Analysis of Algorithms

Date
Page

"Step by step procedure of solving a problem or a task is called Algorithm."

	Priori Analysis	Posteriori Analysis
(i)	Analysis on Algorithm	Analysis of Program
(ii)	Independent of language	Dependent on language
(iii)	Hardware independent	Hardware dependent
(iv)	Time & space function	Watch time & bytes
(v)	Approximate analysis	Proper Analysis

⇒ Characteristics of Algorithms :

(i) Input : Should take 0 or more input.
(ii) Output : Should result atleast one output.
(iii) Definiteness : Should have clear meaning.
(iv) Finiteness : Algorithm should be finite.
(v) Effectiveness : Should not consist of unnecessary procedures.

⇒ How to analyze an Algorithm :

Eg,
```
Algorithm Swap (a, b) {
    temp = a ;  ------ 1
    a = b ;     ------ 1
    b = temp ;  ------ 1
}
```

Time:
$f(n) = 3 = O(1)$

Space:
$s(n) = 3 = O(1)$

[All constant values are said to be $O(1)$]

⇒ Frequency count method:

A =	8	3	9	7	2
	0	1	2	3	4

1. Eg, Algorithm Sum (A, n) {

```
    s = 0;                                  ——— 1
    for ( i=0, i<n, i++ ) {                 ——— n+1
        s = s + A[i];                       ——— n
    }
    return s;                               ——— 1
}
```

(1 for `i=0`, n+1 for `i<n`, n for `i++`)

∴ Time function: $f(n) = 1 + n + 1 + n + 1$

$= 2n + 3$

$= O(n)$

∴ Space function: $s(n) = n + 3$

$= O(n)$

A = n

n = 1

s = 1

i = 1

2. Eg, Algorithm Multiply (A, B, n) {

```
(n+1) ——— for (i=0; i<n; i++) {
(n+1) n ——— for (j=0; j<n; j++) {
        n×n ——— c = [i,j] = 0;
(n+1) n×n ——— for (k=0; k<n; k++) {
        n×n×n ——— c = [i,j] = c[i,j] + A[i,k] * B[k,j];
                }
            }
        }
    }
```

$\therefore f(n) = 2n^3 + 3n^2 + 2n + 1$

$= O(n^3)$

$\therefore s(n) = 3n^2 + 4$

$= O(n^2)$

Space :

$A = n^2$

$B = n^2$

$C = n^2$

$n = 1$

$i = 1$

$j = 1$

$k = 1$

Date
Page

⇒ Time Complexity Cheat Sheet [For loop]:

1. `for (i=0; i<n; i++)` ⟶ $O(n)$

2. `for (i=0; i<n; i = i+2)` ⟶ $O(n)$

3. `for (i=n; i>n; i--)` ⟶ $O(n)$

4. `for (i=1; i<n; i = i*2)` ⟶ $O(\log_2 n)$

5. `for (i=1; i<p; i = i*3)` ⟶ $O(\log_3 p)$

6. `for (i=n; i>1; i = i/2)` ⟶ $O(\log_2 n)$

7. `for (k=1, i=1, k<n, i++) {` ⟶ $O(\sqrt{n})$.
   `k = k+i;`
   `}`

⇒ Types of Time Function:

1. $O(1)$ ⟶ Constant
2. $O(\log n)$ ⟶ Logarithmic
3. $O(n)$ ⟶ Linear
4. $O(n^2)$ ⟶ Quadratic
5. $O(n^3)$ ⟶ Cubic
6. $O(2^n)$ ⟶ Exponential

Order of weightage :

$$1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \ldots < 2^n < 3^n \ldots < n^n$$

⇒ Asymptotic Notations :

1. O Big-oh [Upper Bound]
2. Ω Big-omega [Lower Bound]
3. Θ Theta [Average Bound]

1. Big-oh:

"The function $f(n) = O(g(n))$ if there exists +ve constants c & n_0, such that $f(n) \leq c * g(n)$ for all $n > n_0$"

Eg, $f(n) = 2n + 3$

$\therefore 2n + 3 = 5n$

$f(n)$ $\quad c \uparrow \quad \uparrow g(n)$

2. Omega:

"The function $f(n) = \Omega(g(n))$ if there exists +ve constants c & n_0, such that $f(n) \geq c * g(n)$ ~~for~~ for all $n > n_0$"

