



# SLIIT

*Discover Your Future*

# IT2070 – Data Structures and Algorithms

## Lecture 07

### Introduction to Divider and Conquer Method

**U. U. Samantha Rajapaksha**

M.Sc.in IT, B.Sc.(Engineering) University of Moratuwa

Senior Lecturer SLIIT  
[Samantha.r@slit.lk](mailto:Samantha.r@slit.lk)

use this method to  
sort an array

# Today's Lecture

- Divide and Conquer
  - Applications
    - Quick Sort
    - Merge Sort
  - Analysis
- these are the two Algorithms we use divide and conquer with*

# Divide and Conquer

Divide: Break the problem into sub problem recursively.

Conquer: Solve each sub problems.

Combine: All the solutions of sub problems are combined to get the solution of the original problem.

# Applications

- Quick Sort
  - More work on divide phase.
  - Less work for others.
- Merge Sort
  - Vice versa of Quick sort.

## Quick Sort (contd.)

**3 steps** 1) Divide 2) Conquer 3) Combine

**Divide:** Partition (rearrange) the array  $A[p..r]$  into two (possibly empty) sub arrays  $A[p..q - 1]$  and  $A[q + 1..r]$

- Each element of  $A[p..q - 1]$  is less than or equal to  $A[q]$
- Each element of  $A[q + 1..r]$  is greater than or equal to  $A[q]$ .
- Compute the index  $q$  as part of this partitioning procedure.

**Conquer:** Sort the two subarrays  $A[p..q - 1]$  and  $A[q + 1..r]$  by recursive calls to quicksort.

**Combine:** Since the sub arrays are sorted in place, no work is needed to combine them: the entire array  $A[p..r]$  is now sorted.

there are QuickSort's lines

## Quick Sort procedure

for one partition call the last element find its place  
 Input: Unsorted Array ( $A, p, r$ )  
 Output: Sorted sub array  $A(1..r)$

**QUICKSORT** ( $A, p, r$ )

1 if  $p < r$

2  $q = \text{PARTITION}(A, p, r)$

Sri Lanka Institute of Information

Technology

3 **QUICKSORT** ( $A, p, q-1$ )

