SLIIT
Discover Your Future

IT2070 – Data Structures and Algorithms

Lecture 07
Introduction to Divider and Conquer Method

Today's Lecture

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

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.)

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.

Quick Sort procedure

Input: Unsorted Array (A,p,r)

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

```
QUICKSORT (A,p,r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT (A, p, q-1)
    QUICKSORT (A, q+1, r)
```

To sort an entire array A, the initial call is

QUICKSORT(A, 1, A.length).

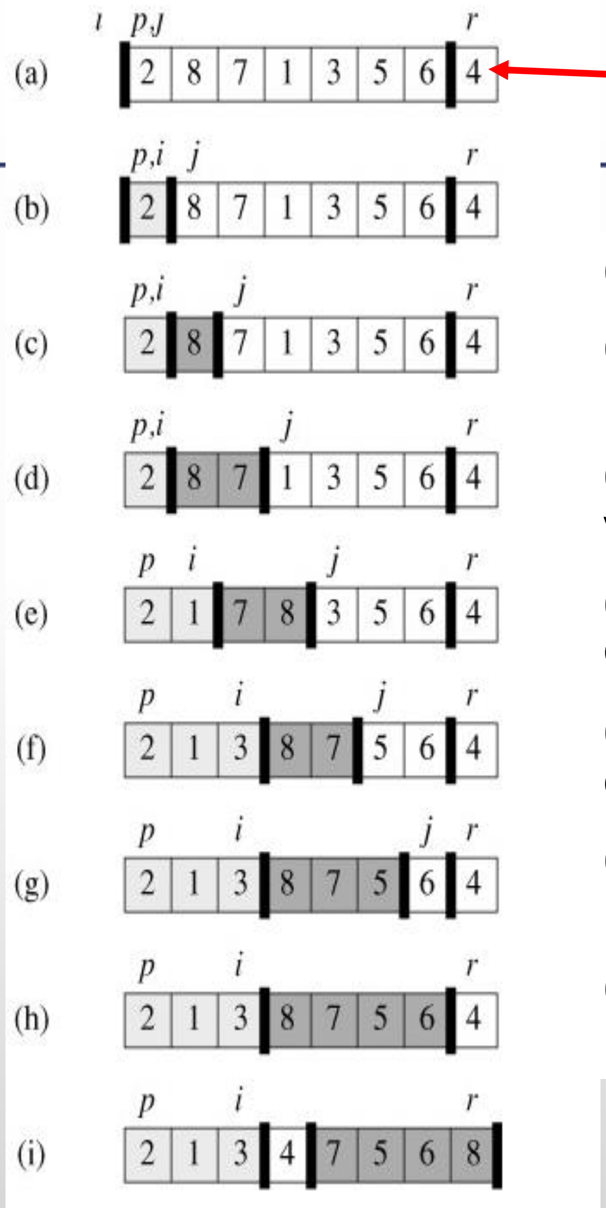
Partition Algorithm

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

