SLIIT ACADEMY

BSc (IT)

Year 2, Semester 1

Design and Analysis of Algorithms

Divide & Conquer Method

Anuruddha Abeysinghe

anuruddha.a@sliit.lk

Today's Lecture

- Divide and Conquer Method
- Applications
  - Quick Sort
  - Merge Sort

Divide and Conquer

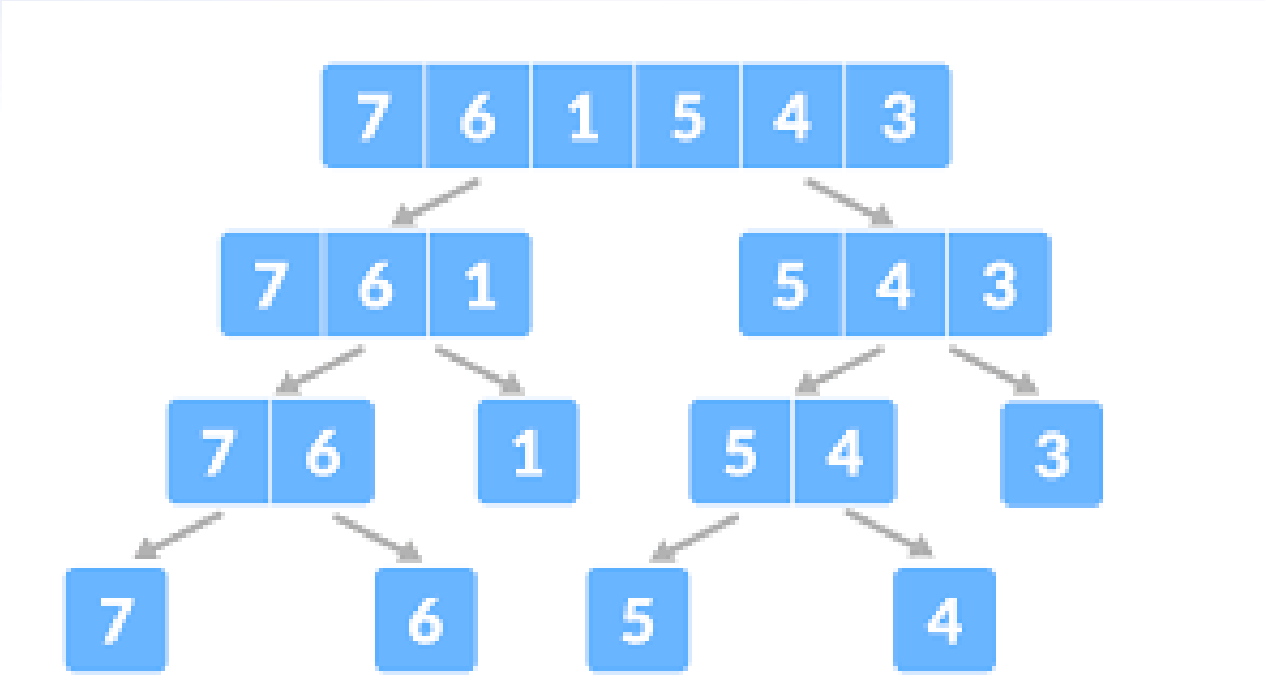
Divide: Break the problem into sub problems recursively.

Conquer: Solve each sub problems.

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

Example

- Finding maximum element for a given array.



Advantages

- Increase the speed.

  By running sub problems in parallel.

- Increase the Cache Performance.

Applications

- ***Quick Sort***

  More work on divide phase.

  Less work for others.

- ***Merge Sort***

  Vice versa of Quick sort.

Quick Sort

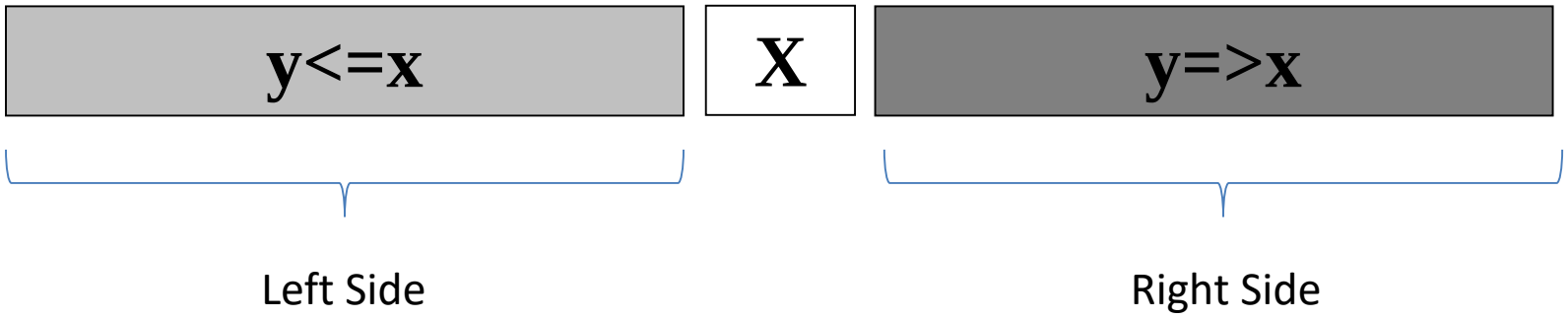
Divide: **Partition** (rearrange) the array A[p..r] into two (possibly empty) sub arrays A[p..q - 1] and A[q + 1..r] such that each element of A[p..q - 1] is less than or equal to A[q], which is, in turn, less than or equal to each element of A[q + 1..r]. Compute the index q as part of this partitioning 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.

