CSE 203: Data Structures and Algorithms-I

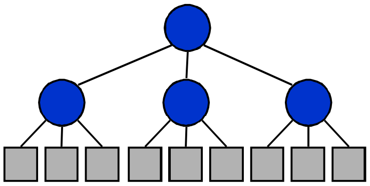
Divide-and-Conquer Technique

Arrays: Merge Sort, Quick Sort

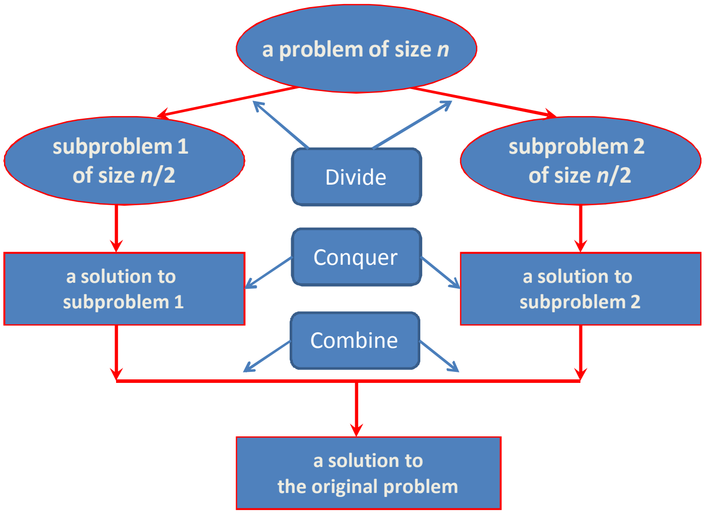
Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Divide-and-Conquer Technique

- Divide-and-Conquer is a general algorithm design paradigm:
  - Divide the problem into a number of subproblems that are smaller instances of the same problem
  - Conquer the subproblems by solving them recursively
  - Combine the solutions to the subproblems into the solution for the original problem
- The base case for the recursion are subproblems of constant size
- Analysis can be done using **recurrence equations**



Divide-and-Conquer



a problem of size *n*

Divide

subproblem 1 of size *n*/2

subproblem 2 of size *n*/2

Conquer

a solution to subproblem 1

a solution to subproblem 2

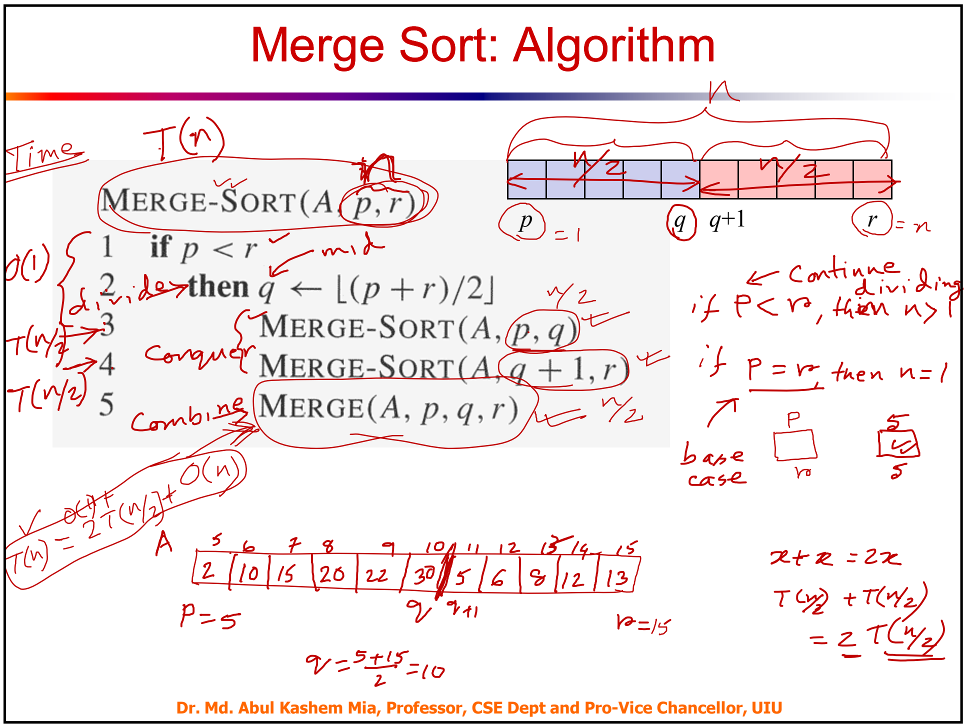
Combine

a solution to the original problem

Merge Sort and Quick Sort

Two well-known sorting algorithms adopt this divide-and-conquer strategy

- Merge sort
  - Divide step is trivial – just split the list into two equal parts
  - Work is carried out in the conquer step by merging two sorted lists
- Quick sort
  - Work is carried out in the divide step using a pivot element
  - Conquer step is trivial



Merge Sort: Algorithm

Time T(n)

```
MERGE-SORT(A, p, r)
    if $p < r$
        then $q \leftarrow \lfloor (p + r)/2 \rfloor$
            MERGE-SORT(A, p, q)
            MERGE-SORT(A, q + 1, r)
            MERGE(A, p, q, r)
```

