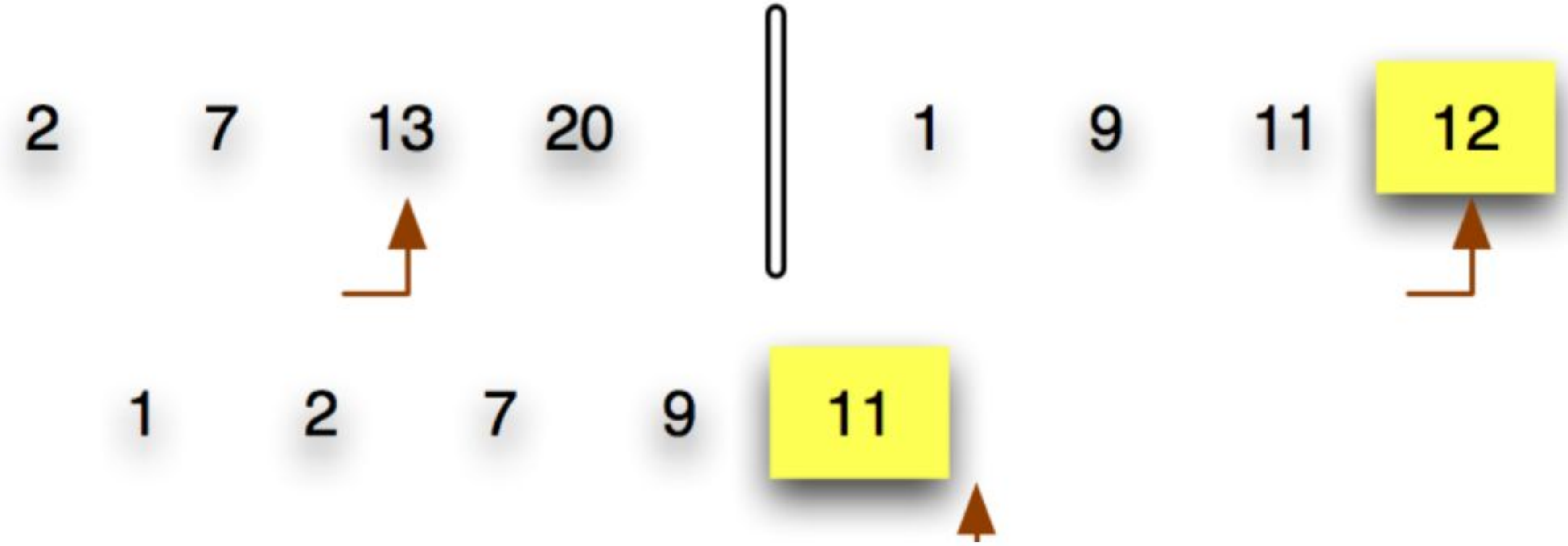
Merge (Simulation)



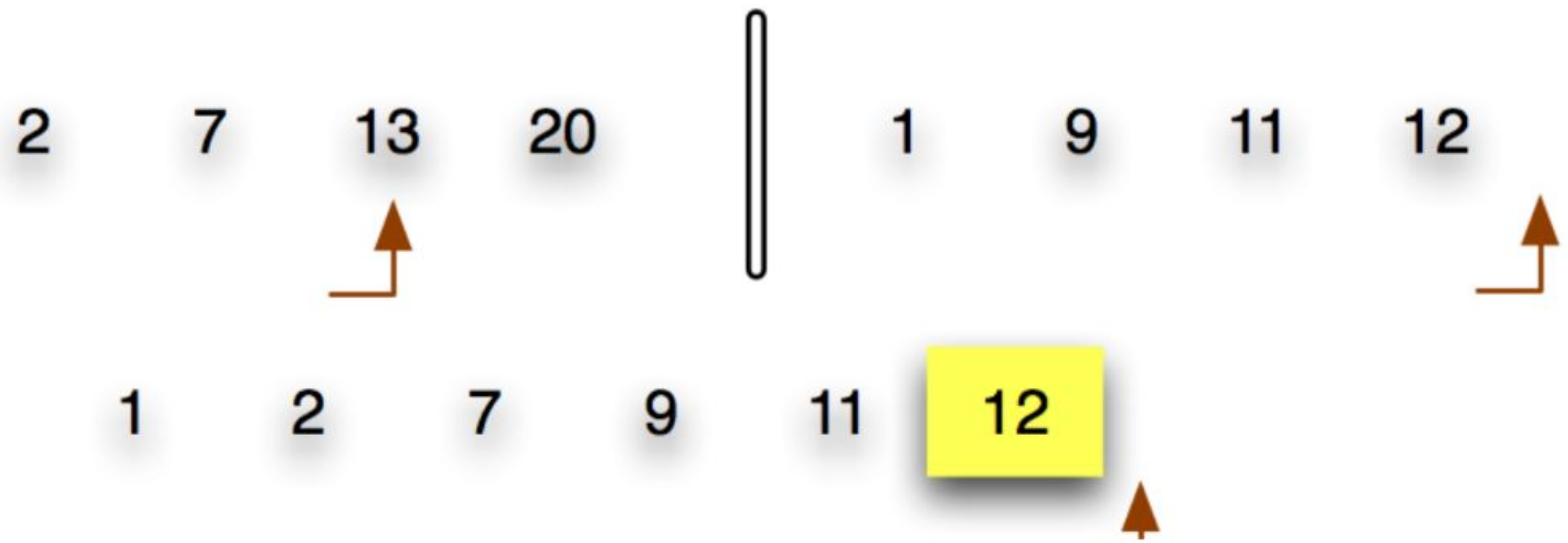
Merge (Simulation)