Quick Sort

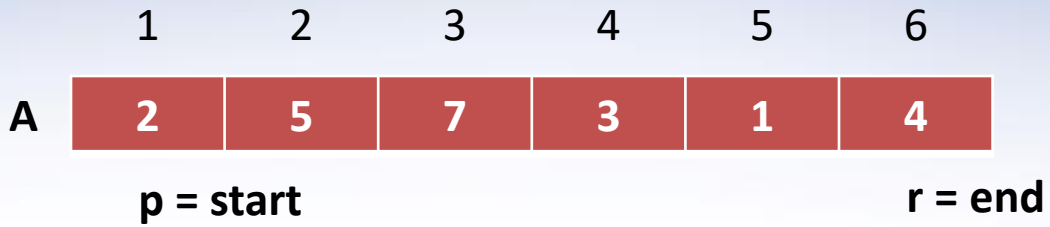
- Left side – Arrange the values as y<=x
- Middle(only one element)
- Right side - Arrange the values as y>=x



Quick Sort procedure

Input: Unsorted Array (A,p,r)

Output: Sorted sub array A(1..r)




```
Procedure QUICKSORT (A,p,r)
1 if p < r
2     then q ← PARTITION(A,p,r)
3          QUICKSORT (A,p,q-1)
4          QUICKSORT (A,q+1,r)
```

To sort an entire array A, the initial call is QUICKSORT(A, 1, length[A]).

Partition Algorithm

PARTITION(*A*, *p*, *r*)

```
1 x ← A[r]
2 i ← p - 1
3     for j ← p to r - 1
4         do if A[j] ≤ x
5                then i ← i + 1
6                     exchange A[i] ↔ A[j]
7    exchange A[i + 1] ↔ 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.

Quick Sort Algorithm

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

1 **if** p < r

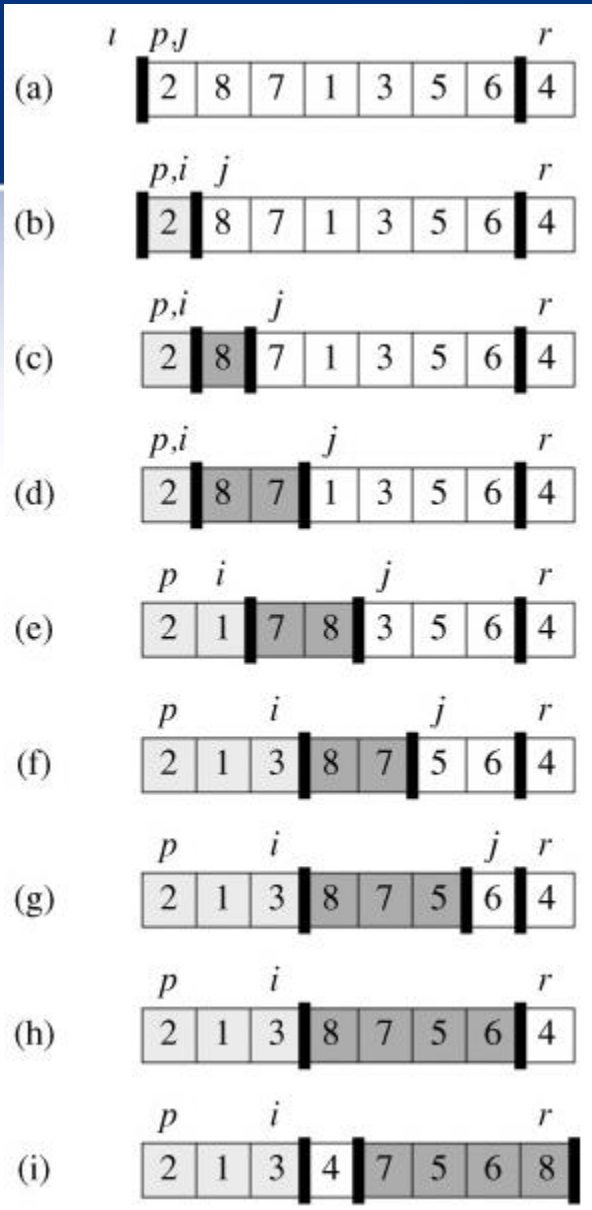
2 **then** q ← **PARTITION**(A,p,r)