O(1) divide mid

T(n/2) T(n/2) Conquer

Combine n/2

$T(n) = 2T(n/2) + O(n)$ O(1)+

n

n/2 n/2

p = 1 q q+1 r = n

← Continue dividing if p < r, then n > 1

if p = r then n = 1

base case p r 5 5 5

A

5	6	7	8	9	10	11	12	13	14	15
2	10	15	20	22	30	5	6	8	12	13

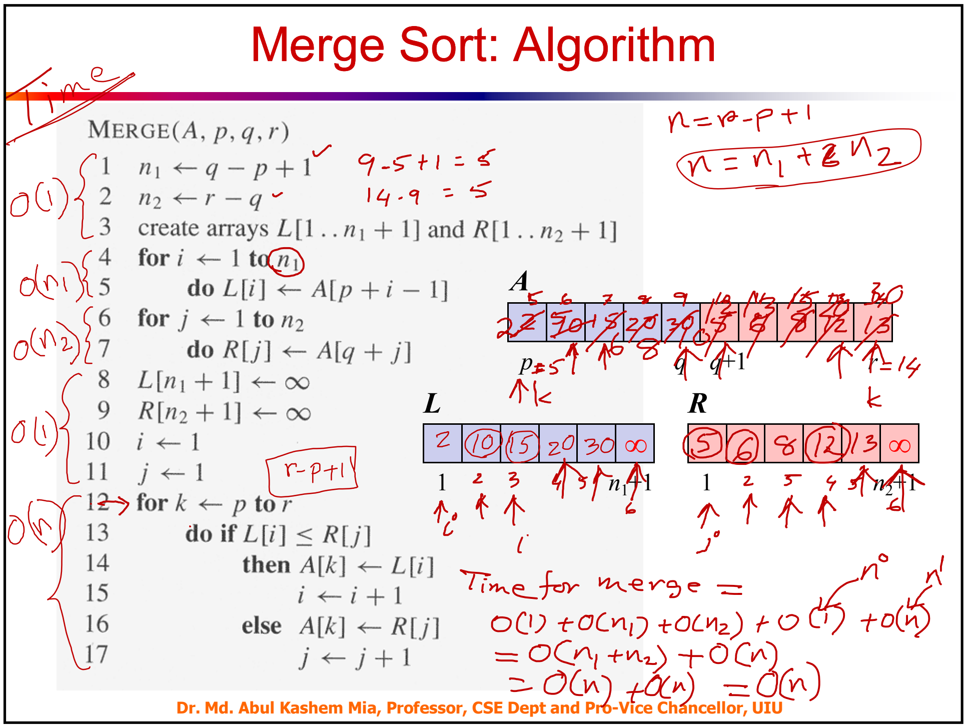
p = 5 q q+1 r = 15

$q = \frac{5+15}{2} = 10$

$x + x = 2x$

$T(n/2) + T(n/2) = 2T(n/2)$

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Merge Sort: Algorithm

Time

```
MERGE(A, p, q, r)
    $n_1 \leftarrow q - p + 1$
    $n_2 \leftarrow r - q$
    create arrays $L[1 \,..\, n_1 + 1]$ and $R[1 \,..\, n_2 + 1]$
    for $i \leftarrow 1$ to $n_1$
        do $L[i] \leftarrow A[p + i - 1]$
    for $j \leftarrow 1$ to $n_2$
        do $R[j] \leftarrow A[q + j]$
    $L[n_1 + 1] \leftarrow \infty$
    $R[n_2 + 1] \leftarrow \infty$
    $i \leftarrow 1$
    $j \leftarrow 1$
    for $k \leftarrow p$ to $r$
        do if $L[i] \leq R[j]$
            then $A[k] \leftarrow L[i]$
                $i \leftarrow i + 1$
            else $A[k] \leftarrow R[j]$
                $j \leftarrow j + 1$
```

O(1) O(n1) O(n2) O(1) O(n) r-p+1

9-5+1 = 5

14-9 = 5

$n = r - p + 1$

$n = n_1 + n_2$

A

5	6	7	8	9	10	11	12	13	14
2	10	15	20	30	5	6	8	12	13

p = 5 q q+1 r = 14 k

L

1	2	3	4	5	n_1+1
2	10	15	20	30	∞

i

R

1	2	3	4	5	n_2+1
5	6	8	12	13	∞

j

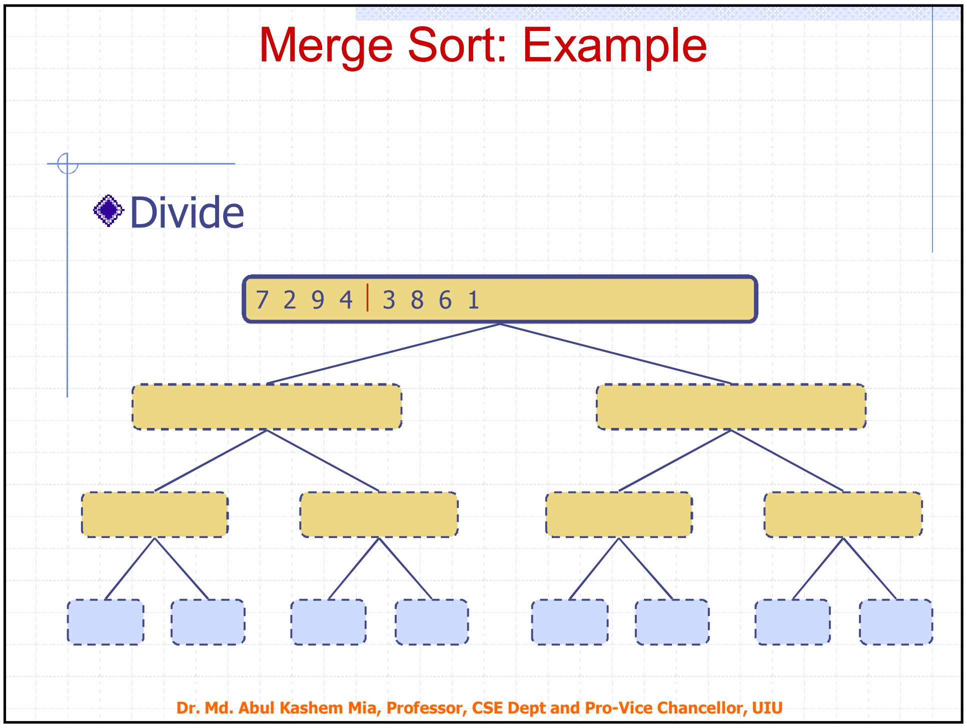
Time for merge =

$O(1) + O(n_1) + O(n_2) + O(1) + O(n)$

$= O(n_1 + n_2) + O(n)$

$= O(n) + O(n) = O(n)$

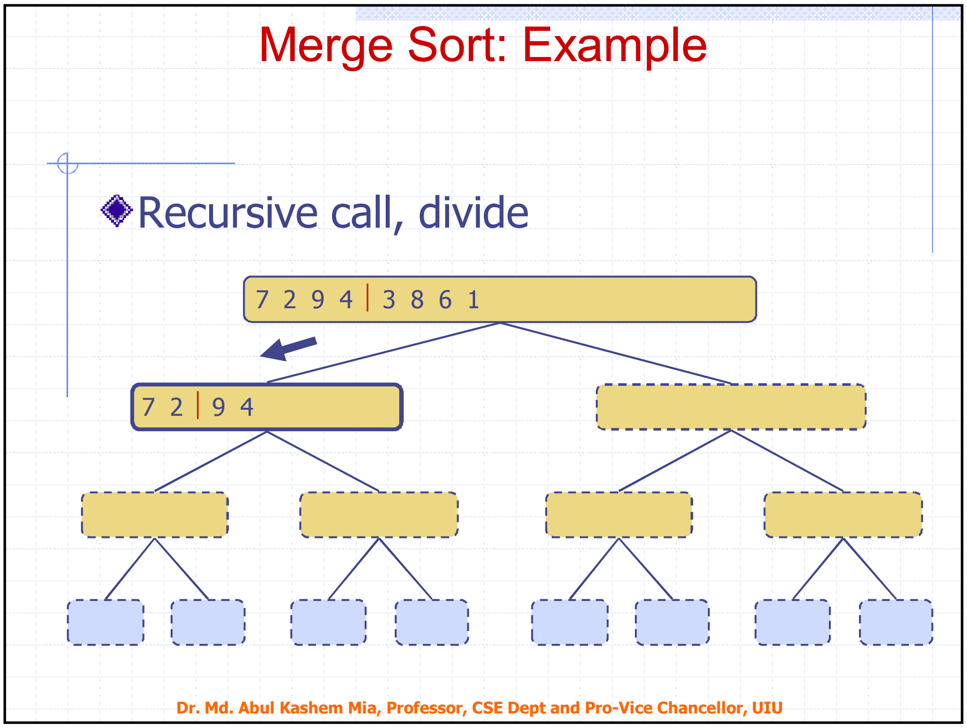
Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Merge Sort: Example