```
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
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.



Element x = A[r] is the pivot element

(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.

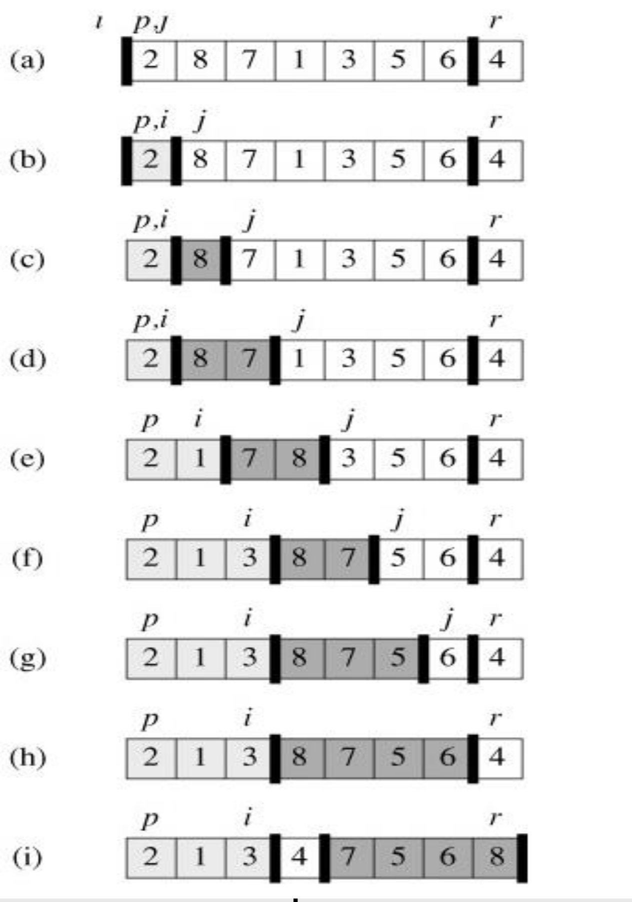
(e) The values 1 and 8 are swapped, and the smaller partition Grows.

(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 <= pivot

second partition with values > pivot

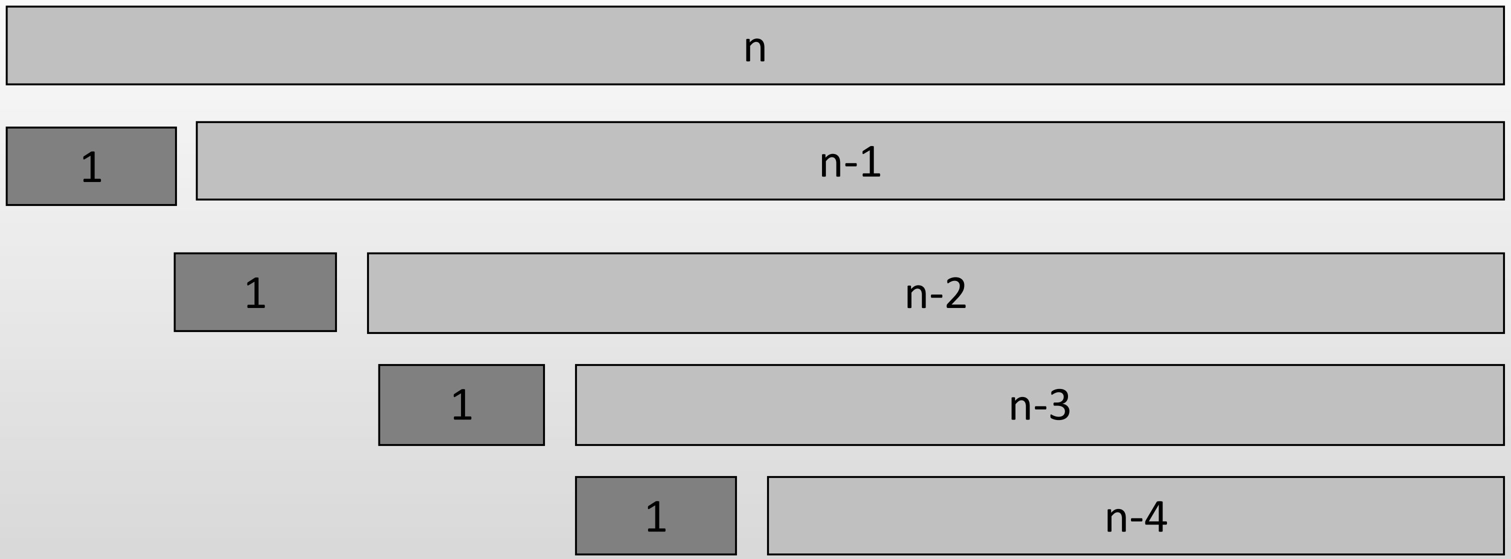
white element - pivot.

Analysis of Quick sort

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



SLIIT Discover Your Future

Analysis of Quick sort

Worst case partitioning (Repeated Substituted method)

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

Therefore **Recurrence** Equation is

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

$$= \underbrace{T(n-1)} + \Theta(n)$$

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

……………………

$$= T(0) + \Theta(1) + \Theta(2) + \ldots + \Theta(n-1) + \Theta(n)$$

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

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

Analysis of Quick sort

(b) Best case partitioning

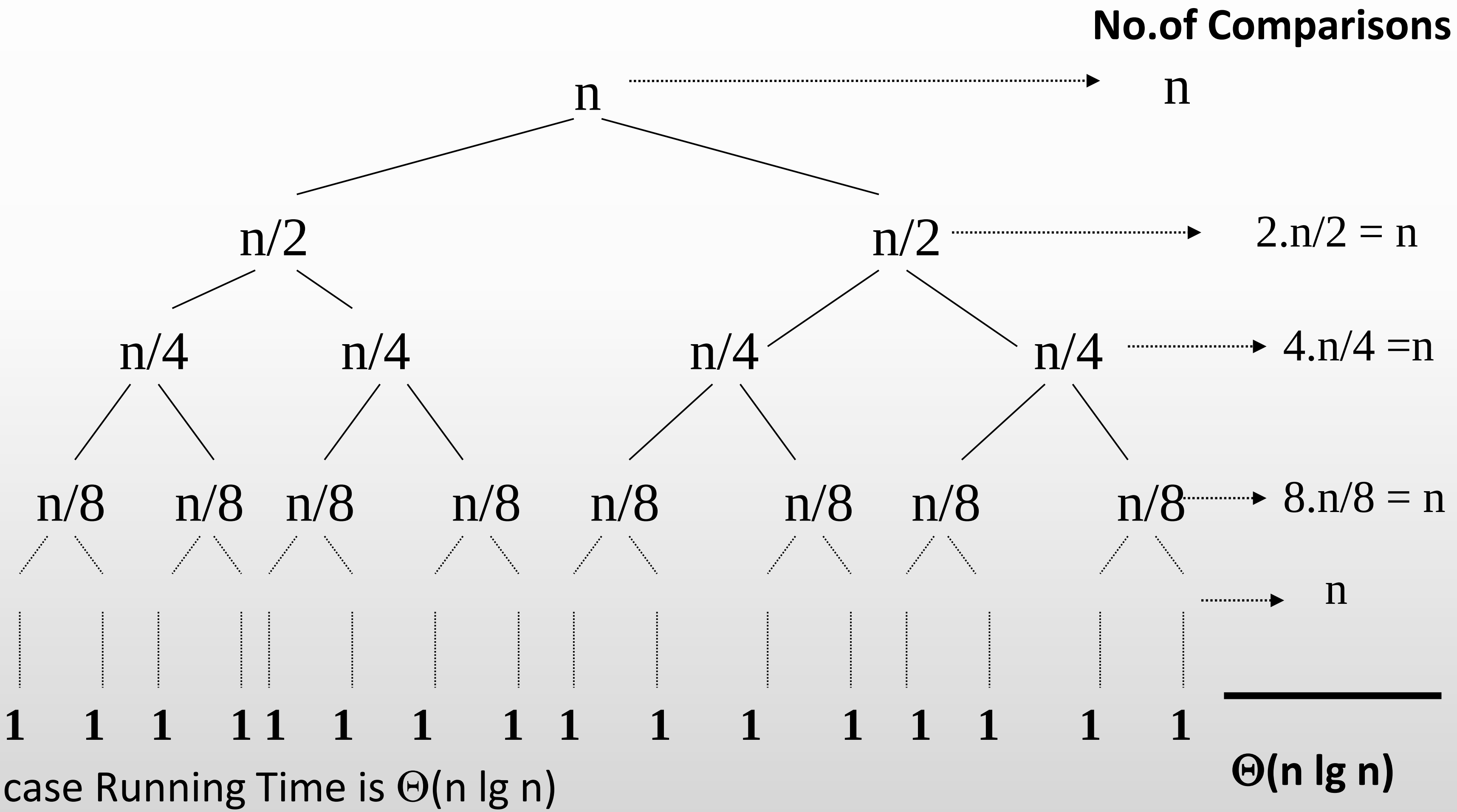
Best case occurs when PARTITION produces two sub arrays , one is of size (n-1)/2 and the other is of size (n-1)/2. In this case, quicksort runs much faster.

Recurrence equation is

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

Analysis of Quick Sort (with recursion tree)

Best Case:



Best case Running Time is Θ(n lg n)

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 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 $A[p \,..\, q]$ 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 $A[p \,..\, r]$.

To accomplish this step, we'll define a procedure MERGE(A, p, q, r).

Merge sort procedure

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

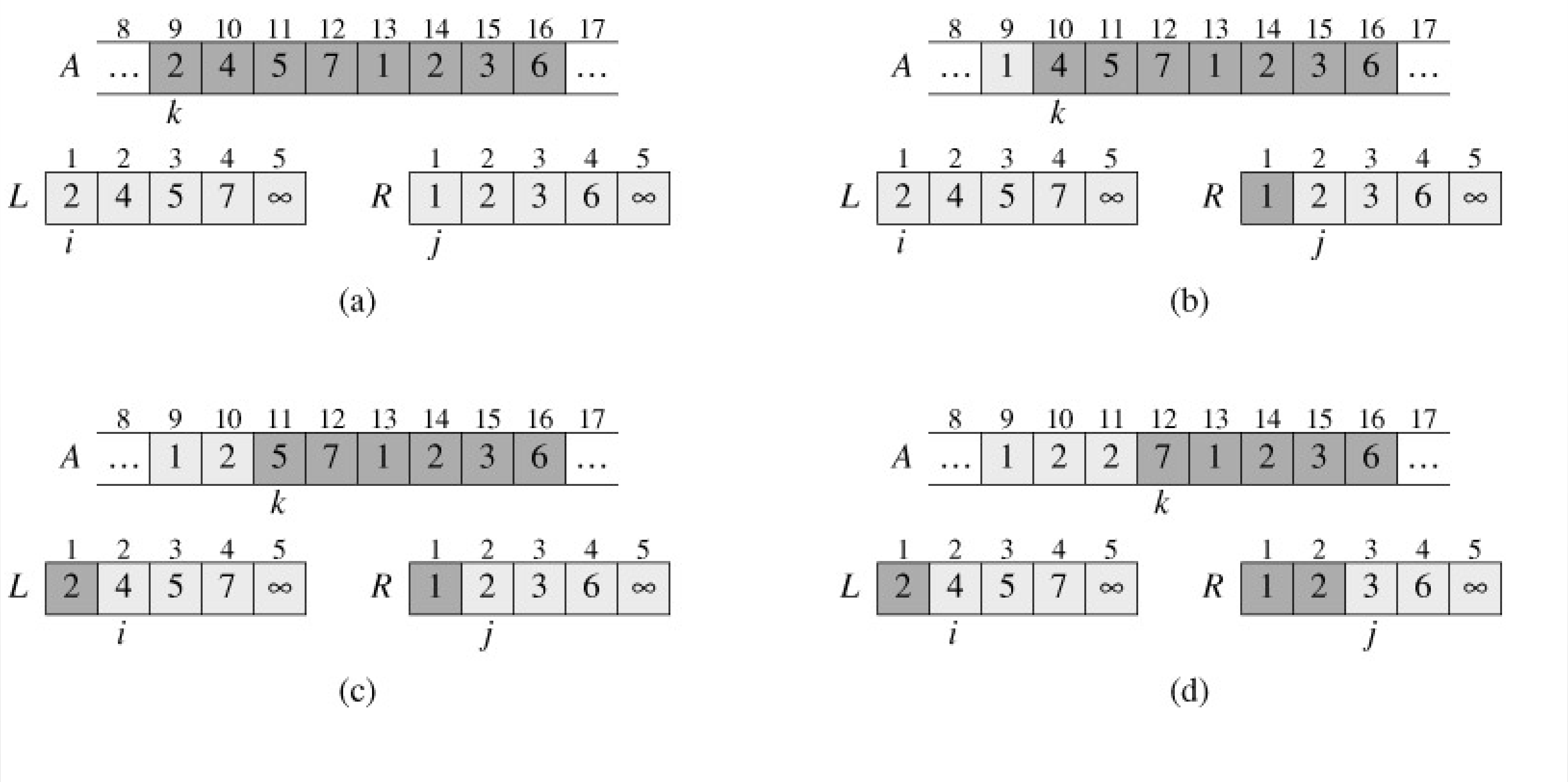
MERGESORT (A, p, r)

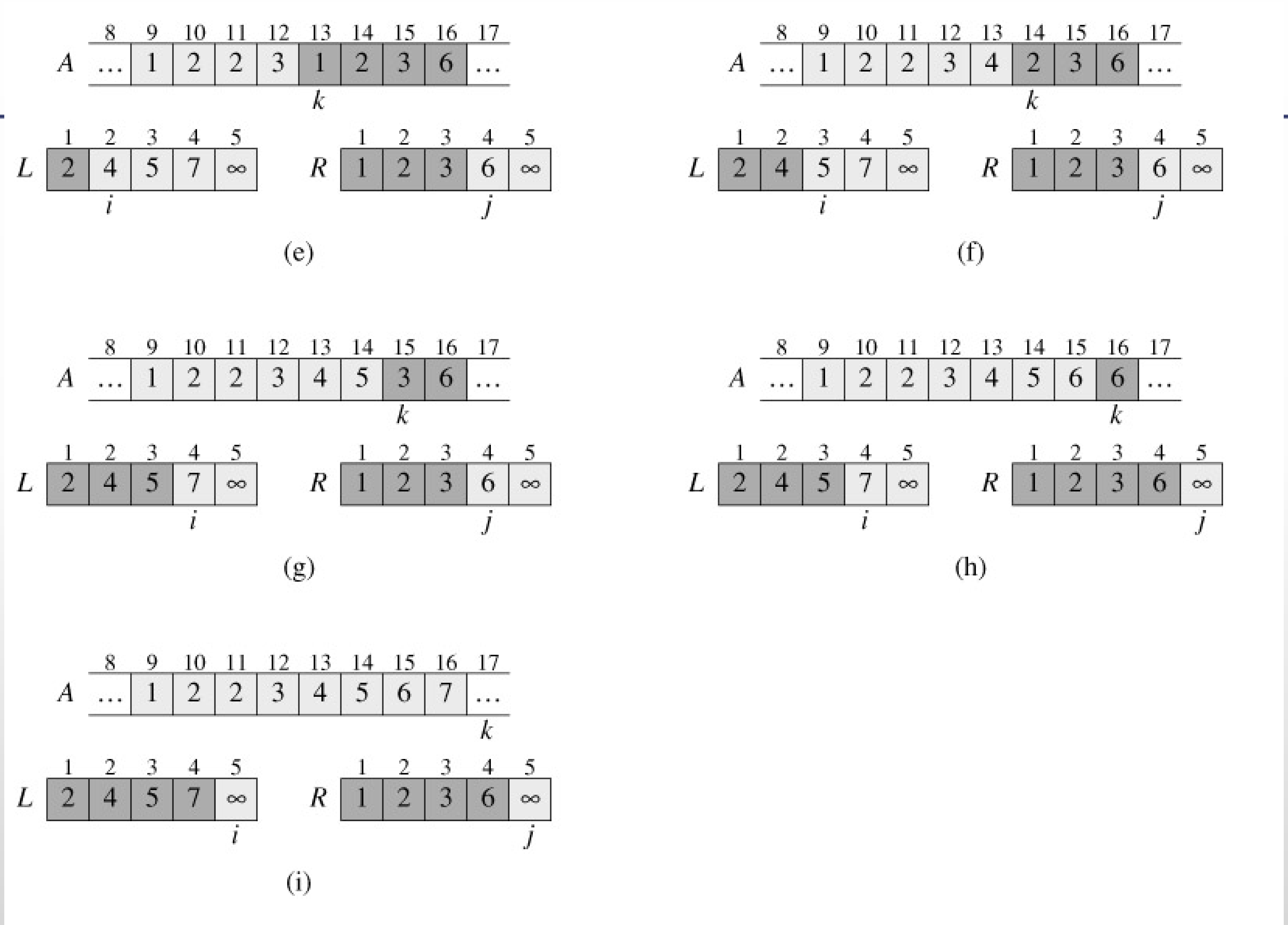
1. **if** p < r
2. q = ⌊(p+r)/2⌋
3. **MERGESORT** (A, p, q)
4. **MERGESORT** (A, q+1, r)
5. **MERGE** (A, p, q, r)

Merge procedure