Eg, $f(n) = 2n + 3$

$\therefore 2n + 3 = 2n$

$f(n)$ $\quad c \uparrow \quad \uparrow g(n)$

3. Theta:

"The function $f(n) = \Theta(g(n))$ if there exists +ve constants c_1, c_2, n_0, such that, $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$:

Eg $\quad f(n) = 2n + 3$

$2n \leq 2n + 3 \leq 5n$

$c_1 * g(n) \quad f(n) \quad c_2 * g(n)$

⇒ Examples of Omega, Theta & Big-Oh :

1. $f(n) = 2n^2 + 3n + 4$

$O(n^2)$ $\qquad\qquad$ $\Omega(n^2)$

$2n^2 + 3n^2 + 4n^2 \geq 2n^2 + 3n + 4 \geq 1 \times n^2$

$\Rightarrow \quad 9n^2 \geq 2n^2 + 3n + 4 \geq n^2 \quad \Rightarrow \quad \Theta(n^2)$

2. $f(n) = n^2 \log n + n$

$1 \times n^2 \log n \leq n^2 \log n + n \leq 10 n^2 \log n \Rightarrow \Theta(n^2 \log n)$

3. $f(n) = n!$

$1 \times 1 \times 1 \times \ldots \times 1 \leq 1 \times 2 \times 3 \times 4 \ldots n \leq n \times n \times n \times \ldots \times n$

$1 \leq n! \leq n^n$

$\Omega(1)$ $\qquad\qquad$ $O(n^n)$

⇒ Properties of Asymptotic Notations :

1. **General Property :**
   If, $f(n)$ is $O(g(n))$, then $a * f(n)$ is $O(g(n))$

2. **Reflexive Property :**
   If, $f(n)$ is given, then, $f(n)$ is $O(f(n))$
   Eg, $f(n) = n^2$, $O(n^2)$

3. **Transitive Property :**
   If, $f(n) = O(g(n))$ & $g(n)$ is $O(h(n))$,
   then, $f(n) = O(h(n))$

4. **Symmetric Property :**
   If, $f(n)$ is $\Theta(g(n))$,
   then, $g(n)$ is $\Theta(f(n))$

5. **Transpose Symmetric :**
   If, $f(n) = O(g(n))$,
   then, $g(n) = \Omega(f(n))$

⇒ Best, Worst & Average Case Analysis : [Linear Search, Binary Search Tree]

1. Linear Search : Best case = $O(1)$
   Worst case = $O(n)$
   Average case = $O\left(\frac{n+1}{2}\right)$

2. Binary Search Tree : Best case = $O(1)$
   Worst case = $O(\log n)$

⇒ Recurrence Relation : [Decreasing Function]

1. $f(n) = n+1$; $T(n) = O(n)$

[Solving using Backsubstitution]

$T(n)$	
$n+1$	→ void Test (int n) {
	→ if (n > 0) {
1	→ printf ("%d", n);
$T(n-1)$	→ Test (n-1);
	}
	}

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

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n-1)+1 & n > 0 \end{cases}$$

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= [T(n-2)+1] + 1 \\ &= T(n-2) + 2 \\ &= [T(n-3)+1] + 2 \\ &= T(n-3) + 3 \\ &\vdots \\ &\vdots \\ &= T(n-K) + K \quad \text{——— ①} \end{aligned}$$

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

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

$T(n-2) = T(n-3) + 1$

Assume $n - K = 0$

$\therefore n = K$

$$\begin{aligned} \therefore T(n) &= T(n-n) + n \\ &= T(0) + n \\ &= 1 + n \end{aligned}$$