- Divide

7 2 9 4 | 3 8 6 1

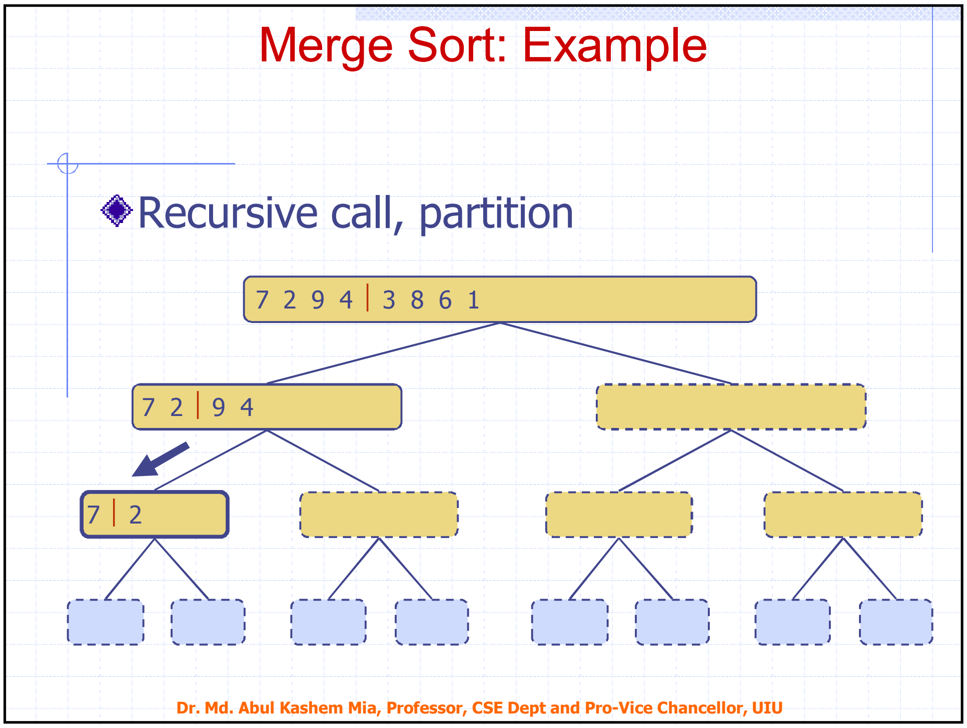


Merge Sort: Example

- Recursive call, divide

7 2 9 4 | 3 8 6 1

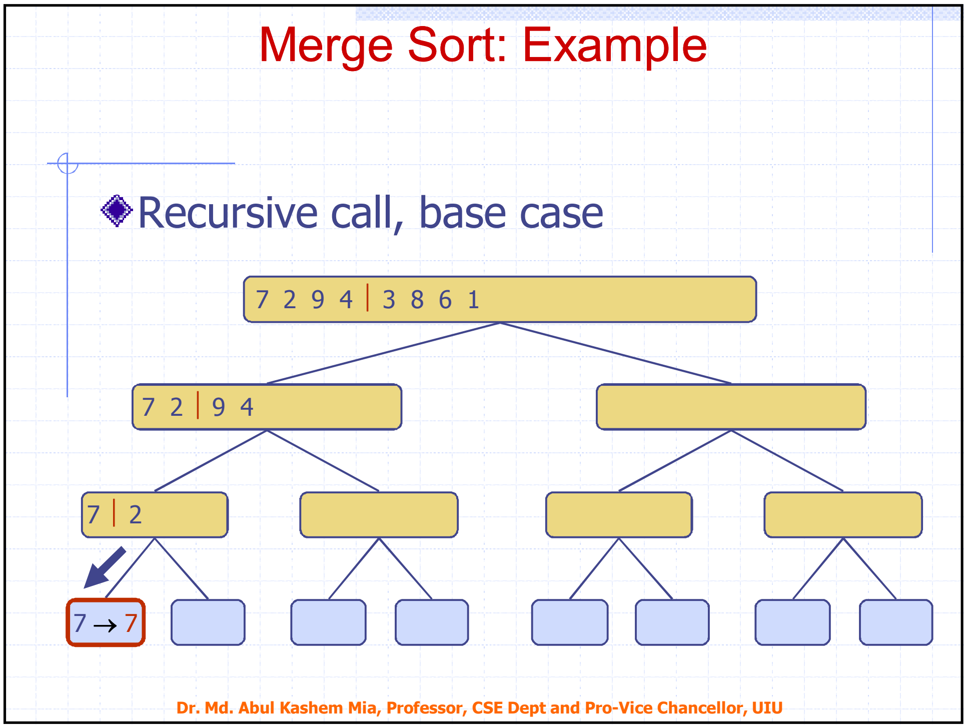
7 2 | 9 4



Merge Sort: Example

- Recursive call, partition

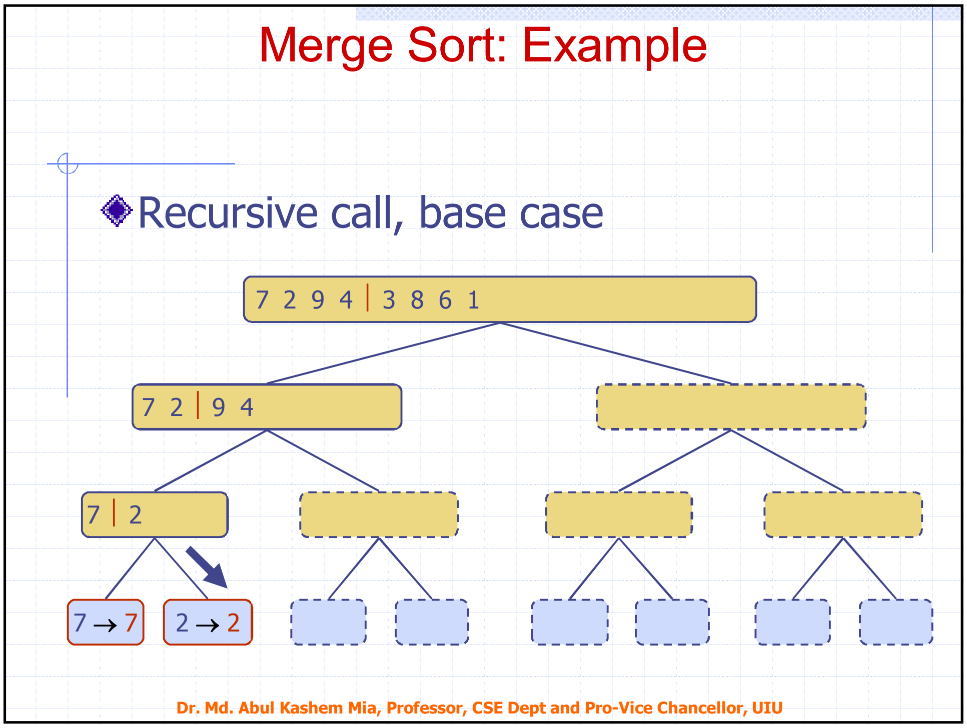
Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Merge Sort: Example

