# CSE3318: Divide & Conquer Algorithm
## Merge Sort
## by
## Dr. Bhanu Jain



All slides are based on: ***Introduction to Algorithms***, by Thomas H. Cormen, Charles E. Leiserson, Ronald E. Rivest, Clifford Stein,3rd edition (CLRS)

# Merge Sort

**MERGE-SORT** $A[1\,..\,n]$

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

*Key subroutine:* MERGE

# Merge Sort

**MERGE-SORT (*A, p, r*)**

1. if p < r
2. q =⌊ (p + r) /2⌋
3. MERGE-SORT(A,p,q)
4. MERGE-SORT(A,q+1,r)
5. MERGE(A,p,q,r)

***Key subroutine:* MERGE**

MERGE-SORT(A,p,r)
Sorts the elements in the subarray A[p..r]
If p<=r, subarray A has at most 1 element-sorted

# Merge Sort

MERGE-SORT(A, p, r)
1. if p < r
2. q = ⌊(p + r) / 2⌋
3. MERGE-SORT(A, p, q)
4. MERGE-SORT(A, q + 1, r)
5. MERGE(A, p, q, r)

**Divide Recursively**: The array is divided recursively into two halves until each subarray has one element (base case).
If p<r, subarray A has at least 2 elements

**Merge**: The MERGE function merges two sorted subarrays back into a single sorted array.

# Merge Sort

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r) / 2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

1. Divide the array into two parts.
2. Use two pointers (i and j) to traverse the sorted subarrays L and R.
3. Compare the elements at the pointers and copy the smaller one into the array A.
4. Use sentinel values (∞) to not have to check the array boundaries explicitly.
5. Efficient merging technique for two sorted subarrays in O(n) time, where n=r−p+1

# Merge Sort

```
MERGE-SORT(A, p, r)
1. if p < r
2.     q = ⌊(p + r) / 2⌋
3.     MERGE-SORT(A, p, q)
4.     MERGE-SORT(A, q + 1, r)
5.     MERGE(A, p, q, r)
```

**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]
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
```

1. Computes n1, the size of the subarray A[p..q].
2. Computes n2, the size of the subarray A[q+1..r].

4-5. Copies elements of the subarray A[p..q] into L[1..n1].

6-7. Copies elements of the subarray A[q+1..r] into R[1..n2].

8-9. Adds sentinel values to the ends of the arrays L and R.

10-11. Initializes i and j to start reading from arrays L and R.

10-11. L[i] and R[j] hold the smallest elements of L and R not yet merged into A.

12. The for loop executes r - p + 1 basic steps.

12-17. After each step, A[p..k-1] contains the k - p smallest elements of L[1..n1+1] and R[1..n2+1] in sorted order.

# Merge Procedure Runs in $\Theta$(n) Time

```
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]
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
```

**n** = $q - p + 1$

**1-3** Constant time

**8-11** Constant time

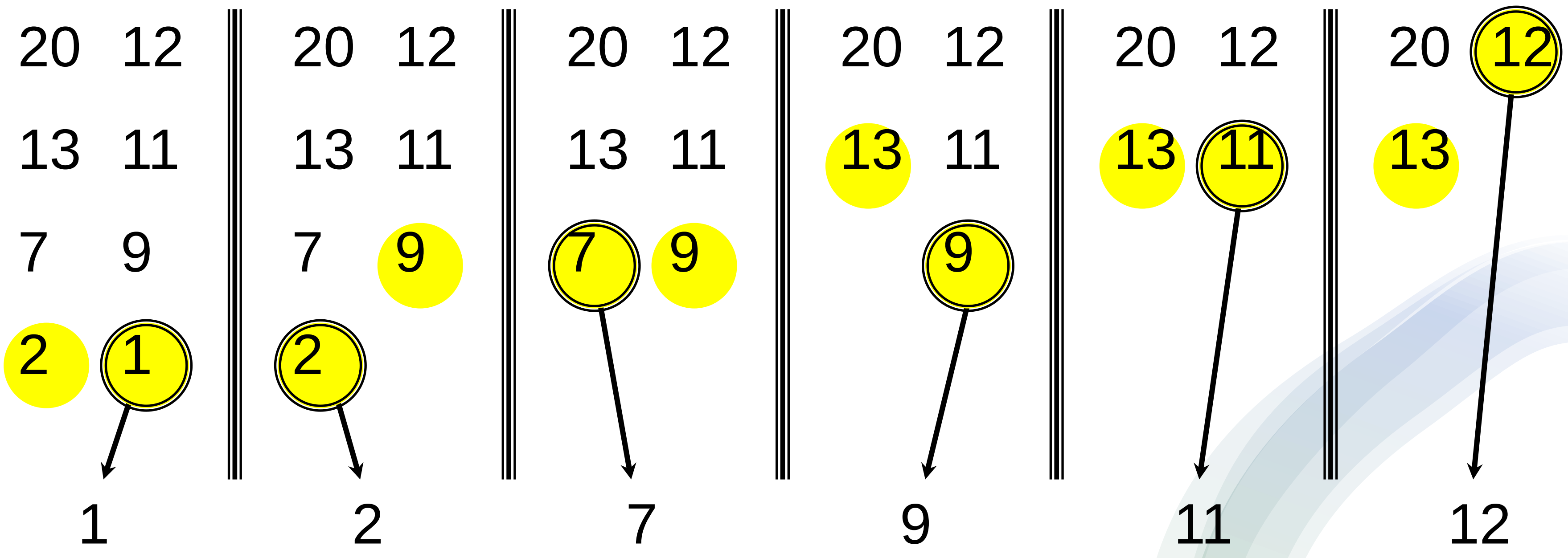
**4-7** $\Theta(n_1+n_2) = \Theta(n)$ time

**12-17** **n** iterations of the for loop (each constant time)

# Merge Procedure Runs in $\Theta$(n) Time

**MERGE(A, p, q, r)**