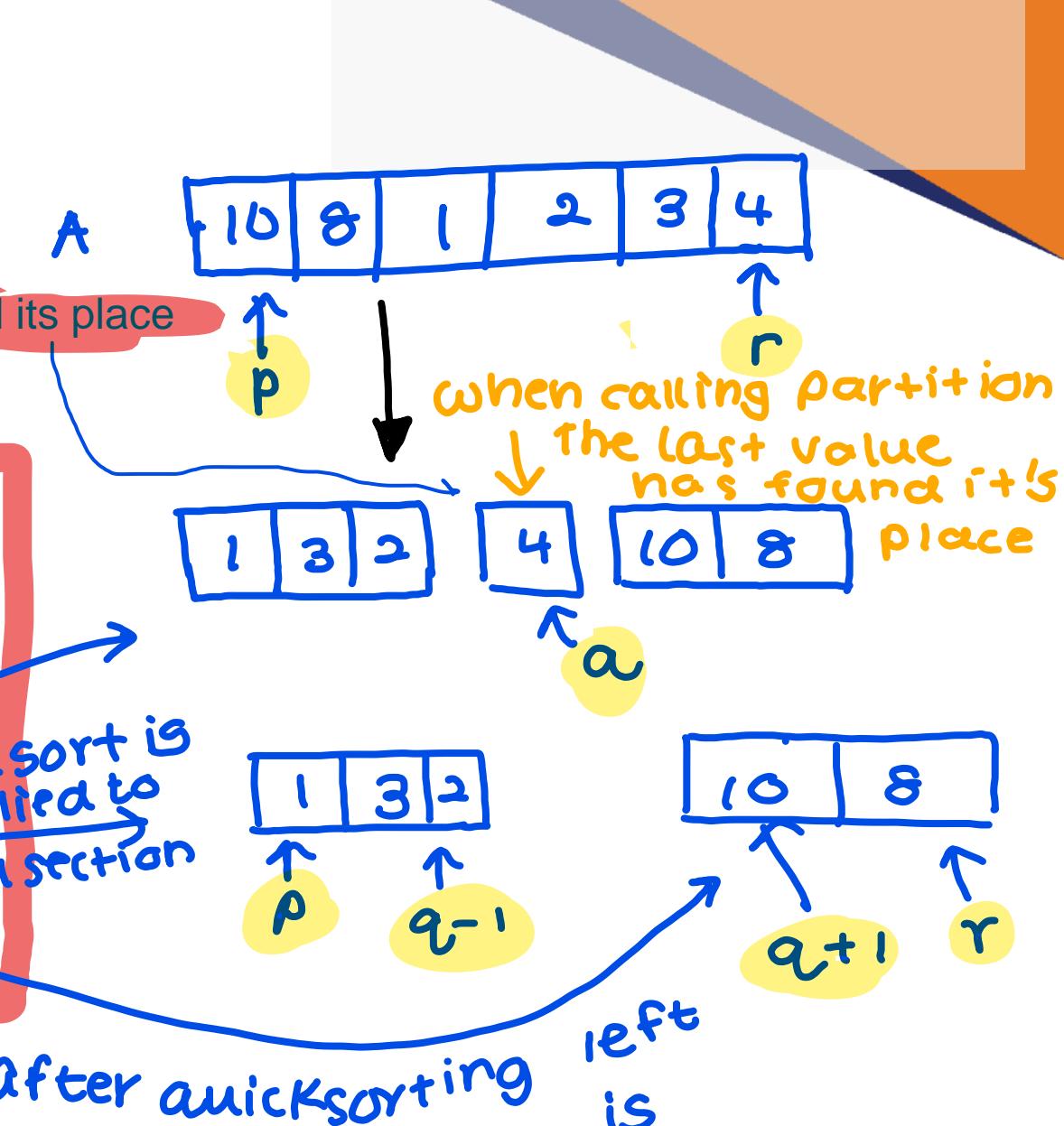
Quicksort from

pto  $q-1$  4 **QUICKSORT** ( $A, q+1, r$ )

6 quicksort is now applied to the left section

To sort an entire array  $A$ , the initial call is

**QUICKSORT**( $A, 1, A.length$ )



after a quicksorting  
side the right section  
Quicksorted

# Partition Algorithm

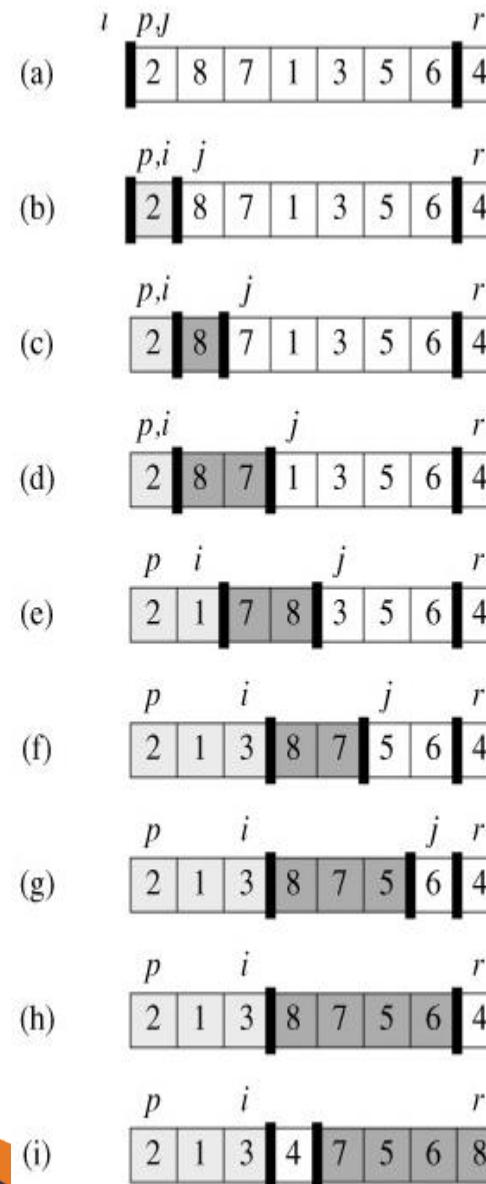
→ analyzing the  
partition function  
in above code

## PARTITION( $A, p, r$ )

```
1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6          exchange  $A[i]$  with  $A[j]$ 
7          exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 
```

The key to the algorithm is the PARTITION procedure, which rearranges the sub array  $A [p..r]$  in place.

## Operation of PARTITION on an 8-element array.



(a) The initial array

(b) The value 2 is "swapped with itself" and put in the partition of smaller values.

(c)-(d) The values 8 and 7 are added to the partition of larger values.

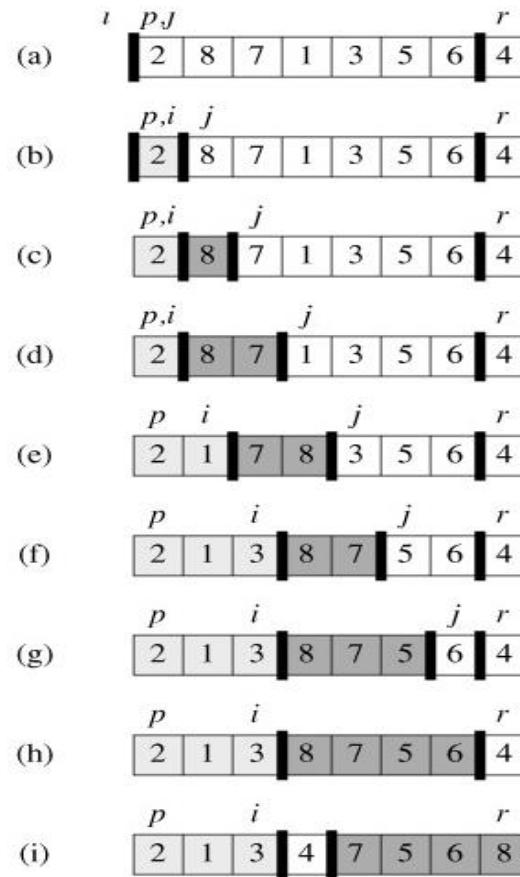
(e) The values 1 and 8 are swapped, and the smaller partition Grows.