- Recursive call, base case

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Merge Sort: Example

- Recursive call, base case

7 2 9 4 | 3 8 6 1

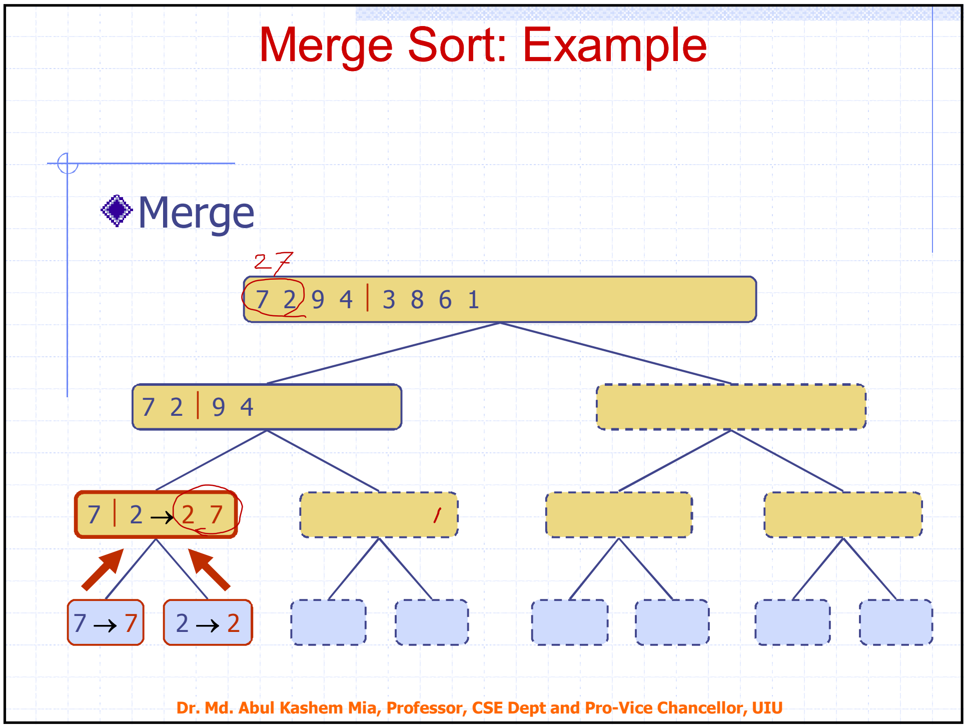
7 2 | 9 4

7 | 2

7 → 7

2 → 2

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Merge Sort: Example

- Merge

27

7 2 9 4 | 3 8 6 1

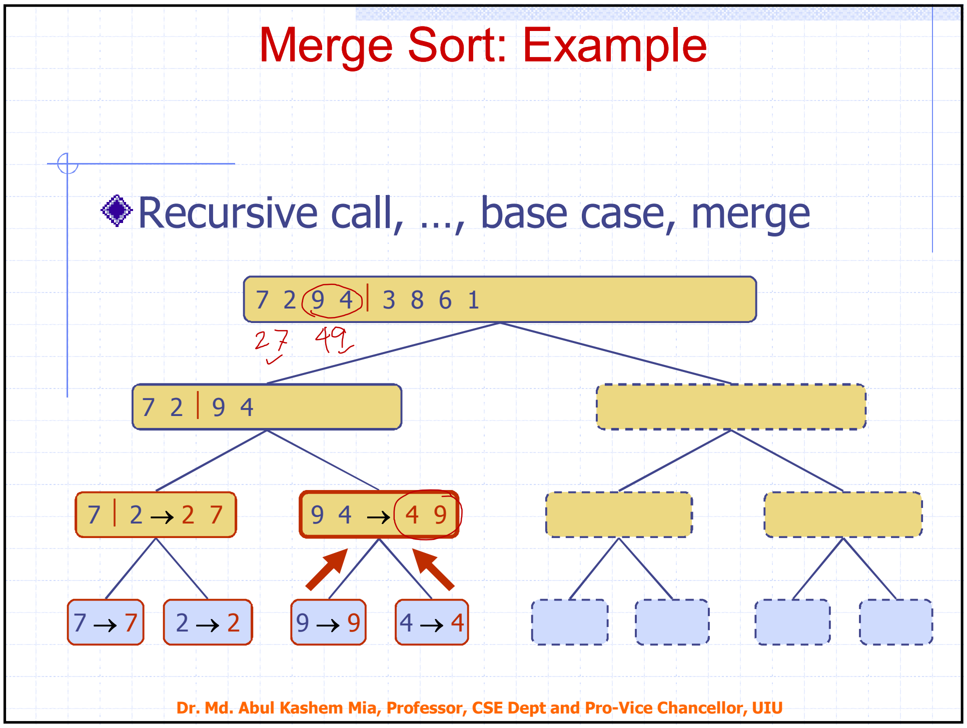
7 2 | 9 4

7 | 2 → 2 7

7 → 7

2 → 2

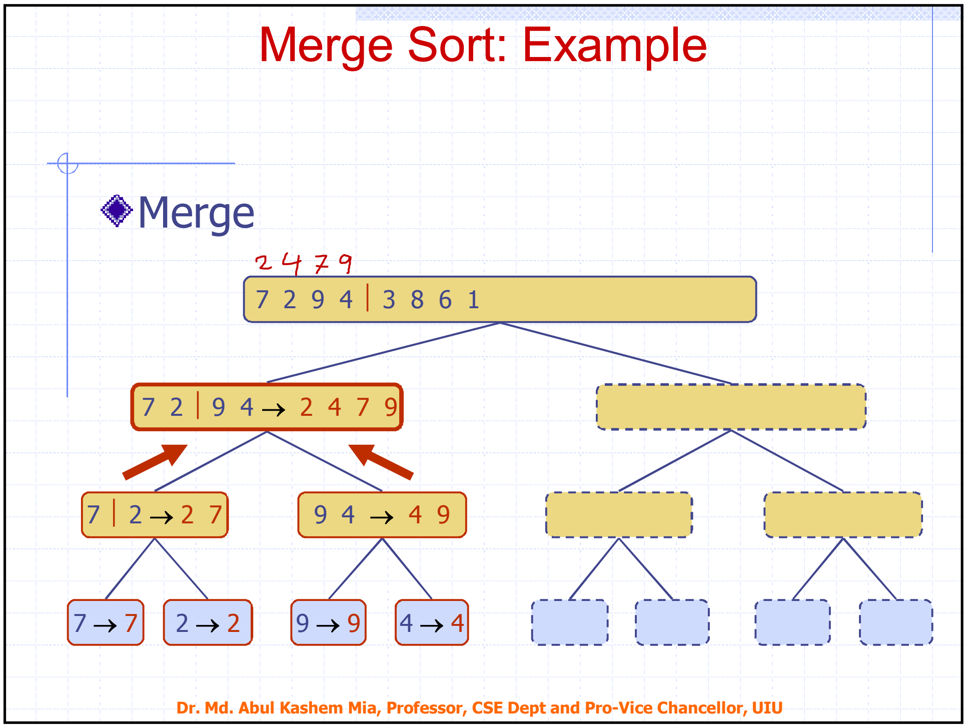
Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Merge Sort: Example