```
n1 = q - p + 1
n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1]
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```




Time $= \Theta(n)$ to merge a total of $n$ elements (linear time).

Image : Courtesy Dr. Junzhou Huang

# Analyzing Divide and Conquer Algorithms

- Division of problem yields a subproblems, each of which is *1*/b the size of the original
- D(*n*) time to divide the problem into subproblems
- C(n) time to combine the solutions
- *c* is some constant

$$T(n) = \begin{cases} \Theta(1) \text{ if } n \leq c; \\ aT(n/b) + D(n) + C(n) \text{ otherwise.} \end{cases}$$

# Recurrence for Merge Sort

- We will omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small $n$, but only when it has no effect on the asymptotic solution to the recurrence.
- **Divide:** Compute the middle of the subarray – constant time D(n) = $\Theta(1)$
- **Conquer:** Recursively solve two subproblems, each of size n/2- contributes to 2T(n/2) running time
- **Combine:** Merge procedure on n-element subarray takes $\Theta(\mathrm{n})$ time. Therefore, C(n)= $\Theta(\mathrm{n})$. (see next slide)

$$T(n) = \begin{cases} \Theta(1) \text{ if } n \le c; \\ aT(n/b) + D(n) + C(n) \text{ otherwise.} \end{cases}$$

$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

# Recurrence for Merge Sort

- We will omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small $n$, but only when it has no effect on the asymptotic solution to the recurrence.
- **Divide:** Compute the middle of the subarray – constant time D(n) = $\Theta(1)$
- **Conquer:** Recursively solve two subproblems, each of size n/2- contributes to 2T(n/2) running time
- **Combine:** Merge procedure on n-element subarray takes $\Theta(n)$ time. Therefore, C(n)= $\Theta(n)$

```
MERGE-SORT(A, p, r)
1. if p < r
2.     q = ⌊(p + r) / 2⌋
3.     MERGE-SORT(A, p, q)
4.     MERGE-SORT(A, q + 1, r)
5.     MERGE(A, p, q, r)
```

1.If $n = 1$, done.
*3-4*. Recursively sort $A[\,1\,..\lceil n/2\rceil\,]$ and $A[\,\lceil n/2\rceil+1\,..\,n\,]$ .
*5*.**"Merge"** the 2 sorted lists

$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$

$$T(n) = \begin{cases} \Theta(1) \text{ if } n \le c; \\ aT(n/b) + D(n) + C(n) \text{ otherwise.} \end{cases}$$

Should be $T(\lceil n/2\rceil) + T(\lfloor n/2\rfloor)$ , but it is irrelevant for asymptotic analysis.

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r) / 2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