(f) The values 3 and 7 are swapped, and the smaller partition grows.

(g)-(h) The larger partition grows to include 5 and 6 and the loop terminates.

(i) Pivot element is swapped so that it lies between the two partitions.

## Operation of PARTITION on an 8-element array.



first partition with values  $\leq$  pivot

pivot is  $r$   
here

white element - pivot.

second partition with values  $>$  pivot

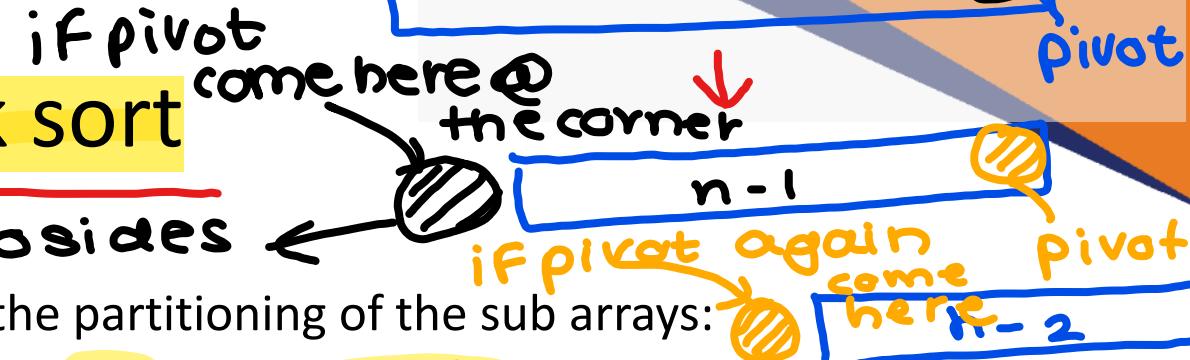
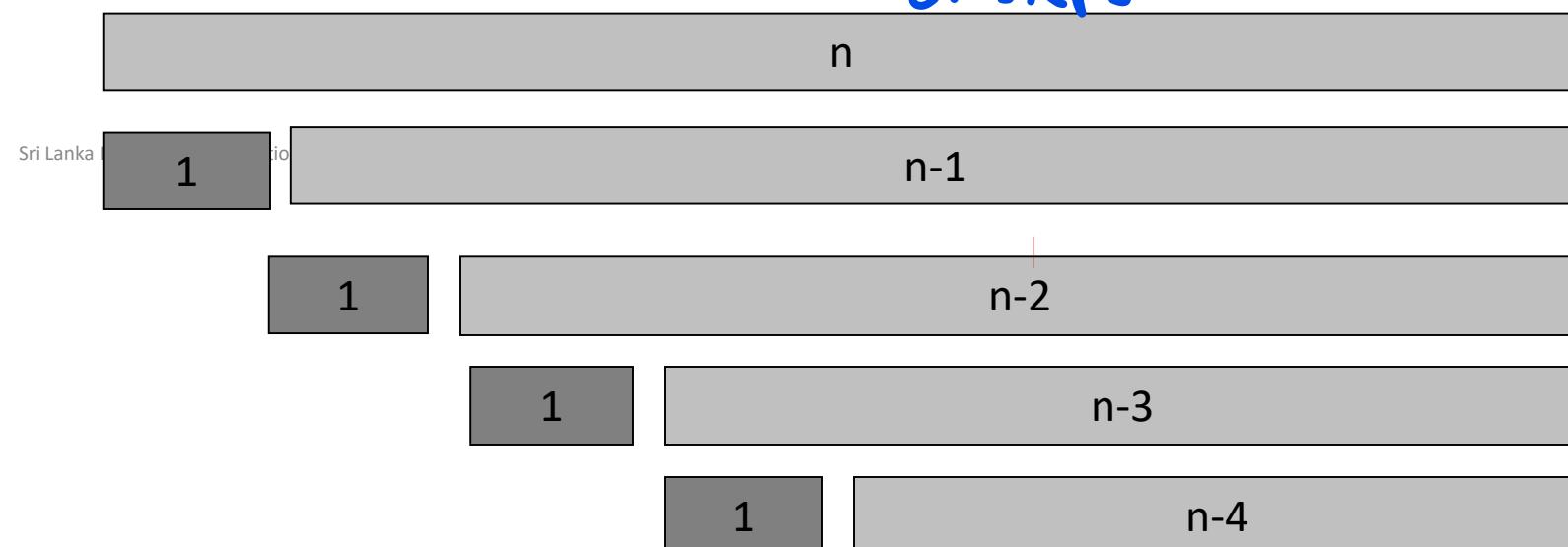
## Analysis of Quick sort

no two sides

The running time of quick sort depends on the partitioning of the sub arrays:

### (a) Worst case partitioning (Unbalanced partitioning)

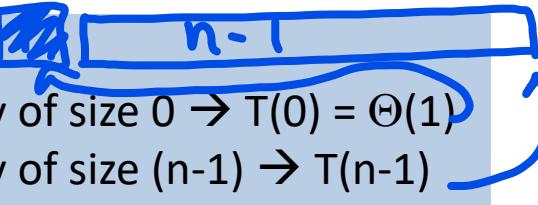
- Worst case occurs when the sub arrays are completely unbalanced. i.e. 0 elements in one sub array and  $n - 1$  elements in the other sub array
- when you try to sort our array we have to spend maximum number of steps*



# Analysis of Quick sort

## Worst case partitioning (Repeated Substituted method)

- Partitioning  $\rightarrow \Theta(n)$
- Recursive call on an array of size 0  $\rightarrow T(0) = \Theta(1)$
- Recursive call on an array of size  $(n-1) \rightarrow T(n-1)$



Therefore Recurrence Equation is

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= \underbrace{T(n-1)}_{T(n-2) + \Theta(n-1) + \Theta(n)} + \Theta(n) \end{aligned}$$

*don't have anything  $\therefore$  ignore it*

Sri Lanka Institute of Information  
Technology.

Recurrence  
equation for  
worst case

$$1+2+3+4+\dots+n$$

when you add a series  
like this

$$= \frac{n(n+1)}{2}$$

at the  
end we get to a point like this

$$= \sum_{k=1}^n (\Theta(k)) = \Theta \sum_{k=1}^n k = \Theta(n^2)$$

Worst case Running Time is  $\Theta(n^2)$

$\Theta(n^2) \leftarrow \text{Big O} //$

Quicksort( $A, p, r$ )

if  $p < r$

$q = \text{PARTITION}(A, p, r)$

Quicksort( $A, p, q-1$ )

Quicksort( $A, q+1, r$ )

$$T(n) = T(n-1) + \Theta(n)$$

$$T(n-1) = T(n-2) + \Theta(n-1)$$

$$T(n-2) = T(n-3) + \Theta(n-2)$$

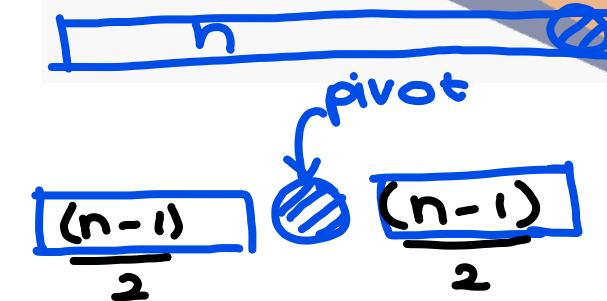
$$T(1) = T(0) + \Theta(1)$$

# Analysis of Quick sort

## (b) Best case partitioning

Best case occurs when PARTITION produces two sub arrays , one is of size  $(n-1)/2$  and the other is of size  $(n-1)/2$ . In this case, quicksort runs much faster.

Best case is  
when pivot  
comes to middle



instead of  
 $\frac{(n-1)}{2}$  for simpli  
city

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

Recurrence equation is

Recurrence equation

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

# Analysis of Quick Sort (with recursion tree)

Best Case:

worst

$h^2$

Best ( $n \lg n$ )