```
PARTITION(A, p, r)

1 x ← A[r]
2 i ← p - 1
3     for j ← p to r - 1
4         do if A[j] ≤ x
5                then i ← i + 1
6                     exchange A[i] ↔ A[j]
7    exchange A[i + 1] ↔ A[r]
8  return i + 1
```

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

4 **QUICKSORT** (A,q+1,r)



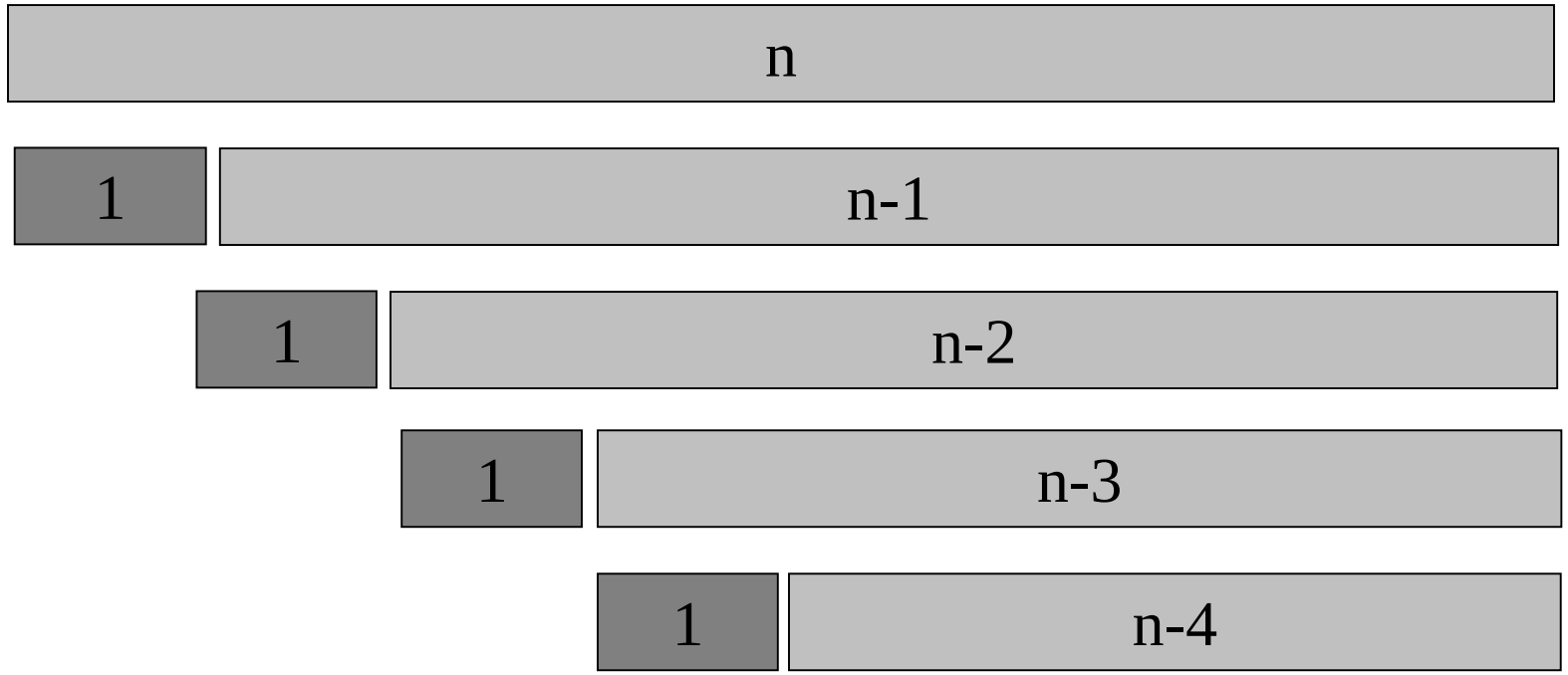
This shows the operation of PARTITION on an 8-element array. PARTITION always selects an element x = A[r] as a pivot element around which to partition the sub array A[p..r]. As the procedure runs, the array is partitioned into four (possibly empty) regions.

The operation of PARTITION on a sample array. Lightly shaded array elements are all in the first partition with values no greater than x. Heavily shaded elements are in the second partition with values greater than x. The unshaded elements have not yet been put in one of the first two partitions, and the final white element is the pivot. (a) The initial array and variable settings. None of the elements have been placed in either of the first two partitions. (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 8 are swapped, and the smaller partition grows. (g)-(h) The larger partition grows to include 5 and 6 and the loop terminates. (i) In lines 7-8, the pivot element is swapped so that it lies between the two partitions.

Analysis of Quick sort - Worst case partitioning(Unbalanced)

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

(a) Worst case partitioning(Unbalanced)



Analysis of Quick sort

- Occurs when the sub arrays are **completely unbalanced**.
- Have 0 elements in one sub array and $n - 1$ elements in the other sub array.
- Time taken for partitioning is $\Theta(n)$ and $T(0) = \Theta(1)$.

- Therefore Recurrence Equation is

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