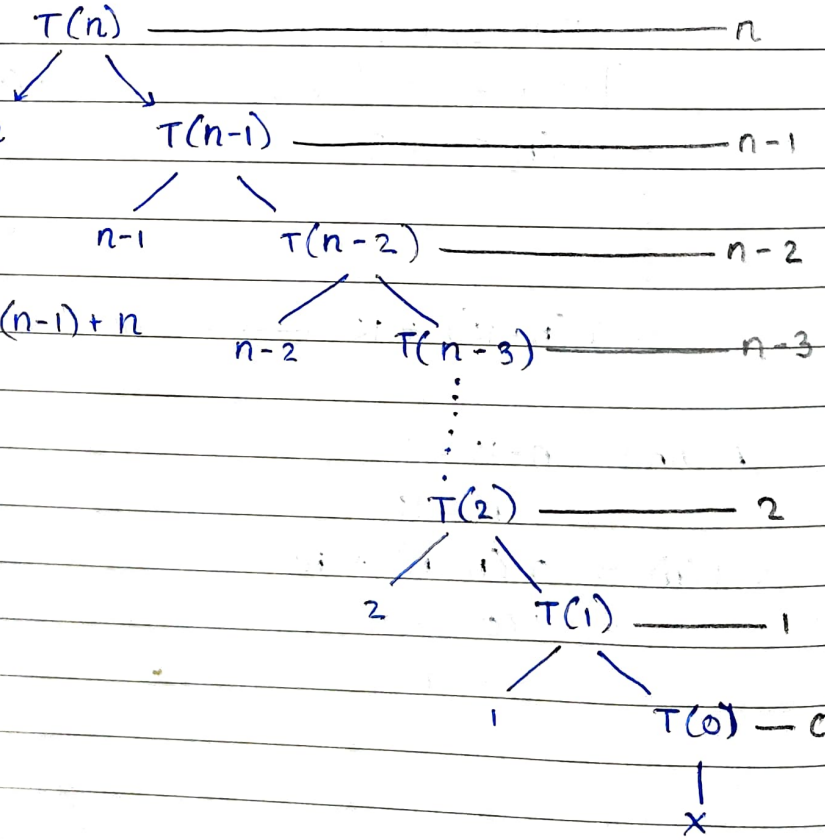
2. [Solving using Recursive Tree]

```
T(n)  void Test (int n) {
  1   ⟶  if (n>0) {
n+1   ⟶  for (i=0; i<n; i++) {
  n   ⟶  printf ("%d", n);
      }
T(n-1) ⟶ Test (n-1);
      }
   }
```

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

$= T(n-1) + n$

$$\therefore T(n) = \begin{cases} 1 & n = 0 \\ T(n-1)+n & n > 0 \end{cases}$$



$\therefore 0+1+2+\ldots+(n-1)+n$

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

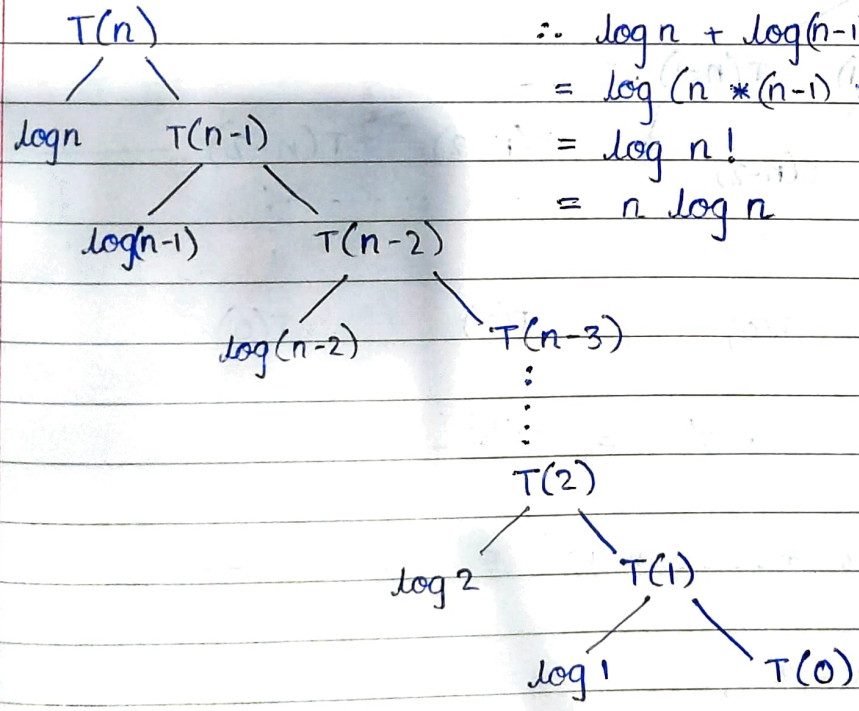
$= O(n^2)$

3.