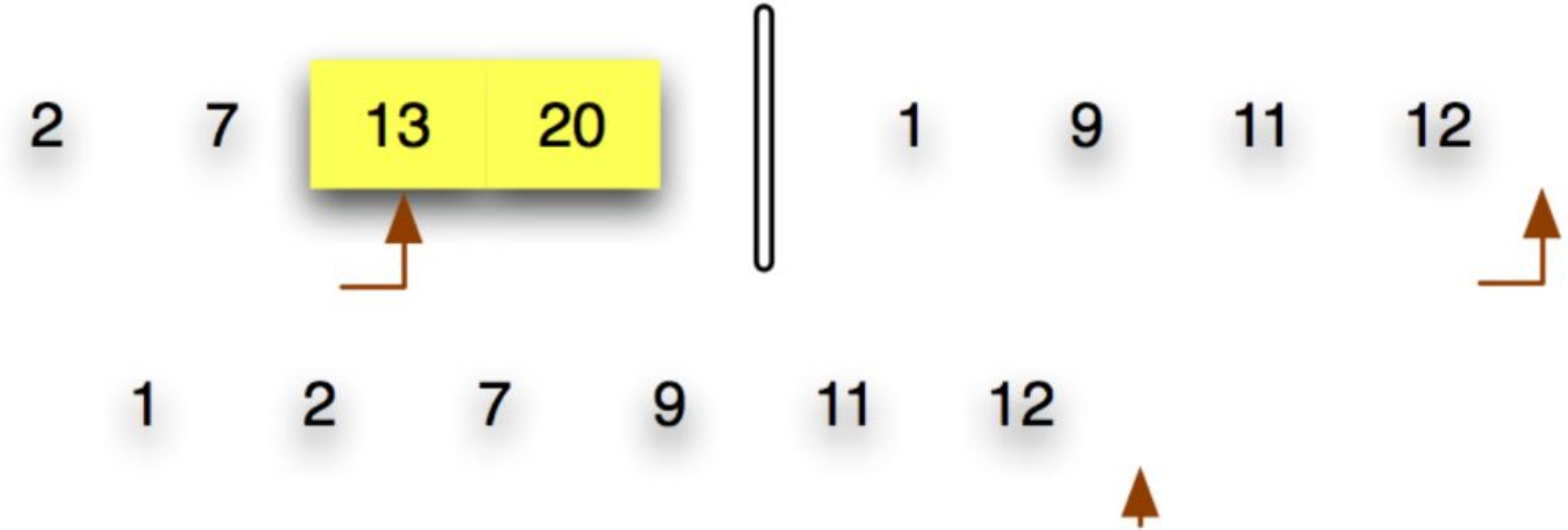
Merge (Simulation)



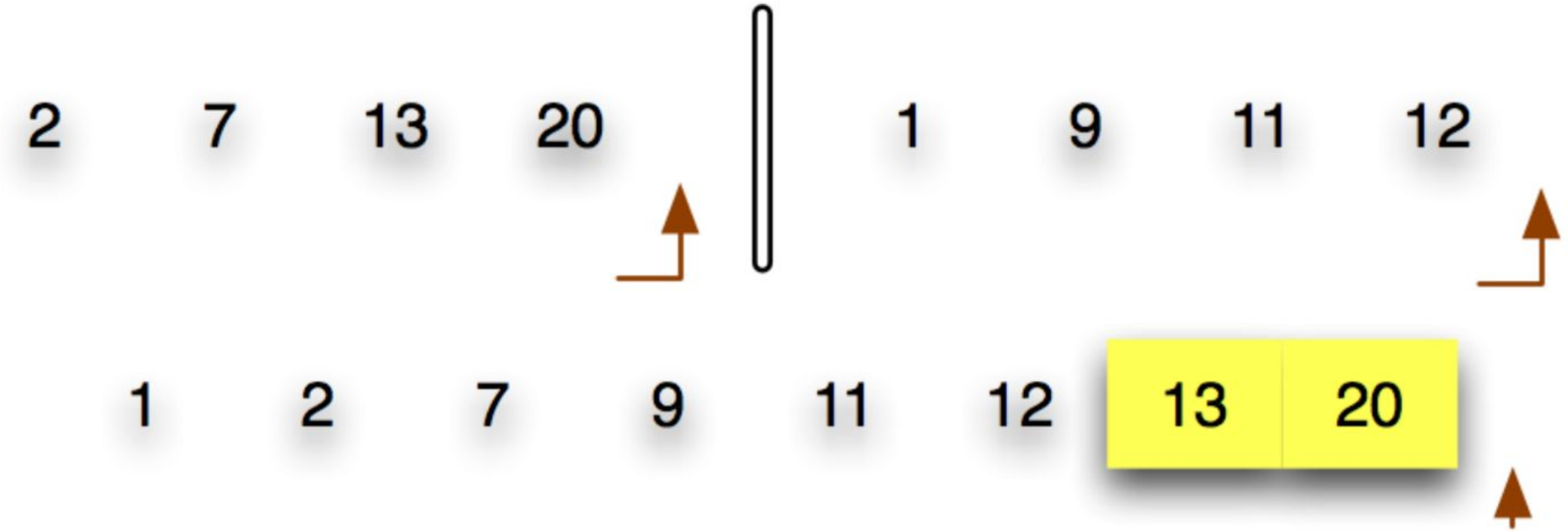
Merge (Simulation)



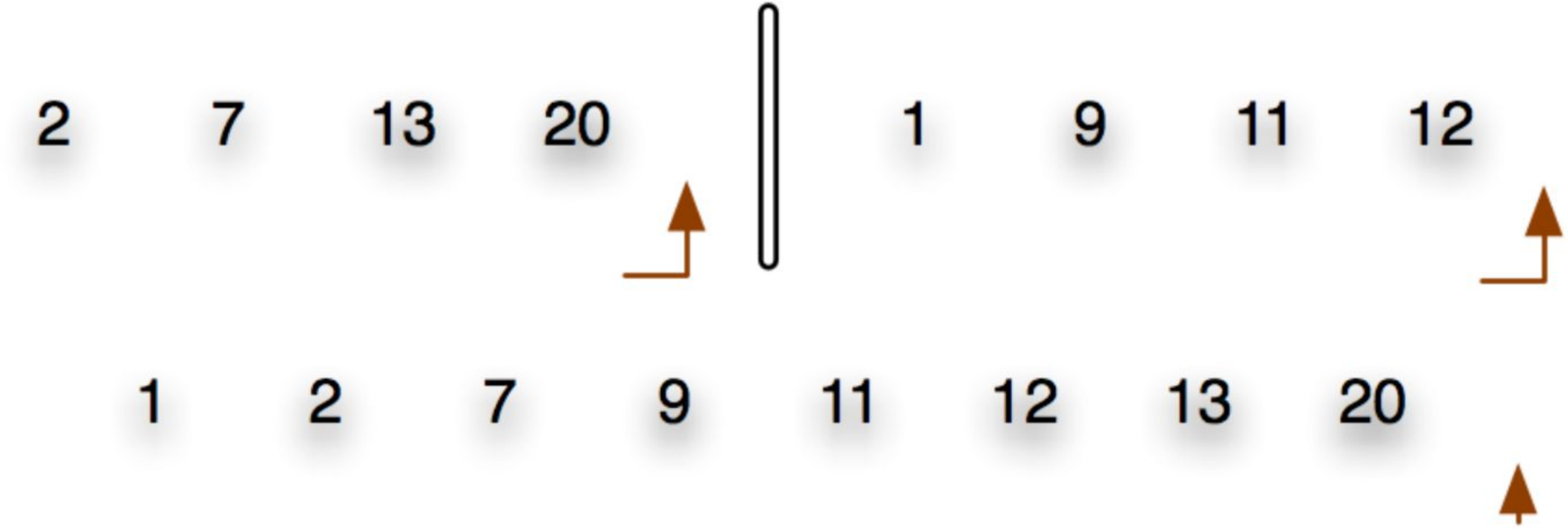
Merge (Simulation)