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

Analysis of Quick sort (Repeated Substituted method.)

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

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

$$= \Theta \sum_{k=1}^{n} k$$

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

Analysis of Quick sort - Best case partitioning(balanced)

(b) Best case partitioning.

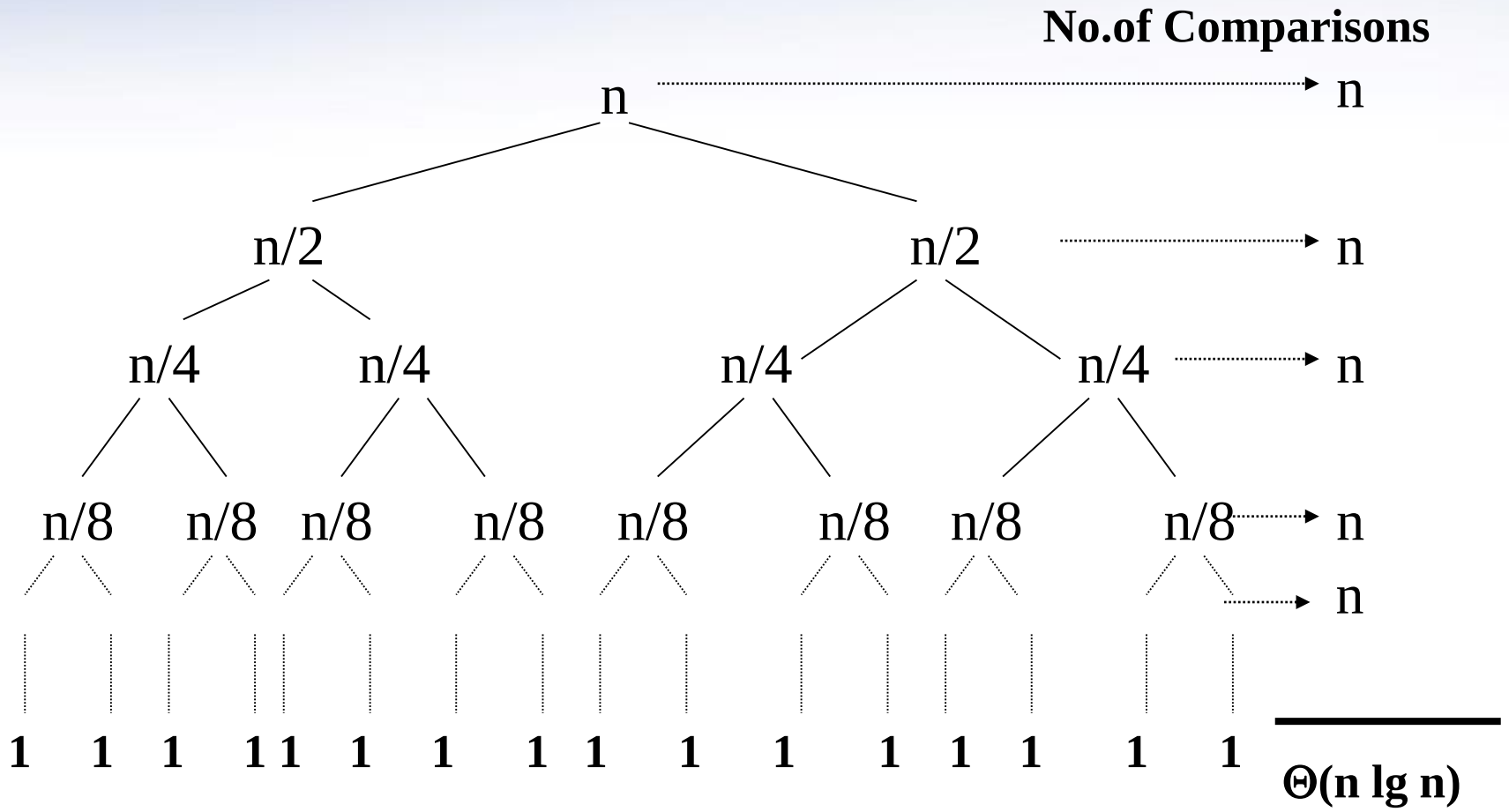
Produce two segments of size n/2.

Recurrence equation is

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

Analysis of Quick sort (with recursion tree)

Best Case:



Analysis of Quick sort (with Master theorem.)

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{\log_b a - \varepsilon}\right) \rightarrow f(n) < n^{\log_b a} \\ \Theta\left(n^{\log_b a} \lg n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \rightarrow f(n) = n^{\log_b a} \\ \Theta\left(f(n)\right) & f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right) \rightarrow f(n) < n^{\log_b a} \\ & \text{if } af(n/b) \le cf(n) \text{ for c} < 1 \text{ and large } n \end{cases}$$

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

$a = 2, b = 2$

$\rightarrow n^{\log_b a} = n^{\log_2 2} = n$

→ CASE 2

→ $T(n) = \Theta(n \lg n)$.

Merge sort

- The ***merge sort*** algorithm closely follows the divide-and-conquer paradigm. Intuitively, it operates as follows.
- **Divide:** Divide the n-element 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

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

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

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

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

Merge sort procedure

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