- Recursive call, ..., base case, merge

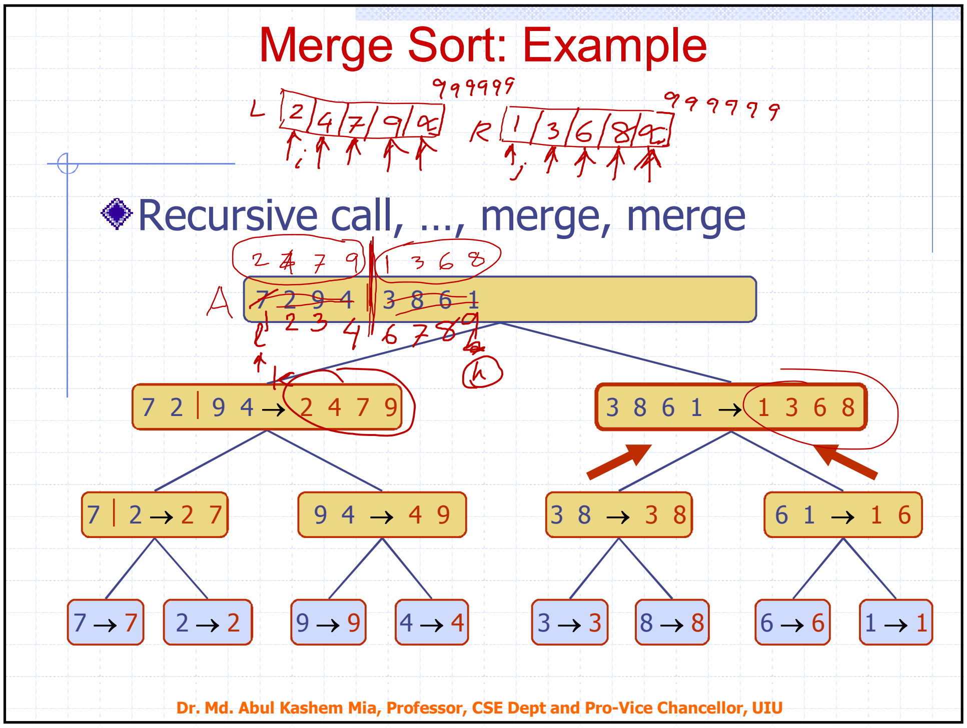
Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Merge Sort: Example

- Merge

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Merge Sort: Example

999999 999999

L 2 4 7 9 ∞ i R 1 3 6 8 ∞ j

- Recursive call, …, merge, merge

2 ~~4~~ 7 9 | 1 3 6 8

A ~~7 2 9 4~~ | ~~3 8 6 1~~

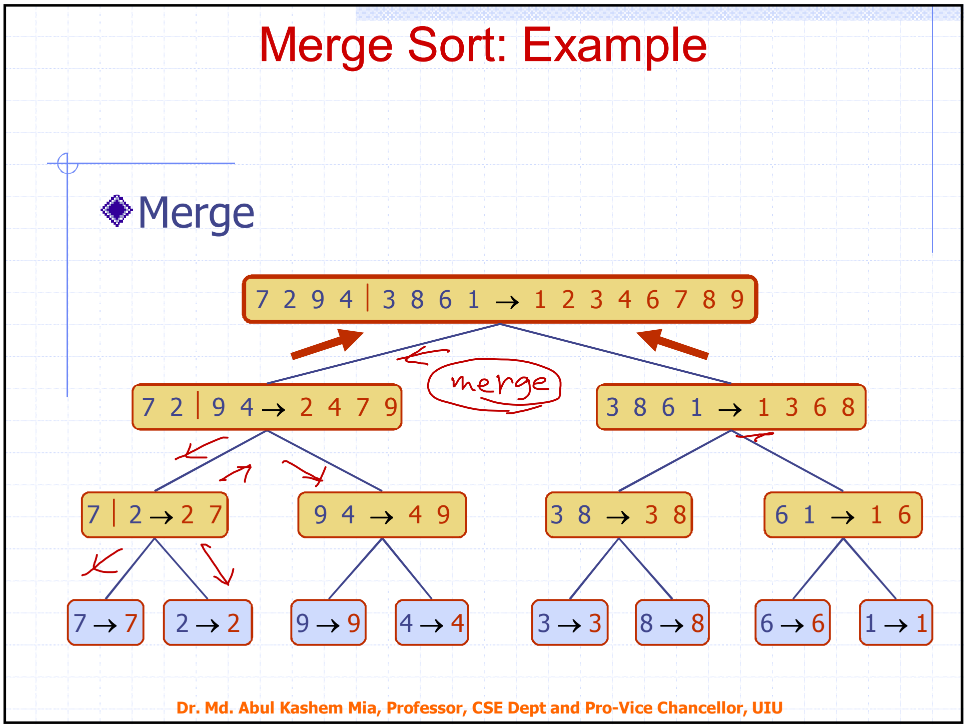
l 2 3 4 | 6 7 8 9 k h

7 2 | 9 4 → 2 4 7 9 3 8 6 1 → 1 3 6 8

7 | 2 → 2 7 9 4 → 4 9 3 8 → 3 8 6 1 → 1 6

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

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Merge Sort: Example

- Merge

7 2 9 4 | 3 8 6 1 → 1 2 3 4 6 7 8 9

merge

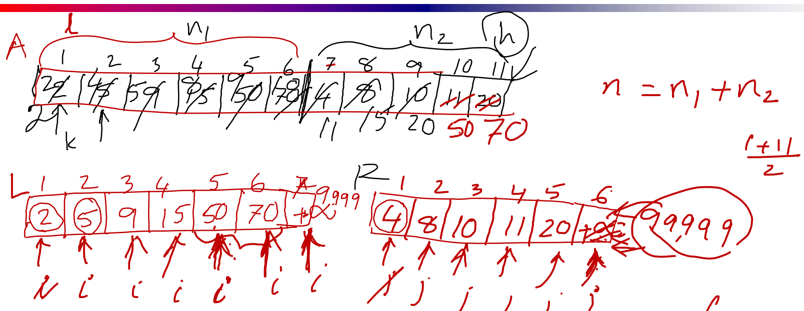
7 2 | 9 4 → 2 4 7 9 3 8 6 1 → 1 3 6 8

7 | 2 → 2 7 9 4 → 4 9 3 8 → 3 8 6 1 → 1 6

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

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Merge: Example



for k = l ; k≤

+∞

```
if L[i] < R[j]
        A[ ] ← L[i]
        i++
else
        A[ ] ← R[j]
        j++
```