we can quickly get our answer

No. of Comparisons

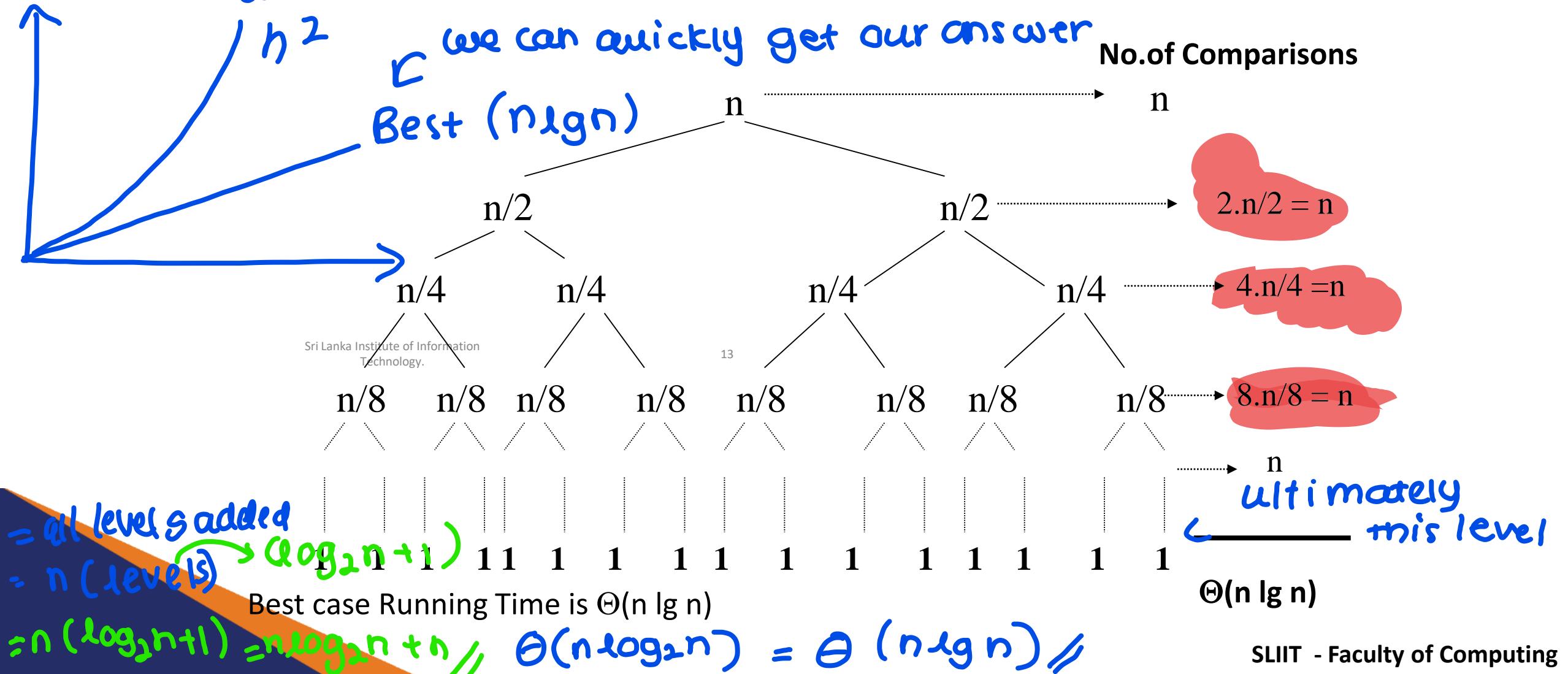
$n$

$$2 \cdot n/2 = n$$

$$4 \cdot n/4 = n$$

$$8 \cdot n/8 = n$$

ultimately  
this level



Like quickSort another algorithm



## Merge sort

Merge Sort is a sorting algorithm based on divide and conquer. Its worst-case running time has a lower order of growth than insertion sort.

The merge sort algorithm closely follows the divide-and-conquer paradigm. Intuitively, it operates as follows.

- **Divide:** Divide the  $n$ -element<sup>14</sup> sequence to be sorted into two subsequences of  $n/2$  elements each.
- **Conquer:** Sort the two subsequences recursively using merge sort.
- **Combine:** Merge the two sorted subsequences to produce the sorted answer.

# Merge sort

**Divide** by splitting into two subarrays  $A[p \dots q]$  and  $A[q + 1 \dots r]$ , where  $q$  is the halfway point of  $A[p \dots r]$ .

**Conquer** by recursively sorting the two subarrays  $A[p \dots q]$  and  $A[q + 1 \dots r]$ .

**Combine** by merging the two sorted subarrays  $A[p \dots q]$  and  $A[q + 1 \dots r]$  to produce a single sorted subarray  $A[p \dots r]$ .

To accomplish this step, we'll define a procedure  $\text{MERGE}(A, p, q, r)$ .