```
Procedure MERGESORT (A,p,r)
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)
```

Merge procedure

```
MERGE(A, p, q, r)
1     n1 ← q - p + 1
2     n2 ← r - q
3     create arrays L[1.. n1 + 1] and R[1.. n2 + 1]
4     for i ← 1 to n1
5           do L[i] ← A[p + i - 1]
6     for j ← 1 to n2
7           do 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          do if L[i] ≤ R[j]
14                then A[k] ← L[i]
15                     i ← i + 1
16                else A[k] ← R[j]
17                     j ← j + 1
```

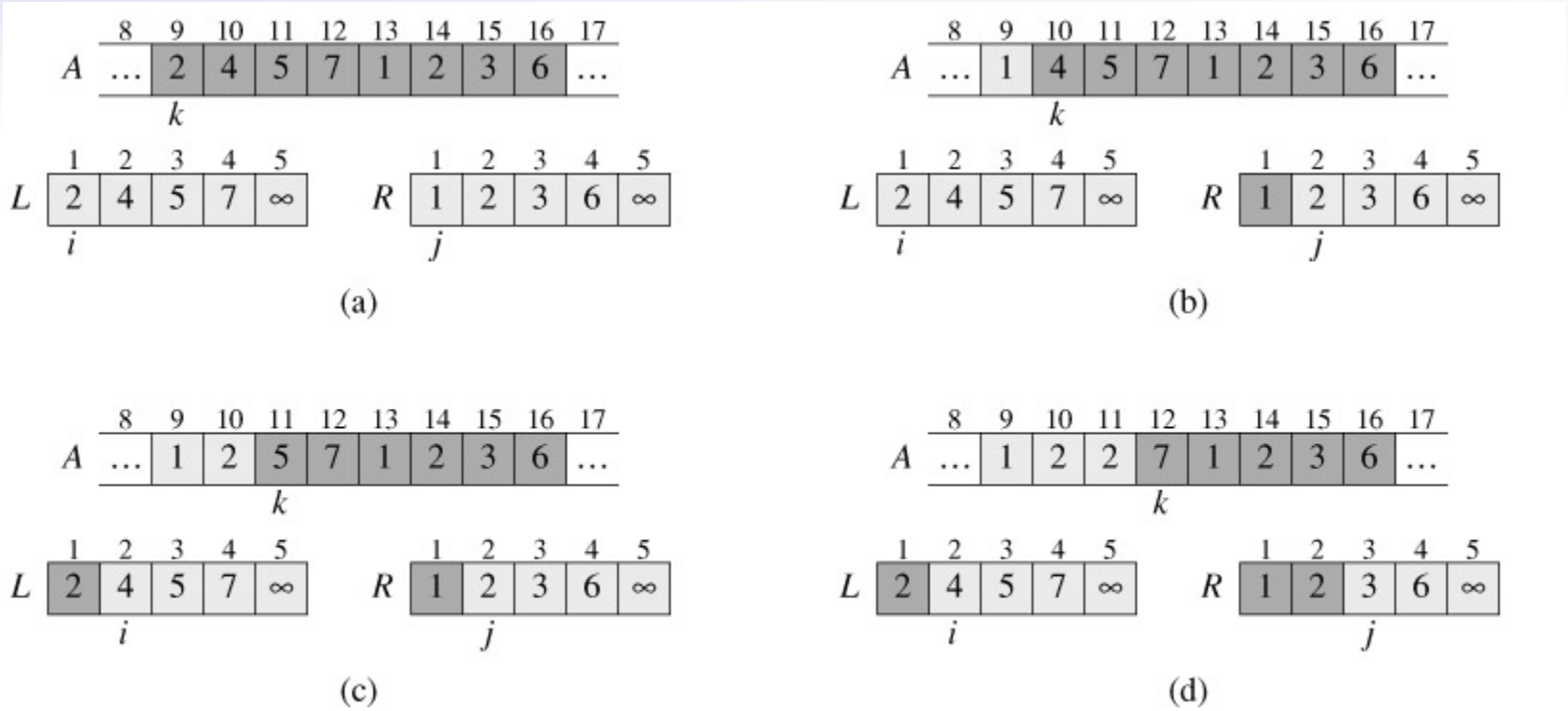
Merge procedure

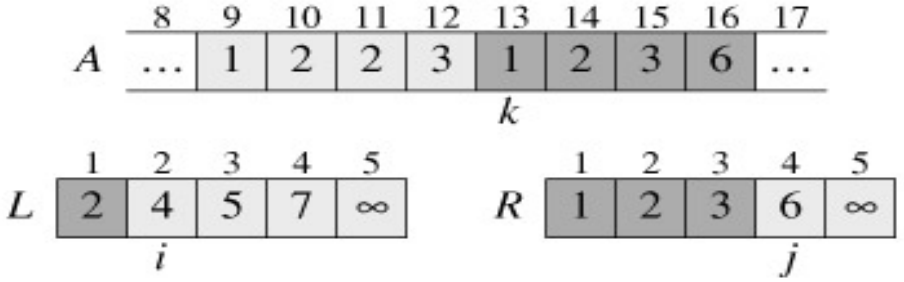
Procedure **MERGESORT** (A,p,r)

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)

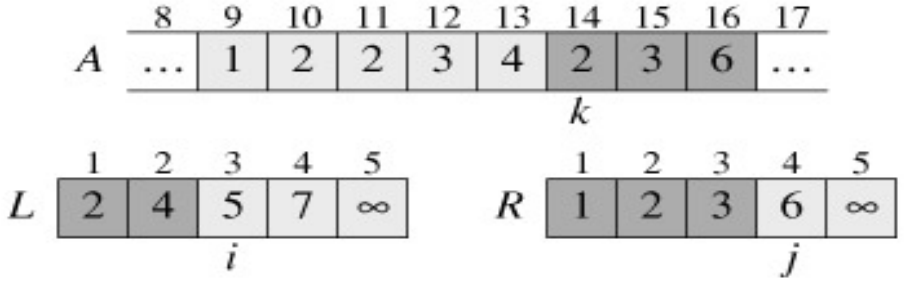
```
MERGE(A, p, q, r)
1    n1 ← q - p + 1
2    n2 ← r - q
3    create arrays L[1.. n1 + 1] and R[1.. n2 + 1]
4    for i ← 1 to n1
5         do L[i] ← A[p + i - 1]
6    for j ← 1 to n2
7         do R[j] ← A[q + j]
8    L[n1 + 1] ← ∞
9    R[n2 + 1] ← ∞
10   i ← 1
11   j ← 1
12   for k ← p to r
13        do if L[i] ≤ R[j]
14             then A[k] ← L[i]
15                  i ← i + 1
16             else A[k] ← R[j]
17                  j ← j + 1
```

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

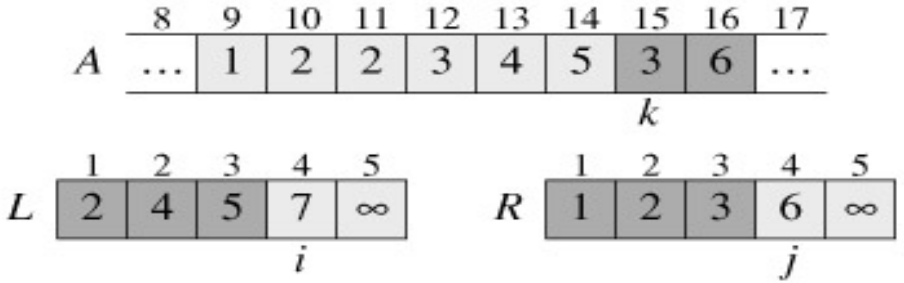



8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

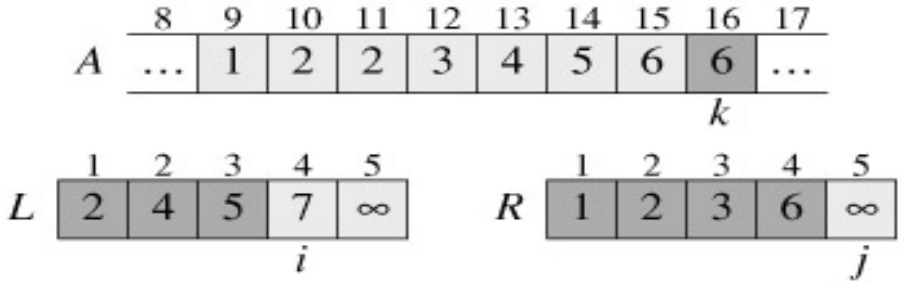
(e)


8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

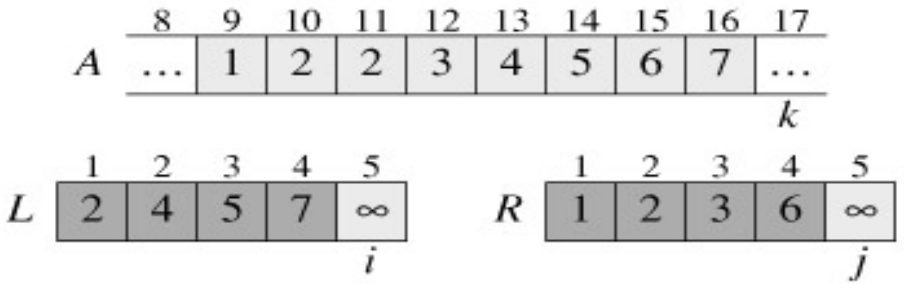
(f)


8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(g)

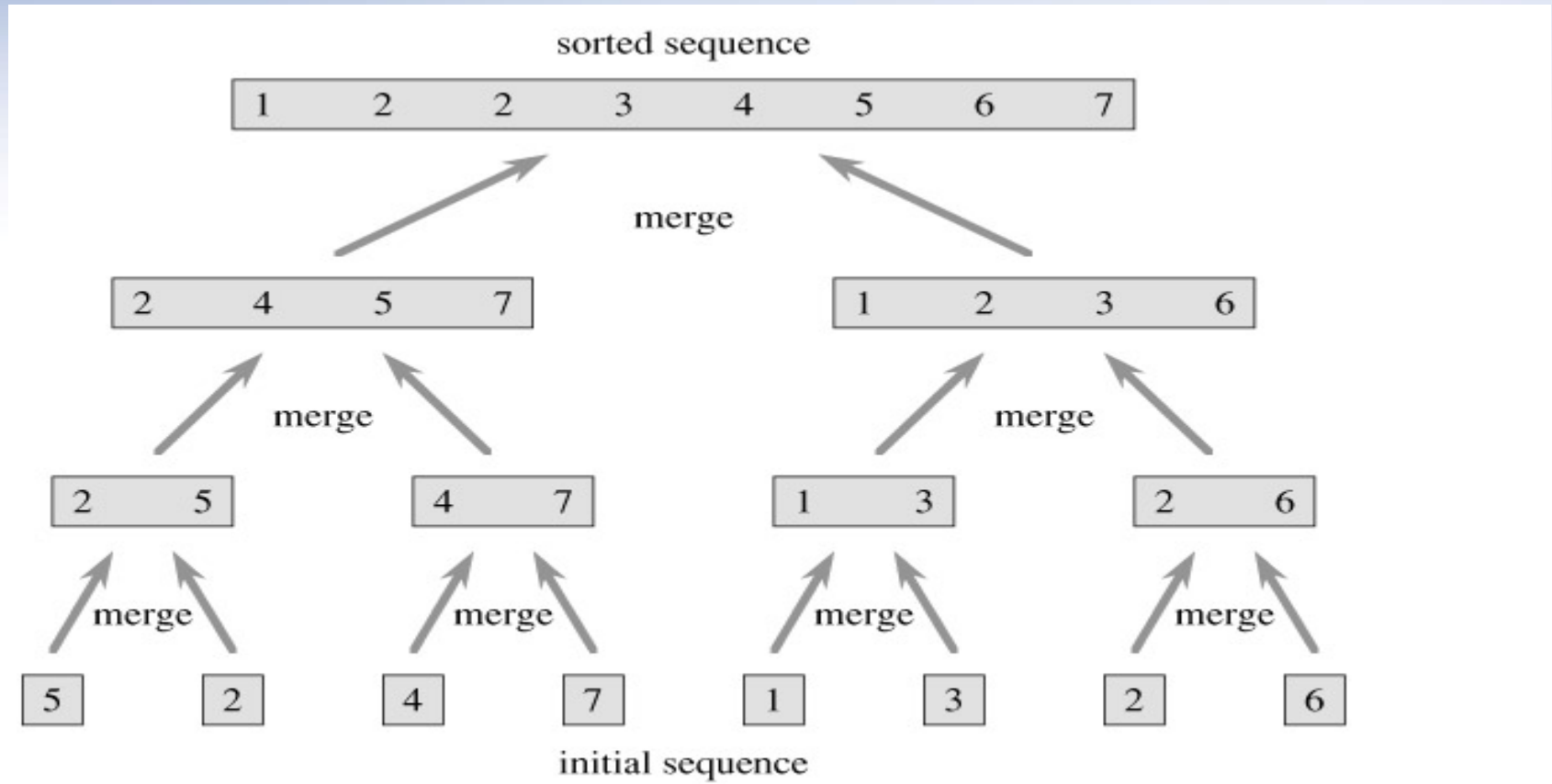

8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(h)


8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 7 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(i)

The operation of merge sort on the array $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$.



Analysis of Merge Sort

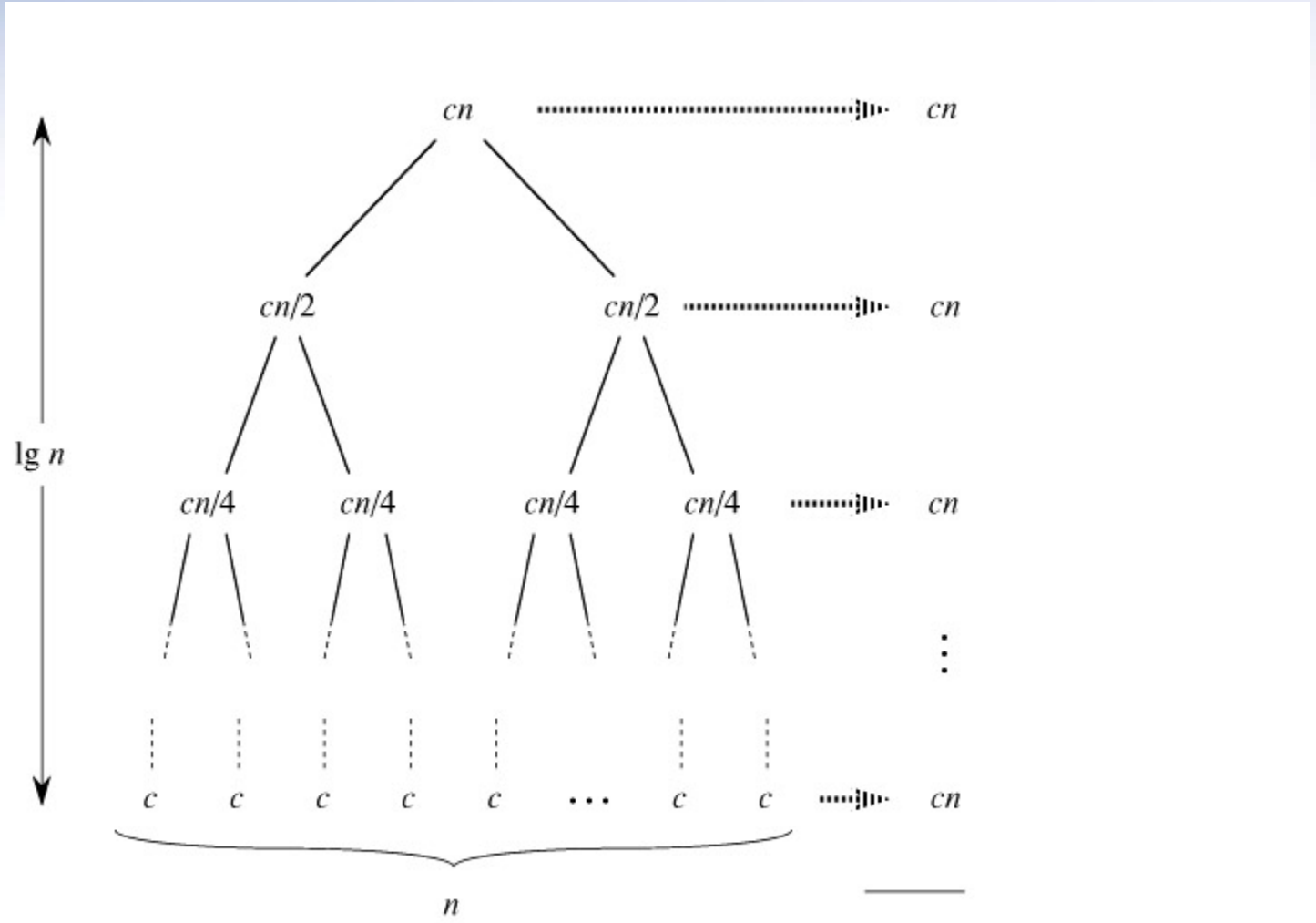
- To find the middle of the sub array will take $D(n)=\Theta(1)$.
- To recursively solve each sub problem will take $2T(n/2)$.
- To combine sub arrays will take $\Theta(n)$.

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

We can ignore $\Theta(1)$ term.

$\mathbf{T(n) = O(n \log n)}$

Analysis of Merge Sort



Summary.

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

Questions ???

Thank You..!!

See you on Next Week..!!

anuruddha.a@sliit.lk