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Merge Sort: Running Time

The recurrence for the worst-case running time T(n) is

$$T(n) \le \begin{cases} O(1) & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{O(1)}_{\text{dividing}} + \underbrace{O(n)}_{\text{merging}} & \text{otherwise} \end{cases}$$

equivalently

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{O(n)}_{\text{dividing + merging}} & \text{otherwise} \end{cases}$$

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Merge Sort: Running Time

The recurrence for the worst-case running time T(n) is

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ 2T(n/2) + O(n) & \text{if } n > 1 \end{cases}$$

equivalently

$$T(n) = \begin{cases} b & \text{if } n = 1 \\ 2T(n/2) + bn & \text{if } n > 1 \end{cases}$$

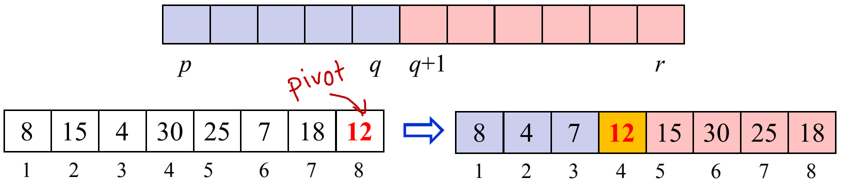
By solving the recurrence, we get

$T(n) = O(n \log n)$ ← both best case and worst case

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Quick Sort: Algorithm

- Another divide-and-conquer algorithm
  - The array A[$p..r$] is *partitioned* into two non-empty subarrays A[$p..q$] and A[q+1..r]



8	15	4	30	25	7	18	12
1	2	3	4	5	6	7	8

⇨

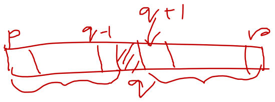
8	4	7	12	15	30	25	18
1	2	3	4	5	6	7	8

    - Invariant: All elements in A[$p..q$] are less than all elements in A[q+1..r]
  - The subarrays are recursively sorted by calls to quicksort
  - Unlike merge sort, no combining step: two subarrays form an already-sorted array

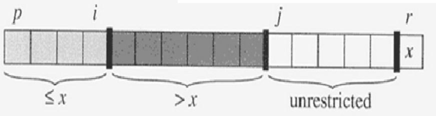
Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Quick Sort: Algorithm

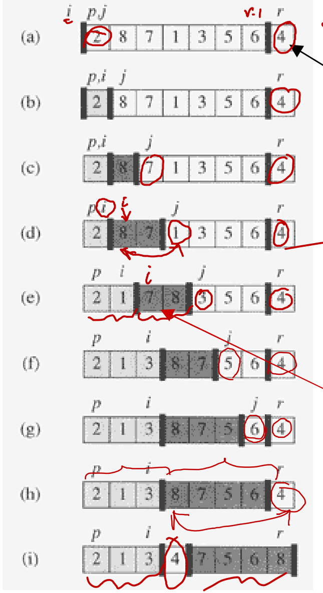
```
QUICKSORT($A, p, r$)
O(1)  if $p < r$
         then $q \leftarrow$ PARTITION($A, p, r$) ← O(n)
                QUICKSORT($A, p, q - 1$)
                QUICKSORT($A, q + 1, r$)
```

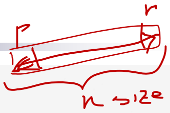


```
PARTITION($A, p, r$)
  $x \leftarrow A[r]$
  $i \leftarrow p - 1$
  for $j \leftarrow p$ to $r - 1$
         do if $A[j] \le x$
               then $i \leftarrow i + 1$
                      exchange $A[i] \leftrightarrow A[j]$
  exchange $A[i + 1] \leftrightarrow A[r]$
  return $i + 1$
```



Quick Sort: Example

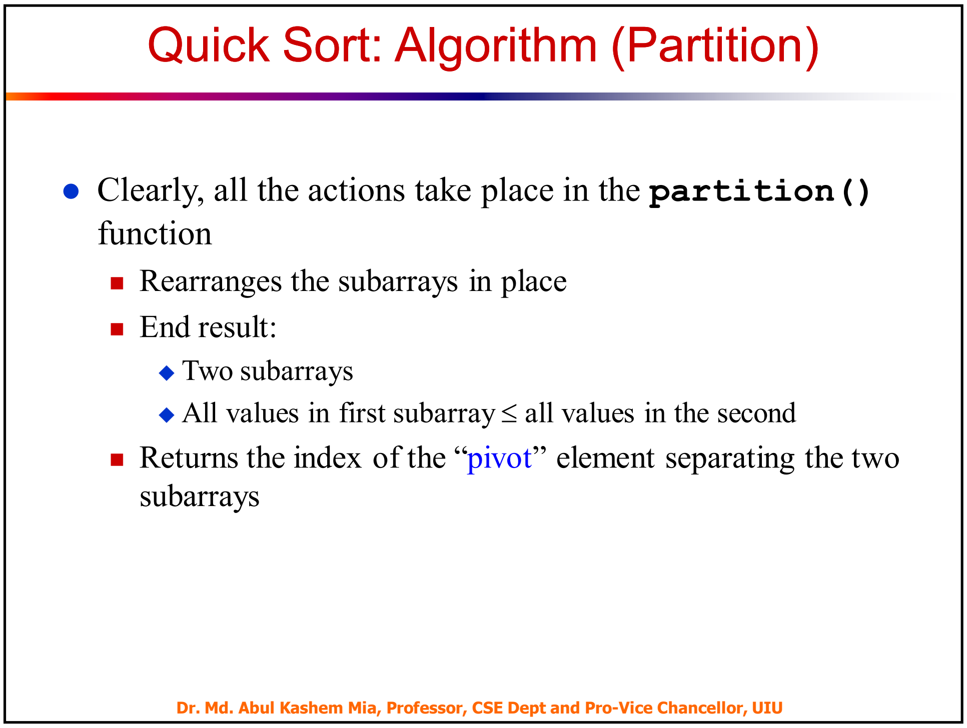




```
PARTITION($A, p, r$)
O(1) { $x \leftarrow A[r]$
       { $i \leftarrow p - 1$
O(n) { for $j \leftarrow p$ to $r - 1$
       {      do if $A[j] \le x$ ← pivot
       {            then $i \leftarrow i + 1$
       {                   exchange $A[i] \leftrightarrow A[j]$
O(1) { exchange $A[i + 1] \leftrightarrow A[r]$
       { return $i + 1$
```