```
MERGE(A, p, q, r)
    n1 = q - p + 1
    n2 = r - q
    create arrays 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
```

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)

Analysis of Merge Sort

- 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 $\Theta(n)$.

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

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

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

Analysis of Merge Sort

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

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

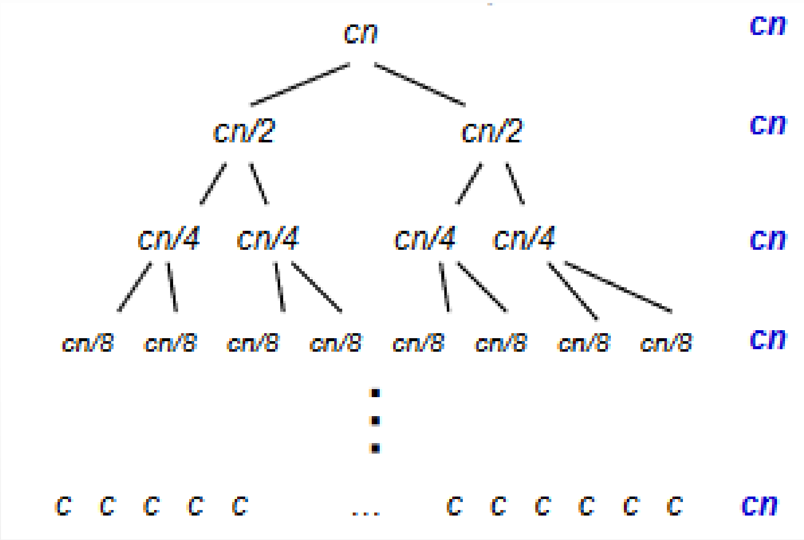
$4T(n/4) = 8T(n/8) + 4cn/4$

-

-

-

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



add and cancel:

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

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

Summary.

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