Merge (Simulation)



Merge (Simulation)



Merge (Algorithm)

MERGE(A, B)

INPUT: Two sorted arrays A and B
OUTPUT: Returns C as the merged array
$\triangleright$ $n_1 = length[A]$, $n_2 = length[B]$, $n = n_1 + n_2$
Create $C[1\,..\,n]$
Initialize two indices to point to A and B
while A and B are not empty
  do Select the smaller of two and add to end of C
    Advance the index that points to the smaller one
if A or B is not empty
  then Copy the rest of the non-empty array to the end of C
return C

Merge Sort (Algorithm)

Out-of-Place Algorithm

$\text{MERGE-SORT}(A)$ ▷ $A[1 \,..\, n]$

if $n = 1$
    then return
    else ▷ recursively sort the two subarrays
        $A_1 = \text{MERGE-SORT}(A[1 \,..\, \lceil n/2 \rceil])$
        $A_2 = \text{MERGE-SORT}(A[\lceil n/2 \rceil + 1 \,..\, n])$
        $A = \text{MERGE}(A_1, A_2)$ ▷ merge the sorted arrays

Quick Sort

Quicksort an n-element array:

1. *Divide* Partition the array into subarrays around a *pivot* x such that the elements in lower subarray $\leq x \leq$ elements in the upper subarray.

2. *Conquer* Recursively sort the two subarrays.
3. *Combine* Trivial – just concatenate the lower subarray, *pivot*, and the upper subarray.

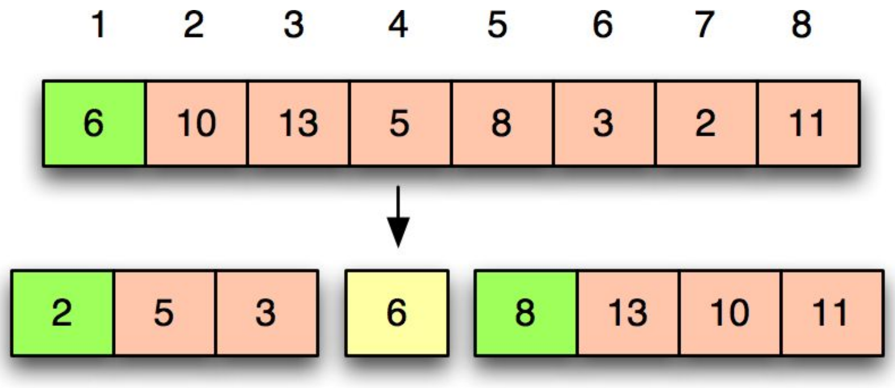
Quick Sort (Simulation)

1	2	3	4	5	6	7	8
6	10	13	5	8	3	2	11

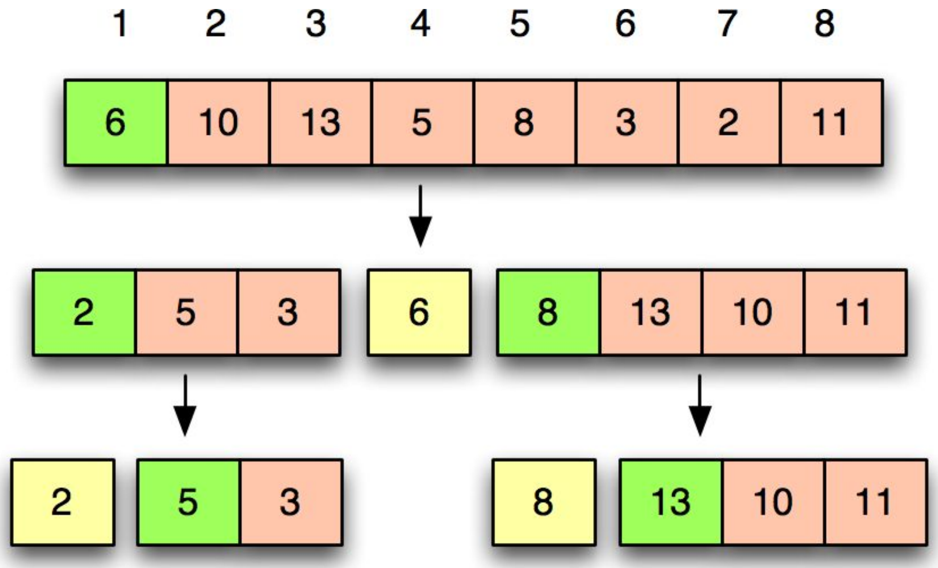
Quick Sort (Simulation)

1	2	3	4	5	6	7	8
6	10	13	5	8	3	2	11

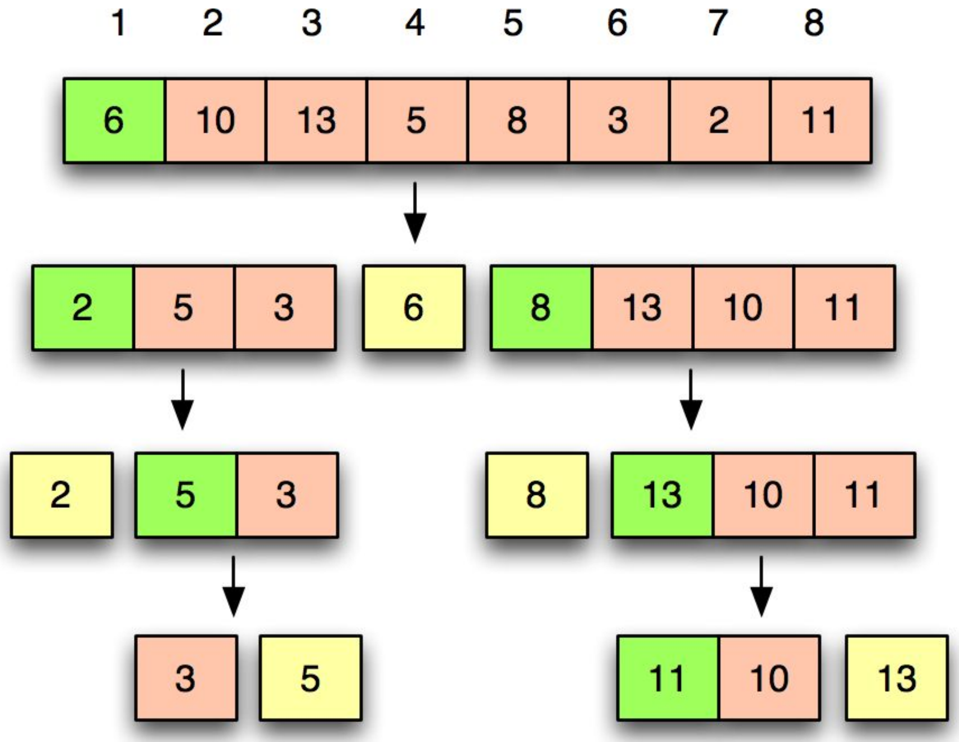
Quick Sort (Simulation)