Total time = $O(1) + O(n) + O(1) = O(n)$

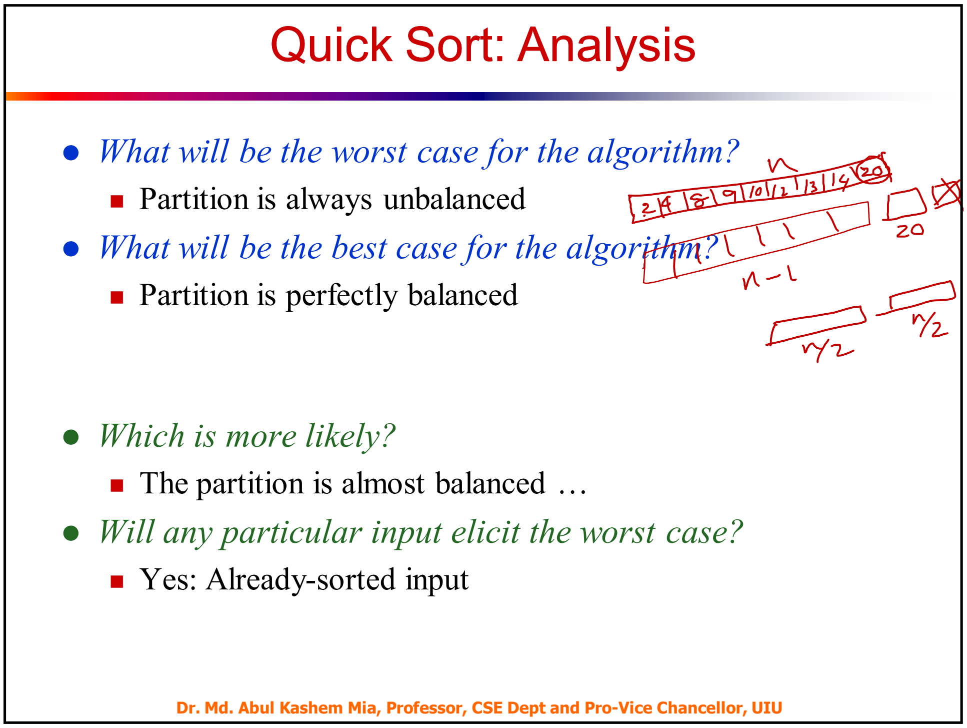
From i +1 to j is a window of elements > A[r]. The cursor j moves right one step at a time.
If the cursor j "discovers" an element ≤ A[r], then this element is swapped with the front element of the window, effectively moving the window right one step; if it discovers an element > A[r], then the window simply becomes longer one unit.



Quick Sort: Algorithm (Partition)

- Clearly, all the actions take place in the **`partition()`** function
  - Rearranges the subarrays in place
  - End result:
    - Two subarrays
    - All values in first subarray ≤ all values in the second
  - Returns the index of the "pivot" element separating the two subarrays

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

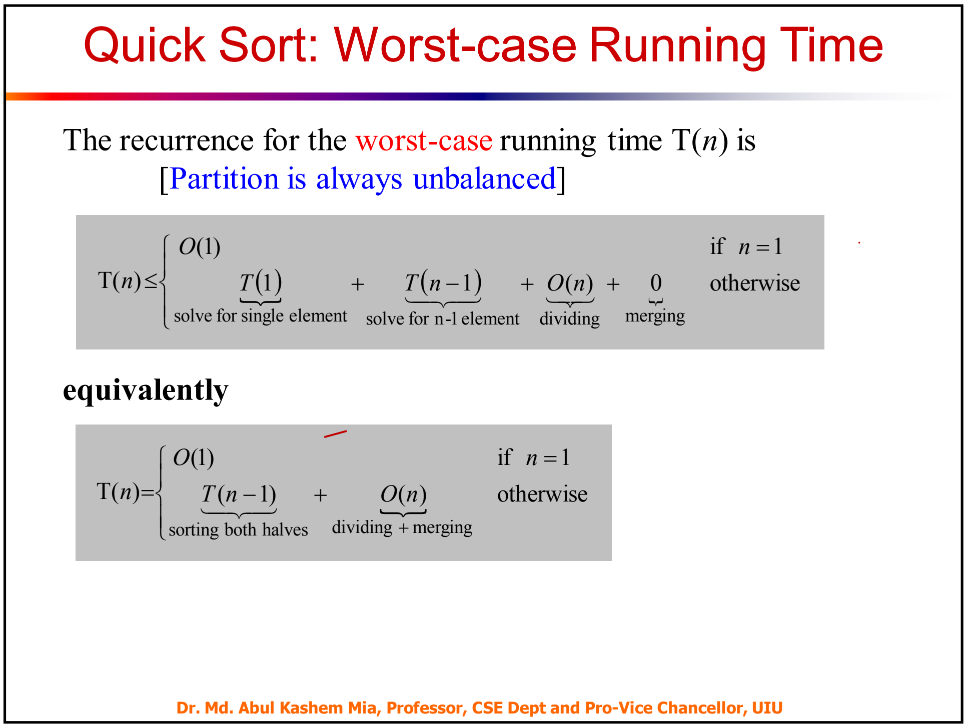


Quick Sort: Analysis

- *What will be the worst case for the algorithm?*
  - Partition is always unbalanced
- *What will be the best case for the algorithm?*
  - Partition is perfectly balanced
- *Which is more likely?*
  - The partition is almost balanced …
- *Will any particular input elicit the worst case?*
  - Yes: Already-sorted input

n
2 4 8 9 10 12 13 14 20
20
n-1
n/2
n/2

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Quick Sort: Worst-case Running Time

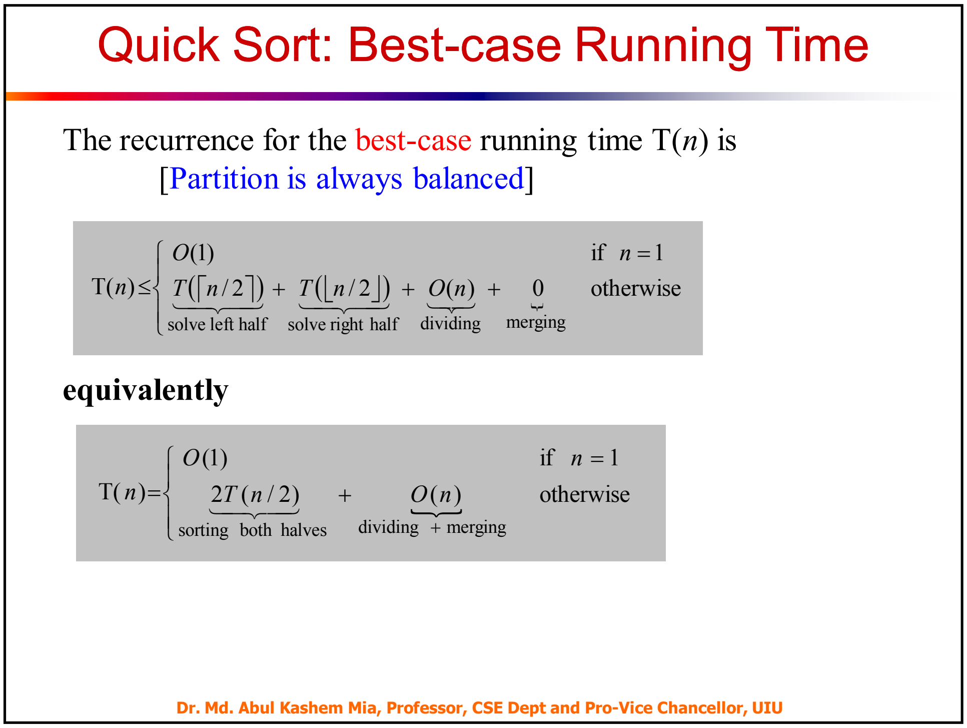
The recurrence for the worst-case running time T(*n*) is
[Partition is always unbalanced]

$$\mathrm{T}(n) \le \begin{cases} O(1) & \text{if } n = 1 \\ \underbrace{T(1)}_{\text{solve for single element}} + \underbrace{T(n-1)}_{\text{solve for n-1 element}} + \underbrace{O(n)}_{\text{dividing}} + \underbrace{0}_{\text{merging}} & \text{otherwise} \end{cases}$$

equivalently

$$\mathrm{T}(n) = \begin{cases} O(1) & \text{if } n = 1 \\ \underbrace{T(n-1)}_{\text{sorting both halves}} + \underbrace{O(n)}_{\text{dividing + merging}} & \text{otherwise} \end{cases}$$