## Merge sort procedure

**Input :** A an array in the range 1 to n.  
**Output :** Sorted array A.

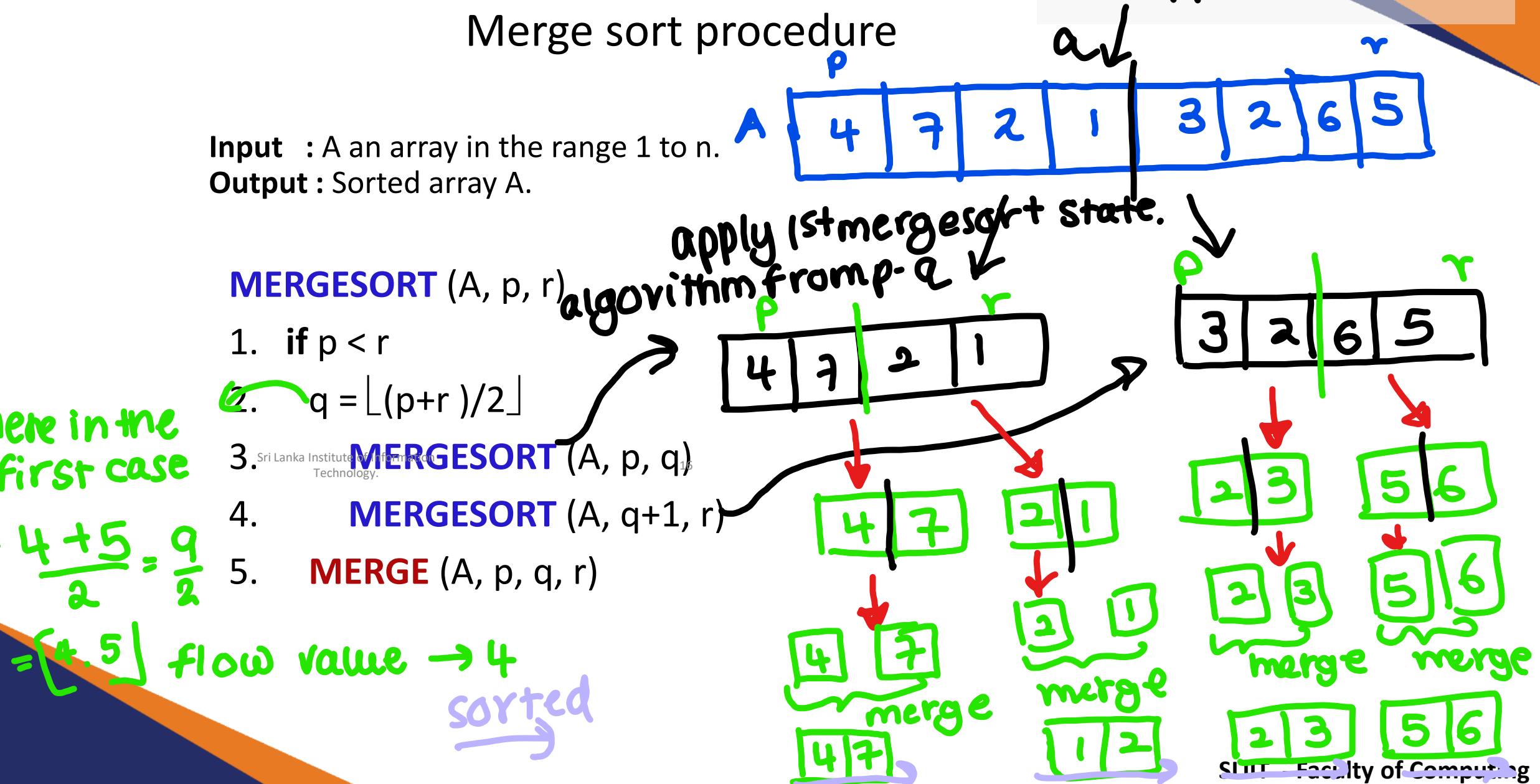
**MERGESORT** (A, p, r)

1. if  $p < r$
2.  $q = \lfloor (p+r)/2 \rfloor$
3. **MERGESORT** (A, p, q)
4. **MERGESORT** (A, q+1, r)
5. **MERGE** (A, p, q, r)

here in the  
first case

$$= \frac{4+5}{2} = \frac{9}{2}$$

$\{4, 5\}$  flow value  $\rightarrow 4$   
sorted



# Merge procedure

## the code for algorithm

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

1     $n_1 = q - p + 1$

2     $n_2 = r - q$

3    create arrays  $L[1.. n_1 + 1]$  and  $R[1.. n_2 + 1]$

4    for  $i = 1$  to  $n_1$

5          $L[i] = A[p + i - 1]$

6    for  $j = 1$  to  $n_2$

7          $R[j] = A[q + j]$

8    {      $L[n_1 + 1] = \infty$

9    R[n\_2 + 1] =  $\infty$

10    i = 1

11    j = 1

12    for  $k = p$  to  $r$

13         if  $L[i] \leq R[j]$

14              $A[k] = L[i]$

15              $i = i + 1$

16         else      $A[k] = R[j]$

17              $j = j + 1$

making  
last  
element in  
array d

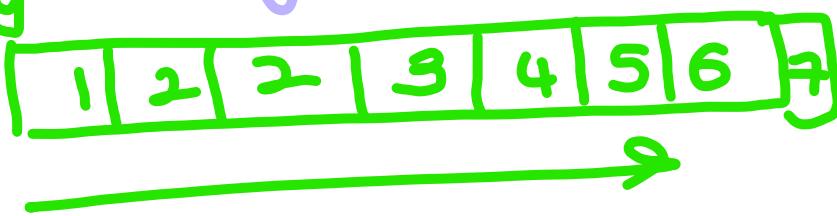
this  
merge  
happens  
according  
to the  
theory  
above