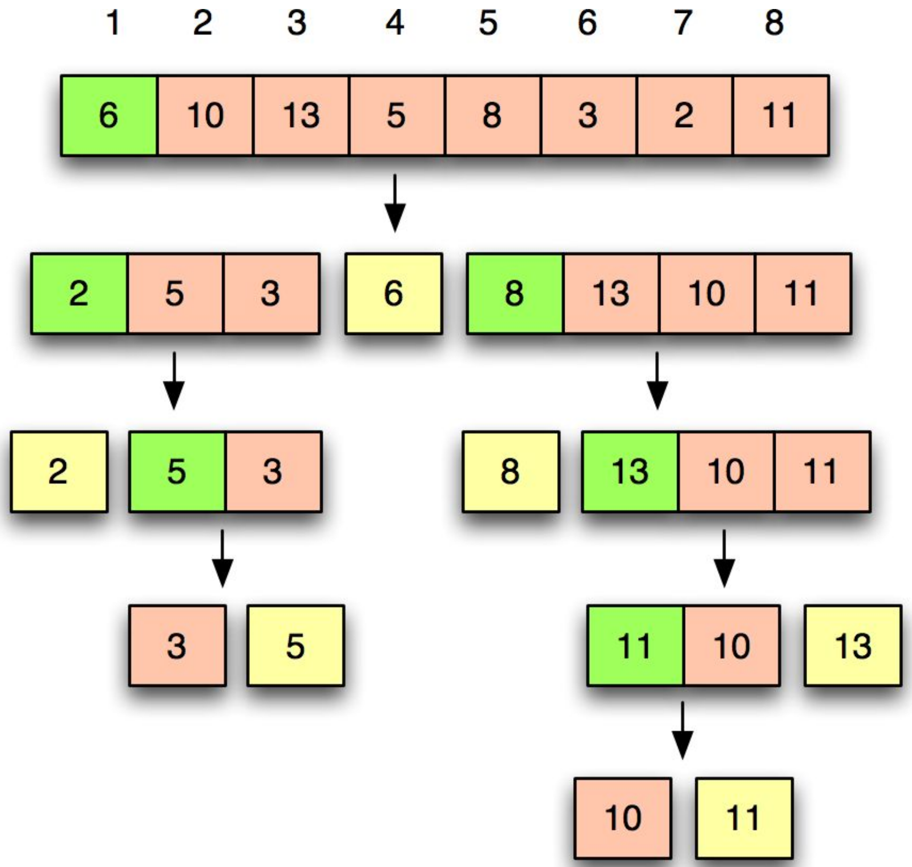
Quick Sort (Simulation)



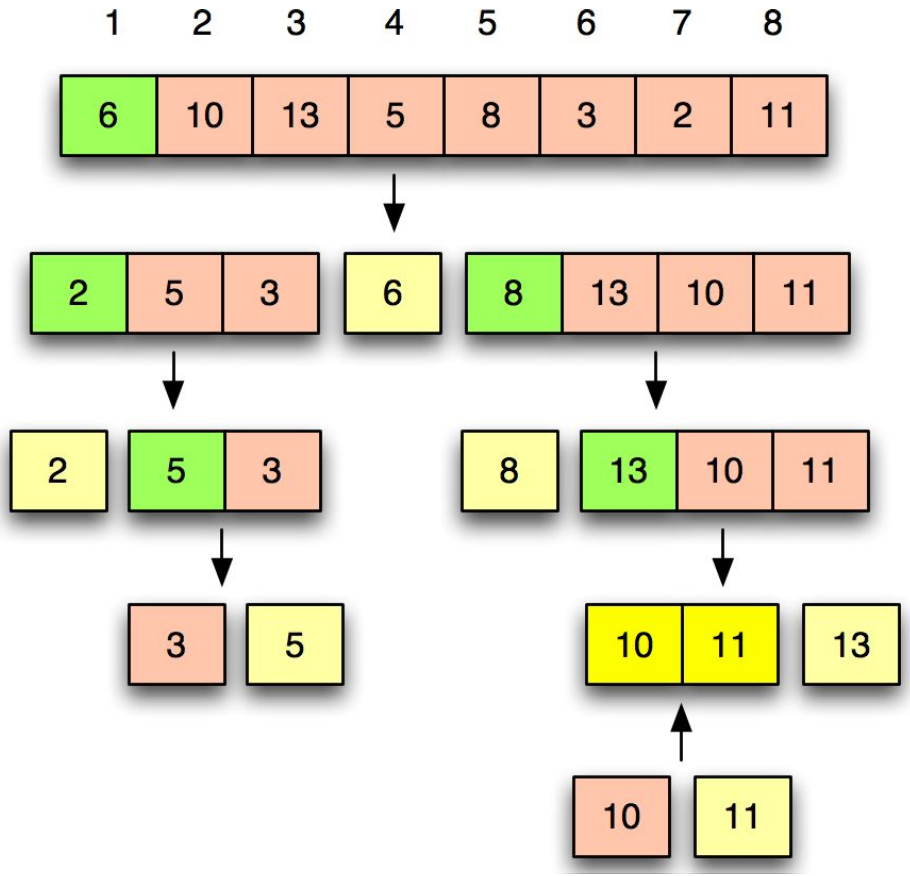
Quick Sort (Simulation)



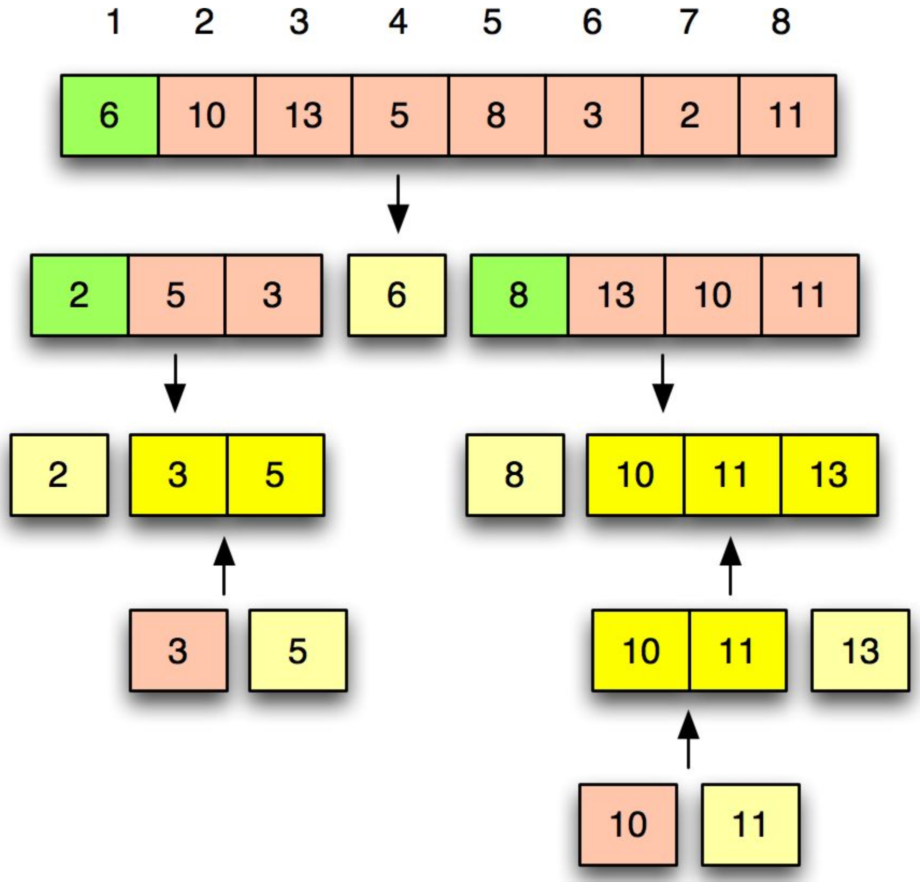
Quick Sort (Simulation)



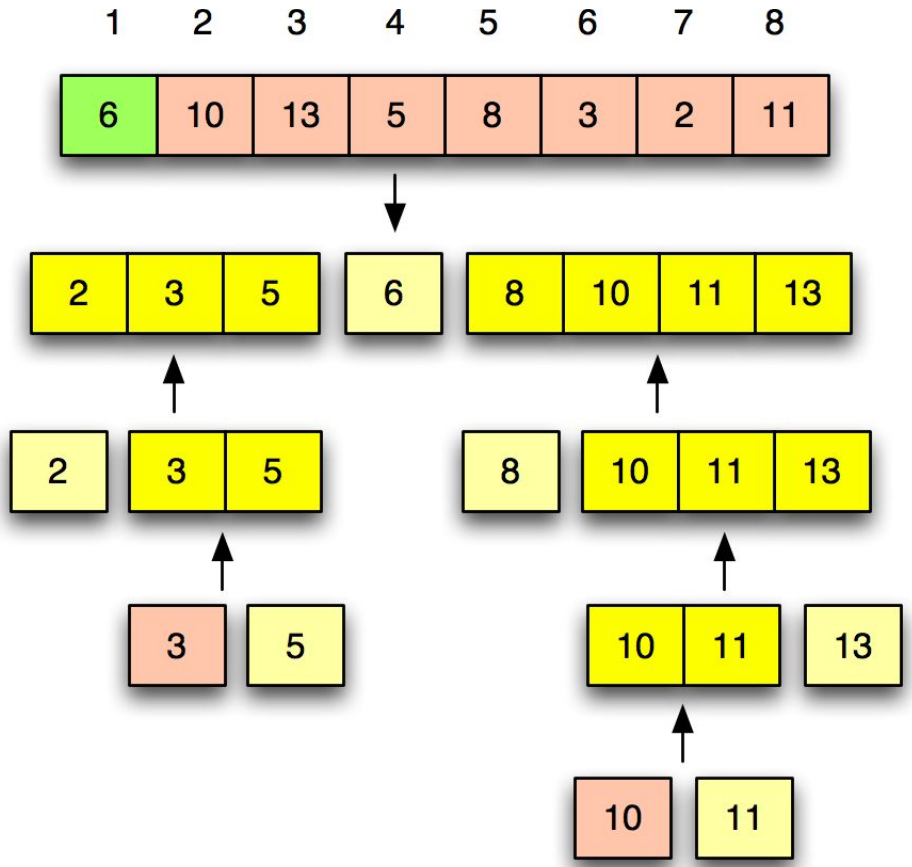
Quick Sort (Simulation)



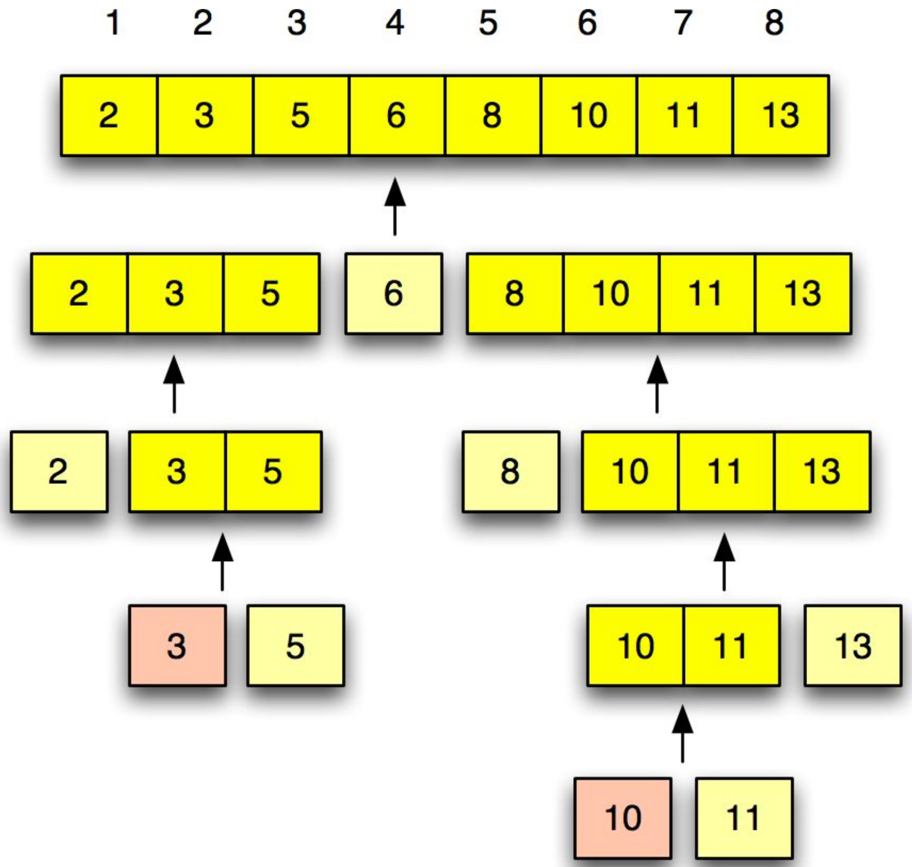
Quick Sort (Simulation)



Quick Sort (Simulation)



Quick Sort (Simulation)



Partitioning (Algorithm)

Algorithm

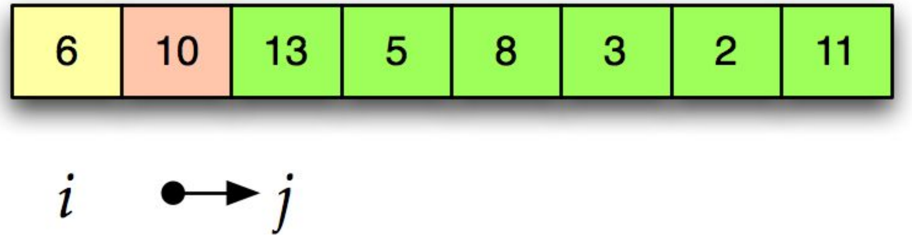
PARTITION(A, p, q) $\triangleright$ $A[p \,.\,.\, q]$

$x \leftarrow A[p]$ $\qquad\qquad\triangleright$ pivot $= A[p]$
$i \leftarrow p$
for $j \leftarrow p + 1$ **to** q
    do if $A[j] \leq x$
        then $i \leftarrow i + 1$
            exchange $A[i] \leftrightarrow A[j]$
exchange $A[p] \leftrightarrow A[i]$
return i

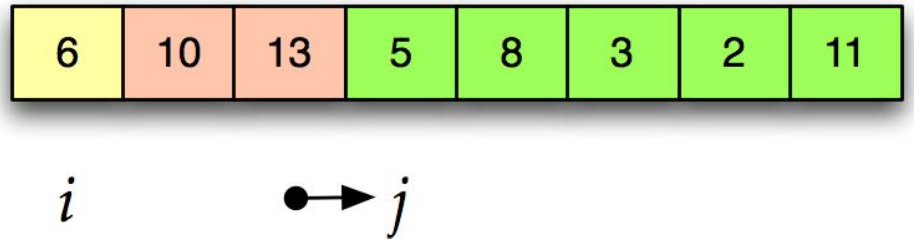
Partitioning (Simulation)

6	10	13	5	8	3	2	11
i	j						

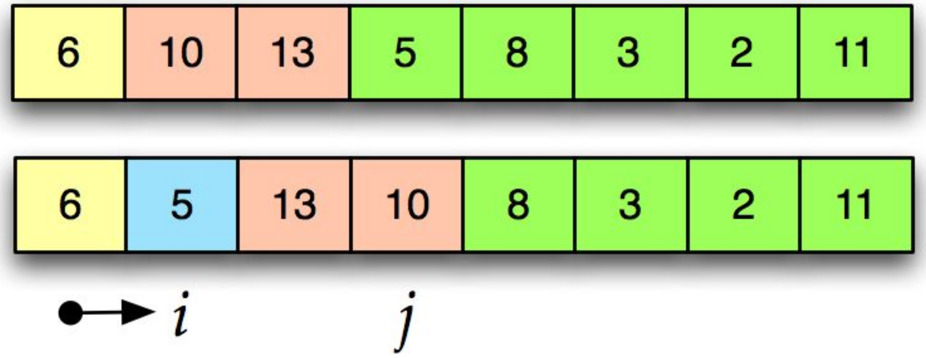
Partitioning (Simulation)



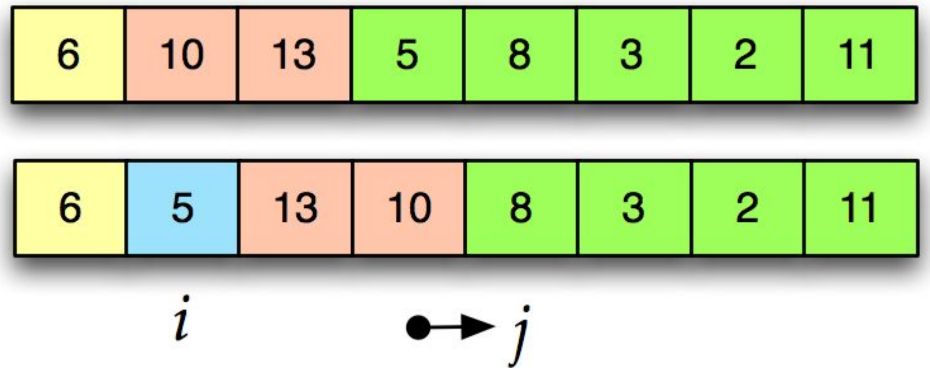
Partitioning (Simulation)



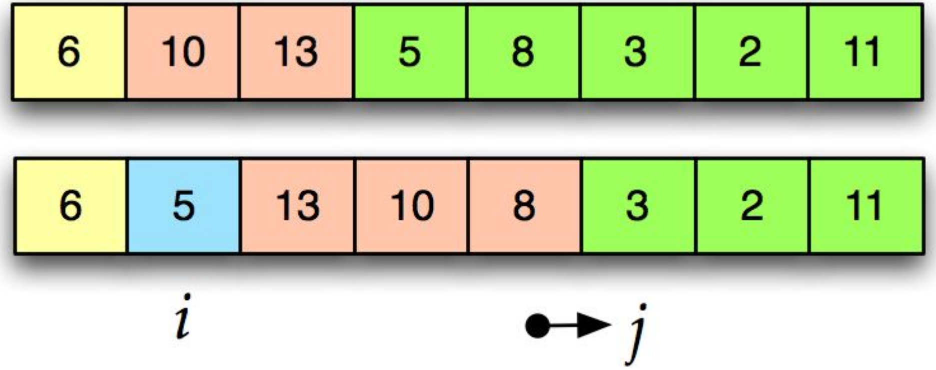
Partitioning (Simulation)