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Quick Sort: Best-case Running Time

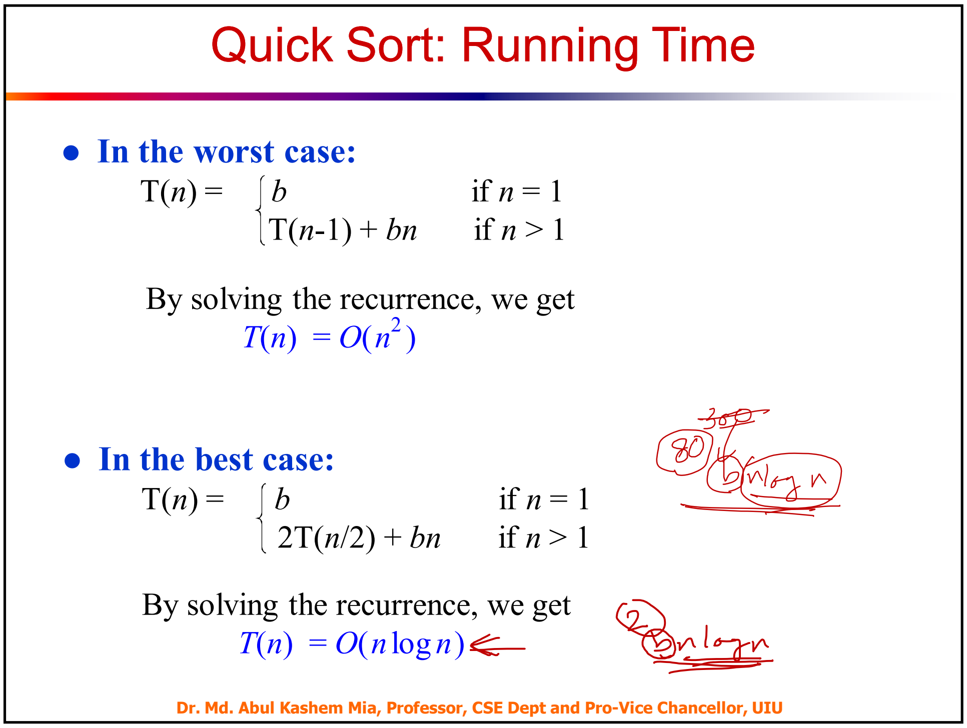
The recurrence for the best-case running time T(*n*) is
[Partition is always balanced]

$$\mathrm{T}(n) \le \begin{cases} O(1) & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{O(n)}_{\text{dividing}} + \underbrace{0}_{\text{merging}} & \text{otherwise} \end{cases}$$

equivalently

$$\mathrm{T}(n) = \begin{cases} O(1) & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{O(n)}_{\text{dividing + merging}} & \text{otherwise} \end{cases}$$

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Quick Sort: Running Time

- **In the worst case:**

$$T(n) = \begin{cases} b & \text{if } n = 1 \\ T(n-1) + bn & \text{if } n > 1 \end{cases}$$

By solving the recurrence, we get

$$T(n) = O(n^2)$$

- **In the best case:**

$$T(n) = \begin{cases} b & \text{if } n = 1 \\ 2T(n/2) + bn & \text{if } n > 1 \end{cases}$$

By solving the recurrence, we get

$$T(n) = O(n \log n) \Leftarrow$$

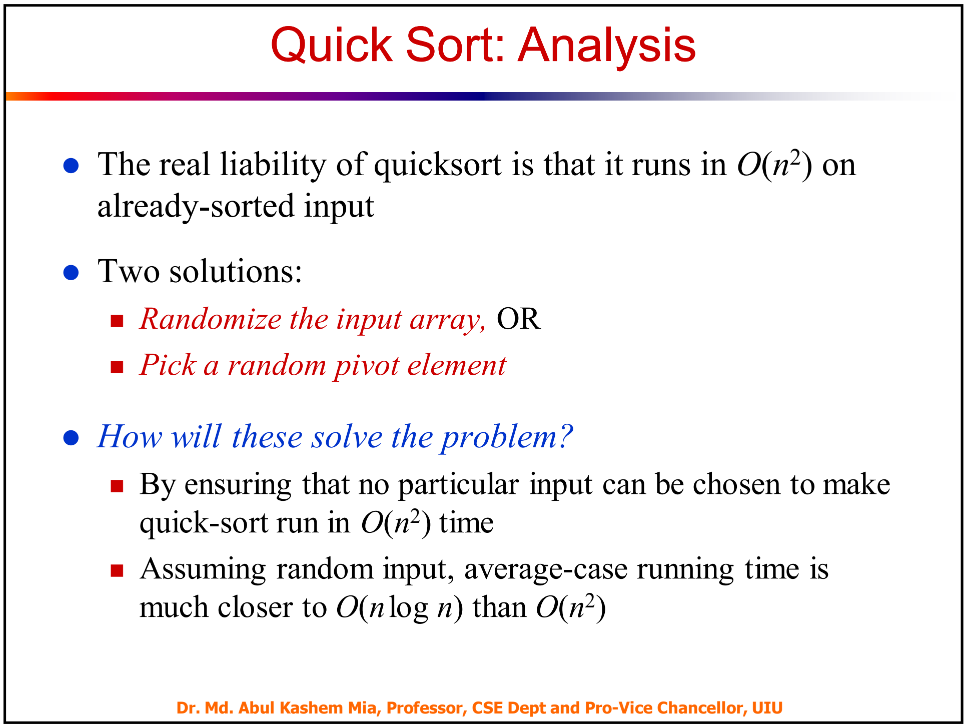
~~300~~

80

b n log n

2 b n log n

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU



Quick Sort: Analysis

- The real liability of quicksort is that it runs in $O(n^2)$ on already-sorted input
- Two solutions:
  - *Randomize the input array,* OR
  - *Pick a random pivot element*
- *How will these solve the problem?*
  - By ensuring that no particular input can be chosen to make quick-sort run in $O(n^2)$ time
  - Assuming random input, average-case running time is much closer to $O(n \log n)$ than $O(n^2)$

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU