Merge sort

- Merge sort is a well-known example of an algorithm design called Divide-and-Conquer consisting of the following 3 steps:
  - Divide: divide the given instance into smaller instances.
  - Conquer: solve all of the smaller instances.
  - Combine: combine the outcomes of the smaller instances.

Merge sort

MERGE-SORT $A[1 . . n]$

1. If $n = 1$, done.
2. Recursively sort $A[1 . . .n/2.]$ and $A[[n/2]+1 . . n]$.
3. "Merge" the 2 sorted lists.

Key subroutine: MERGE

Merge Sort

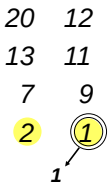
```
MergeSort(A, left, right) {
  if (left < right) {
      mid = floor((left + right) / 2);
      MergeSort(A, left, mid);
      MergeSort(A, mid+1, right);
      Merge(A, left, mid, right);
  }
}

// Merge() takes two sorted subarrays of A and
// merges them into a single sorted subarray of A
//    (how long should this take?)
```

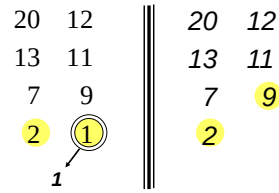

Merging two sorted arrays

20	12
13	11
7	9
2	1

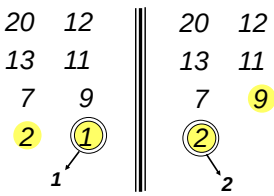
Merging two sorted arrays



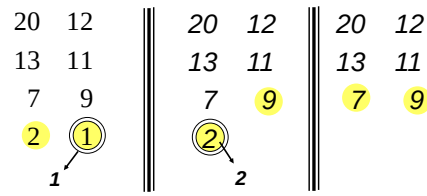
Merging two sorted arrays



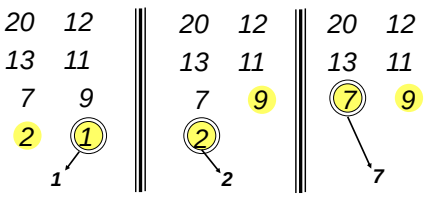
Merging two sorted arrays



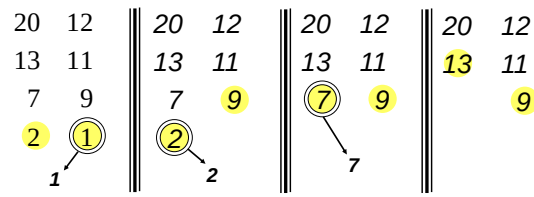
Merging two sorted arrays



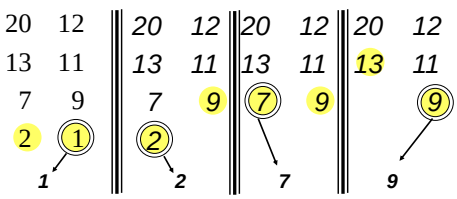
Merging two sorted arrays



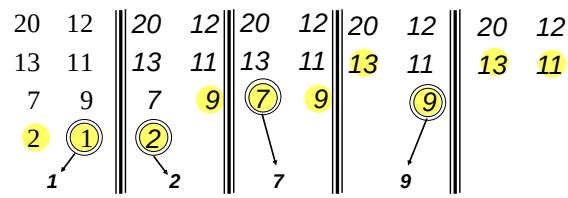
Merging two sorted arrays



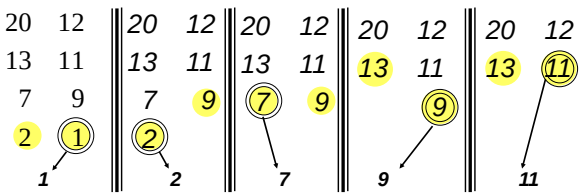
Merging two sorted arrays



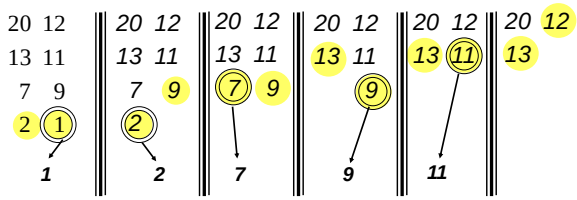
Merging two sorted arrays



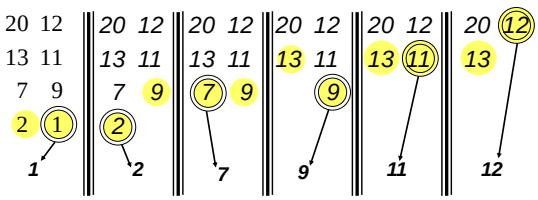
Merging two sorted arrays



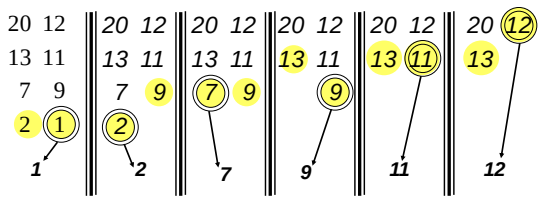
Merging two sorted arrays



Merging two sorted arrays