# Recurrence for Merge Sort

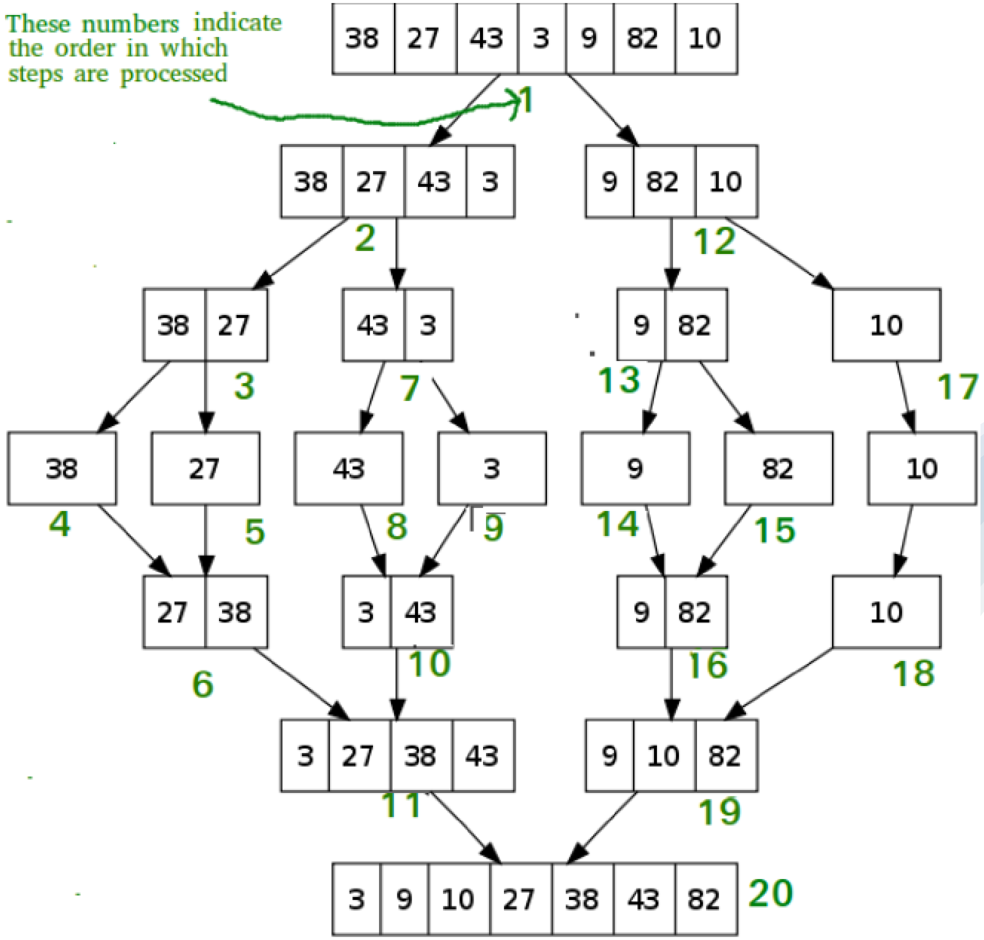



Image Credit:Geeks for Geeks

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r) / 2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

# Recurrence for Merge Sort

Given array is [38, 27, 43, 3, 9, 82, 10]

**p,q,r**

1 4 7

1 2 4

1 1 2 **First MERGE() call**

3 3 4

5 6 7

5 5 6

Sorted array is [3, 9, 10, 27, 38, 43, 82]

Geeks for Geeks

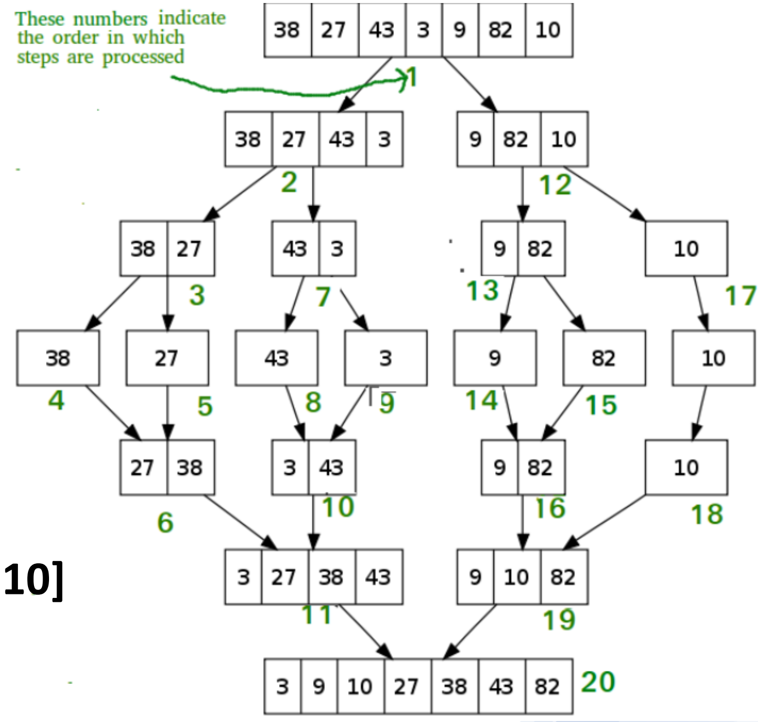
# Recurrence for Merge Sort

```
MERGE-SORT(A, p, r)
1. if p < r
2.     q = ⌊(p + r) / 2⌋
3.     MERGE-SORT(A, p, q)
4.     MERGE-SORT(A, q + 1, r)
5.     MERGE(A, p, q, r)
```

**End of/inside 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] 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
```

$$T(n) = \begin{cases} \Theta(1) \text{ if } n = 1; \\ 2T(n/2) + \Theta(n) \text{ if } n > 1. \end{cases}$$




**Given array is [38, 27, 43, 3, 9, 82, 10]**

**p,q,r, A**

1 4 7

1 2 4

1 1 2

1 1 2 [**27, 38**, 43, 3, 9, 82, 10] L= [38] R= [27]

3 3 4

3 3 4 [27, 38, **3, 43**, 9, 82, 10] L= [43] R= [3]

1 2 4 [3, 27, 38, 43, 9, 82, 10] L= [27, 38] R= [3, 43]

5 6 7

5 5 6

5 5 6 [3, 27, 38, 43, **9, 82**, 10] L= [9] R= [82]

5 6 7 [3, 27, 38, 43, **9, 10, 82**] L= [9, 82] R= [10]

1 4 7 [**3, 9, 10, 27, 38, 43, 82**] L= [3, 27, 38, 43] R= [9, 10, 82]

Sorted array is [3, 9, 10, 27, 38, 43, 82]

# The END!