here we  
are sorting  
to left  
array

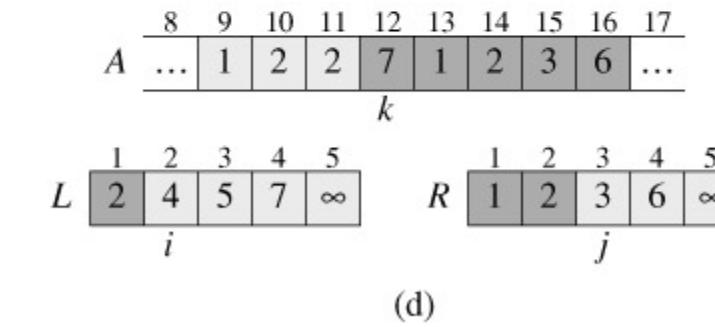
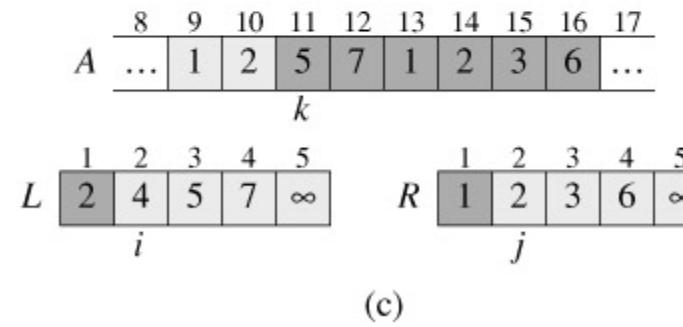
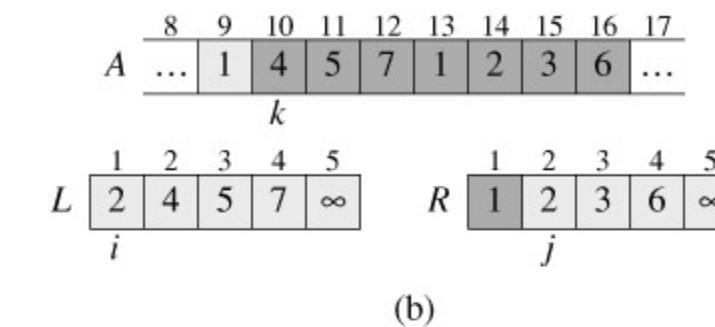
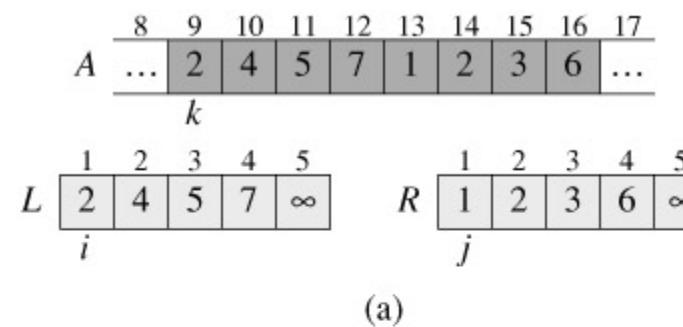
here we  
are  
sorting to right array

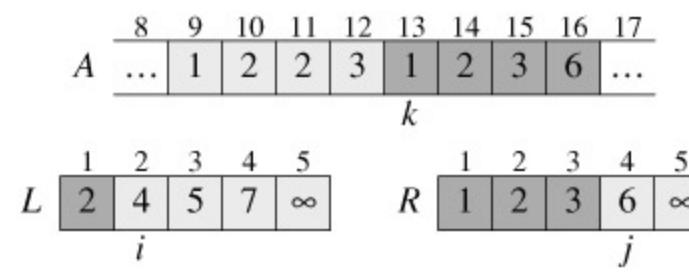
initializing i and j in the 2 arrays

comparison of numbers in the arrays  
moving indicator

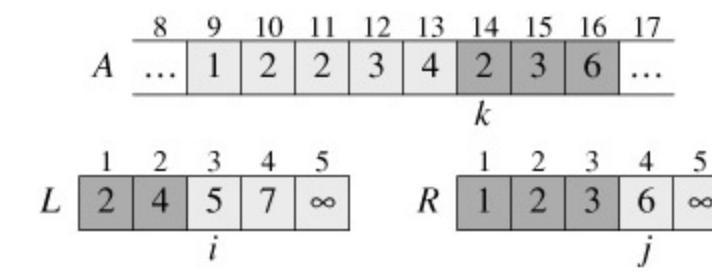


# Illustration when the subarray $A[9..16]$ contains the sequence $\langle 2, 4, 5, 7, 1, 2, 3, 6 \rangle$

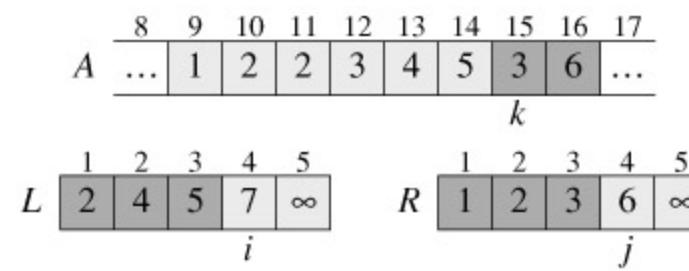




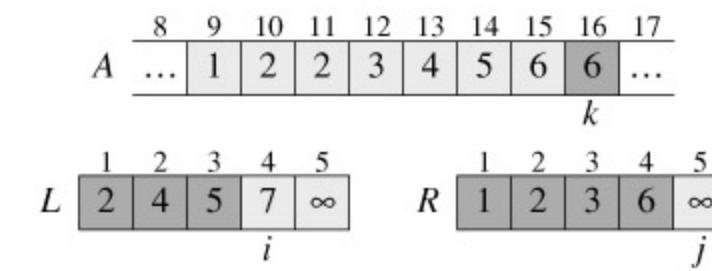
(e)



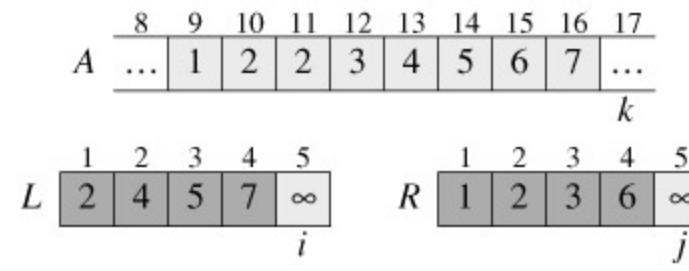
(f)



(g)



(h)



(i)

## Analysis of Merge Sort

**no best case scenario because the theory in this topic is the same**

- To find the middle of the sub array will take  $\Theta(1)$ .
- To recursively solve each sub problem will take  $2T(n/2)$ .
- To combine sub arrays will take  $\Theta(n)$ .

$$T(n) \boxed{n}$$

$$T\left(\frac{n}{2}\right) \boxed{n/2} \boxed{n/2}$$

**MERGESORT (A, P, R)**

$$\text{Therefore } T(n) = 2T(n/2) + \Theta(n) + \Theta(1)$$

Sri Lanka Institute of Information Technology.

We can ignore  $\Theta(1)$  term. <sup>20</sup>

recurrence  
equation

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

```

if(P < R)
    q1 ← q = L(P+R)/2
    T(n/2) ← MERGESORT (A, P, q)
    T(n/2) ← MERGESORT (A, q+1, R)
    Θ(n) ← MERGE ( A, P, q, R)
  
```

# Analysis of Merge Sort

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

$$2T(n/2) = 4T(n/4) + 2cn/2$$

$$4T(n/4) = 8T(n/8) + 4cn/4$$

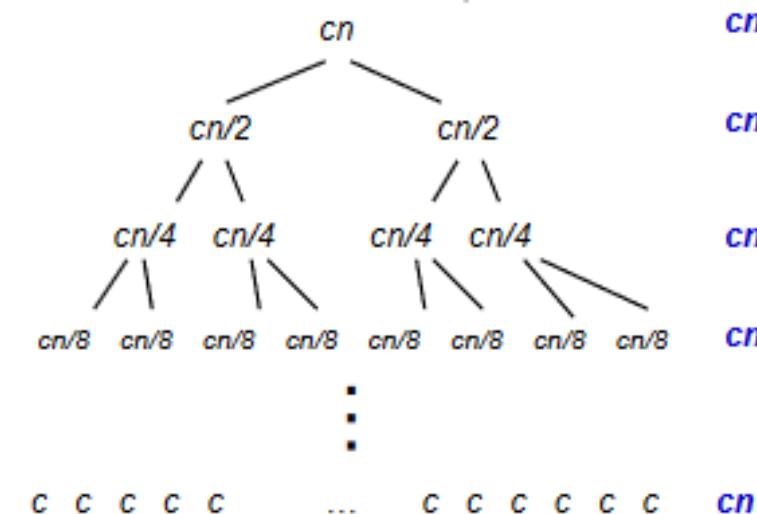
-  
-  
-  
-

$$T(2) = nT(1) + (n/2)c^*2$$

**add and cancel:**

$$T(n) = nT(1) + cn + cn + \dots + cn$$

$$= nT(1) + cn * \log_2 n = \Theta(n \log n)$$



# Summary.

- Divide and conquer method.
- Quicksort algorithm.
- Quicksort analysis.
- Mergesort algorithm.
- Mergesort analysis.