Merging two sorted arrays



Time = Θ(n) to merge a total of n elements (linear time).

Procedure Merge

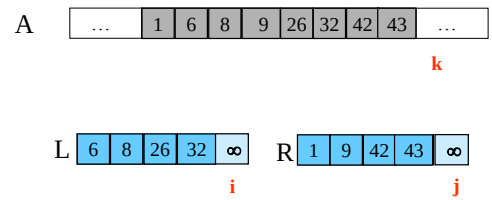
```
Merge(A, p, q, r)
1  n1 = q – p + 1
2  n2 = r – q
3  //let L[1.. n1 +1] and R[1.. n2+1] to be new arrays
4  for i = 1 to n1
5       L[i] = A[p + i – 1]
6  for j = 1 to n2
7       R[j] = A[q + j]
8  L[n1+1] = ∞
9  R[n2+1] = ∞
10 i = 1
11 j = 1
12 for k = p to r
13      if L[i] ≤ R[j]
14           A[k] = L[i]
15           i = i + 1
16      else  A[k] = R[j]
17           j = j + 1
```

Input: Array containing sorted subarrays A[p..q] and A[q+1..r].

Output: Merged sorted subarray in A[p..r].

Sentinels, to avoid having to check if either subarray is fully copied at each step.

Merge – Example



Merge-Sort (A, p, r)

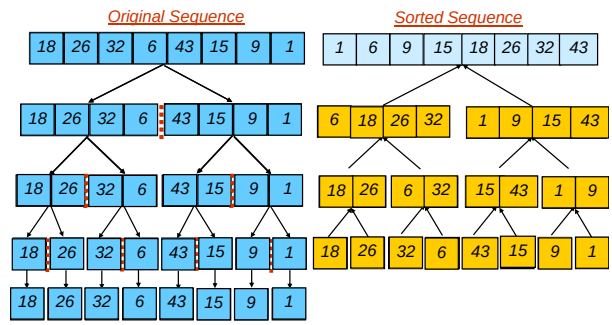
INPUT: a sequence of *n* numbers stored in array A

OUTPUT: an ordered sequence of *n* numbers

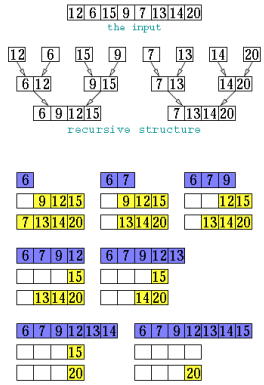
```
MergeSort (A, p, r)   // sort A[p..r] by divide & conquer
1   if p < r
2       then q = ⌊(p+r)/2⌋
3            MergeSort (A, p, q)
4            MergeSort (A, q+1, r)
5            Merge (A, p, q, r) // merges A[p..q] with
       A[q+1..r]
```

Initial Call: MergeSort(A, 1, n)

Merge Sort – Example



Another Example: Merge Sort



Analysis of Merge Sort

Statement	Effort
`MergeSort(A, left, right) {`	`T(n)`
`  if (left < right) {`	`Θ(1)`
`    mid = floor((left + right) / 2);`	`Θ(1)`
`    MergeSort(A, left, mid);`	`T(n/2)`
`    MergeSort(A, mid+1, right);`	`T(n/2)`
`    Merge(A, left, mid, right);`	`Θ(n)`
`  }`	
`}`	

- So T(n) = $\Theta(1)$ when n = 1, and
  $2T(n/2) + \Theta(n)$ when n > 1
- So what (more succinctly) is T(n)?