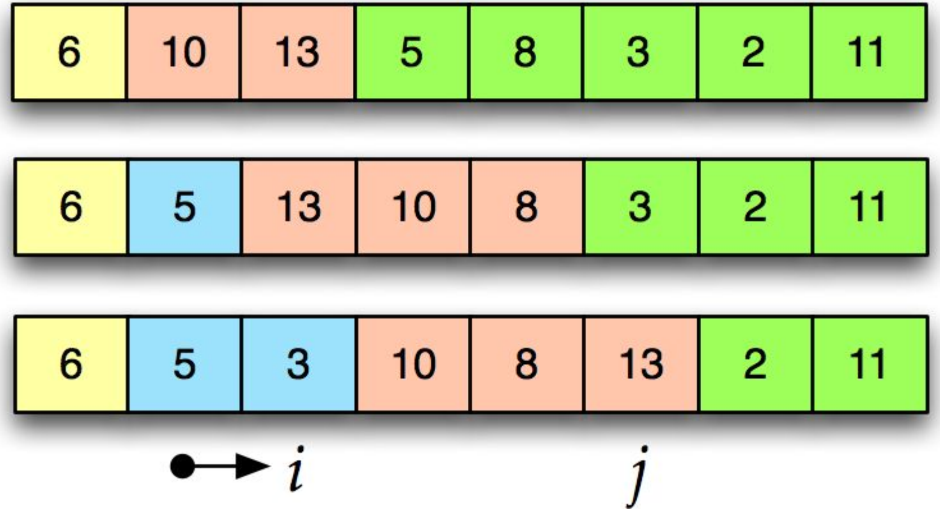
Partitioning (Simulation)



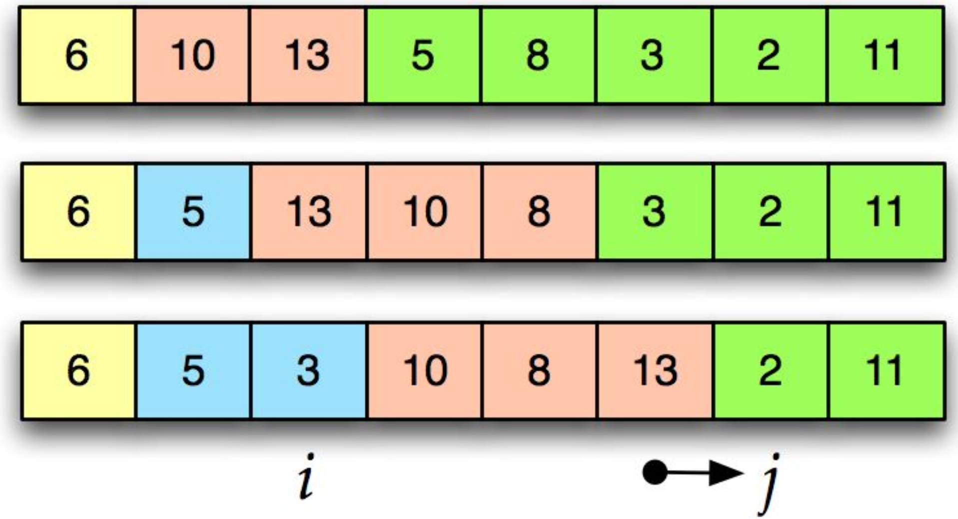
Partitioning (Simulation)



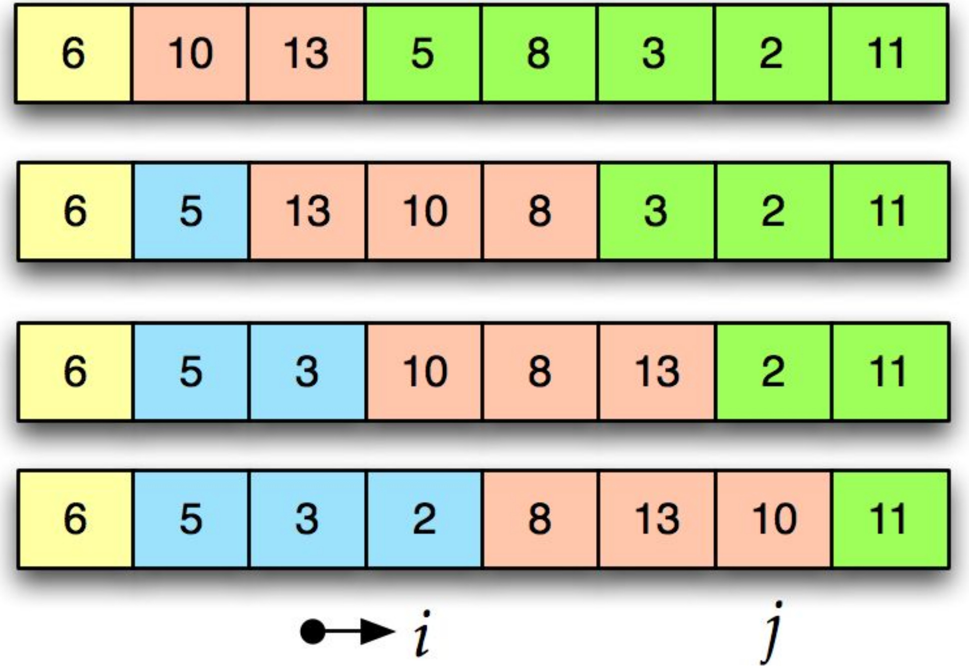
Partitioning (Simulation)



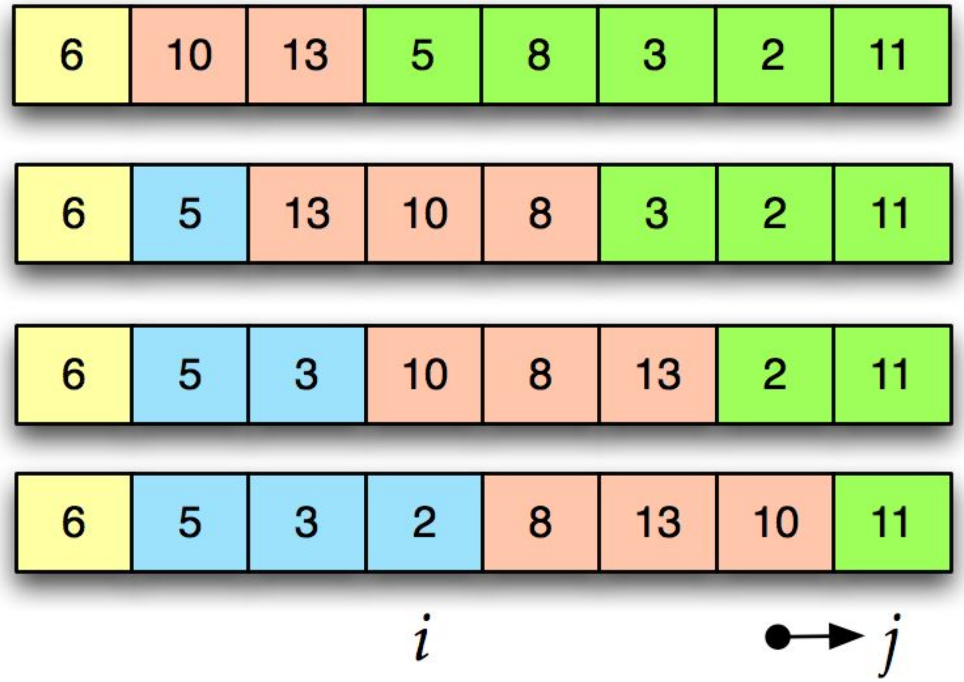
Partitioning (Simulation)



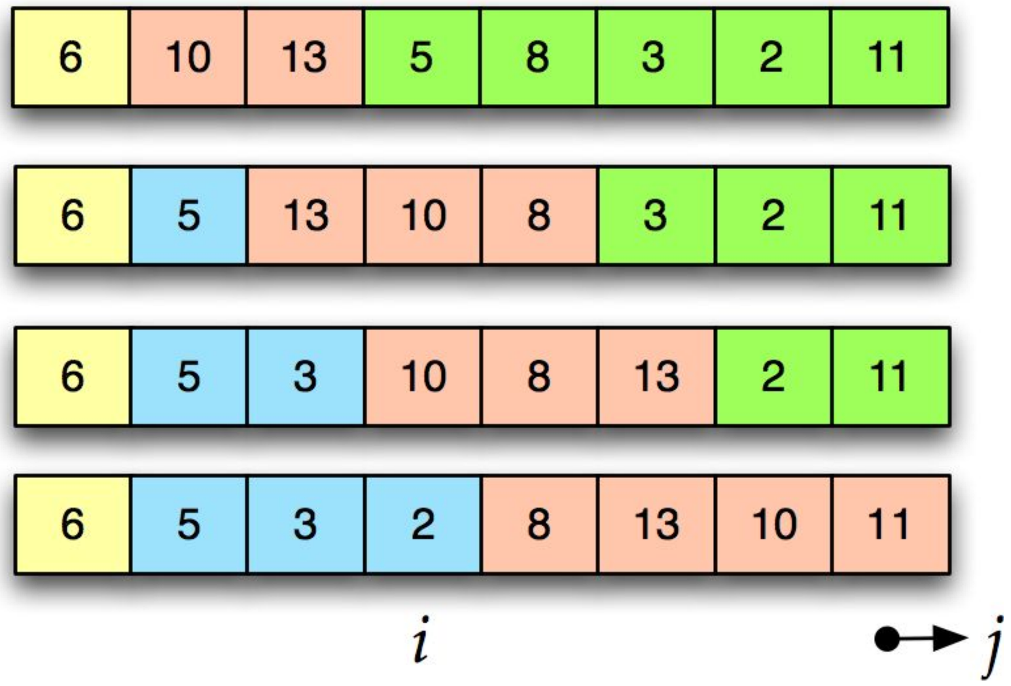
Partitioning (Simulation)



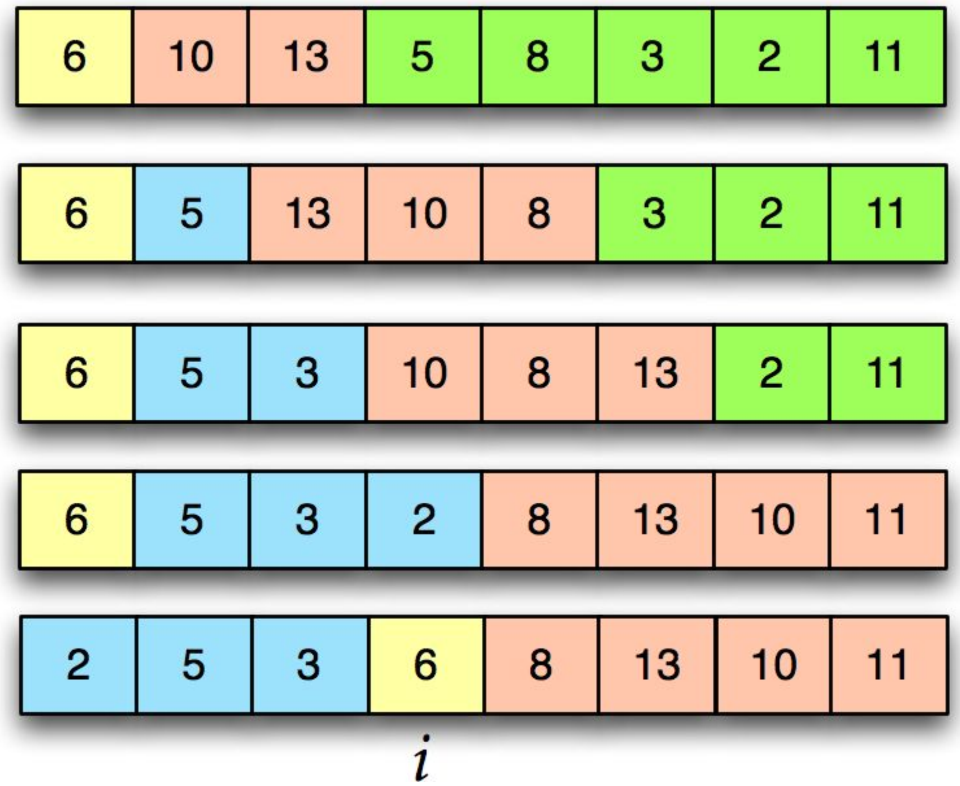
Partitioning (Simulation)



Partitioning (Simulation)



Partitioning (Simulation)



Quick Sort (Algorithm)

Algorithm

$\textsc{Quicksort}(A, p, r) \triangleright A[p \,..\, r]$

if $p < r$

  then $q \leftarrow \textsc{Partition}(A, p, r)$

    $\textsc{Quicksort}(A, p, q - 1)$

    $\textsc{Quicksort}(A, q + 1, r)$

Quick Sort (Soft Analysis - Worst Case)

- Worst-case happens when pivot is always the minimum or maximum element.

Quick Sort (Soft Analysis - Best Case)

- Best-case happens when pivot is the median element, creating equal size partitions.