$T(n)$	void Test(int n) {
1	if (n>0) {
$\log n$	for (i=1, i<n; i=i*2) {
	printf("%d", i);
	}
$T(n-1)$	Test(n-1);
	}
	}

$\therefore T(n) = T(n-1) + \log n$

$$T(n) = \begin{cases} 1 & n=0 \\ T(n-1) + \log n & n>0 \end{cases}$$



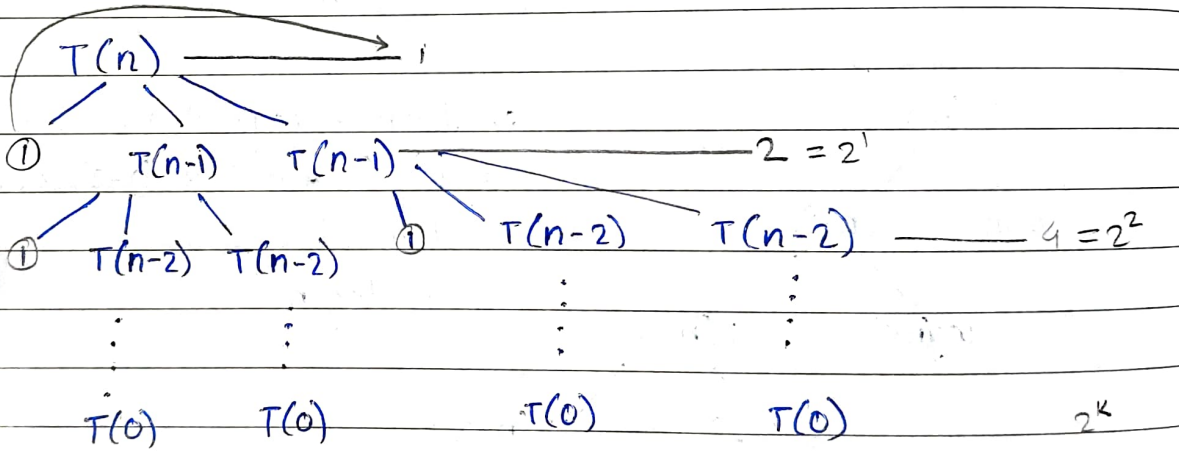
$\therefore \log n + \log(n-1) + \dots + \log 2 + \log 1$
$= \log(n * (n-1) \times (n-2) \times \dots \times 2 \times 1)$
$= \log n!$
$= n \log n$

4.

$T(n)$	`void Test (int n) {`
	`if (n > 0) {`
1	`print ("%d", n);`
$T(n-1)$	`Test (n-1);`
$T(n-1)$	`Test (n-1);`
	`}`
	`}`

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

$$\therefore \boxed{T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1)+1 & n > 0 \end{cases}}$$



$$1 + 2^1 + 2^2 + \ldots + 2^K = 2^{K+1} - 1$$

$$\left[a + ar + ar^2 + \ldots + ar^K = \frac{a(r^{K+1} - 1)}{r - 1} \right.$$

$\therefore a = 1, \quad r = 2,$

$$= \frac{1(2^{K+1} - 1)}{2 - 1}$$

$$= 2^{K+1} - 1$$

Assume $n - K = 0$

$n = K$

$$= 2^{n+1} - 1$$

$$= O(2^n)$$

$$T(n) = \begin{cases} 1 & n = 0 \\ 2T(n-1) + 1 & n > 0 \end{cases}$$

$$T(n) = 2T(n-1) + 1 \quad \text{---} ①$$

$$T(n) = 2[2T(n-2) + 1] + 1$$

$$= 2^2 T(n-2) + 2 + 1 \quad \text{---} ②$$

$$= 2^2 [2T(n-3) + 1] + 2 + 1$$

$$= 2^3 T(n-3) + 2^2 + 2 + 1 \quad \text{---} ③$$

$$\vdots$$

$$= 2^k T(n-k) + 2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2 + 1$$

Assume $n - k = 0$

$n = k$

$$\therefore T(n) = 2^k T(n-k) + [2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2^1 + 1]$$

$$= 2^n T(0) + 2^k - 1$$

$$= 2^n (1) + 2^n - 1$$

$$= 2^{n+1} - 1$$

$$= O(2^n)$$

⇒ Recurrence Relations Cheatsheet:

1. $T(n) = T(n-1) + 1 \longrightarrow O(n)$
2. $T(n) = T(n-1) + n \longrightarrow O(n^2)$
3. $T(n) = T(n-1) + \log n \longrightarrow O(n \log n)$
4. $T(n) = 2T(n-1) + 1 \longrightarrow O(2^n)$
5. $T(n) = 3T(n-1) + 1 \longrightarrow O(3^n)$
6. $T(n) = 2T(n-1) + n \longrightarrow O(n 2^n)$

⇒ Master Theorem for decreasing function:

$$\boxed{T(n) = aT(n-b) + f(n)}$$

Where, $a > 0$, $b > 0$ & $f(n) = O(n^k)$

Case 1: When, $a < 1 \Rightarrow O(n^k)$
$O(f(n))$

Case 2: When, $a = 1 \Rightarrow O(n \times f(n))$
$O(n^{k+1})$

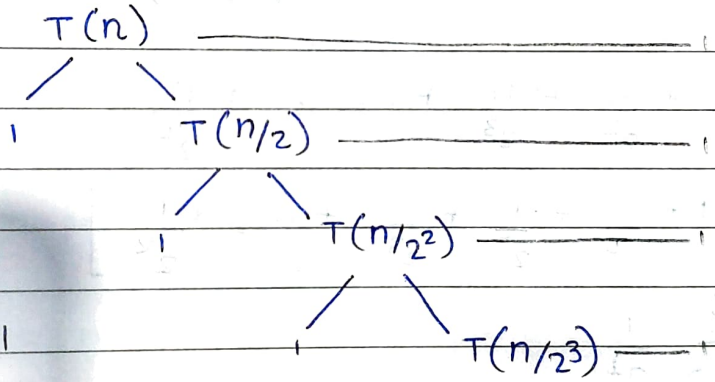
Case 3: When, $a > 1 \Rightarrow O(n^k\ a^{n/b})$
$O(f(n)\ a^{n/b})$

⇒ Recurrance Relation : [Dividing Function]

1.
```
void Test (int n) {                    ——— T(n)
    if (n > 1) {
        print ("%d", n);                ——— 1
        Test (n/2);                    ——— T(n/2)
    }
}
```

$\therefore T(n) = T(n/2) + 1$

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



Since, $\frac{n}{2^K} = 1$

$\Rightarrow n = 2^K$

$\Rightarrow K = \log_2 n$

$= O(\log n)$

$\therefore \frac{n}{2^K} = 1$

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

$= [T(n/2^2) + 1] + 1$

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

$= T(n/2^3) + 3$

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

Assume $n/2^K = 1$

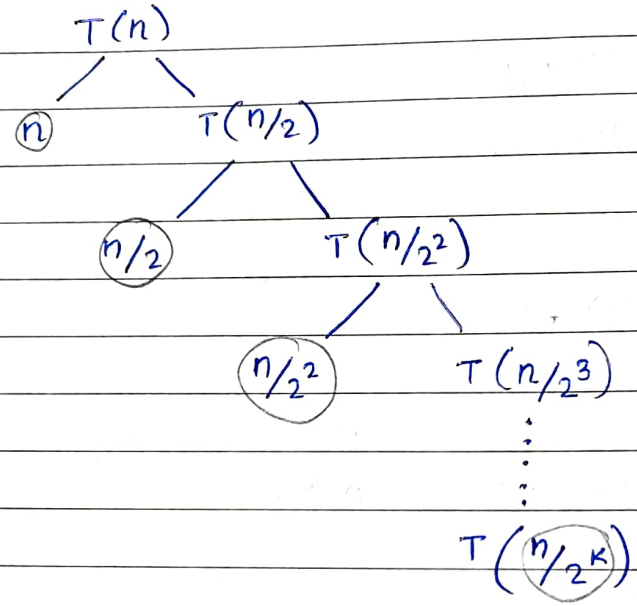
$\therefore n = 2^K \Rightarrow K = \log_2 n$

$\therefore T(n) = T(1) + \log n$

$= 1 + \log n$

$= O(\log n)$

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



$$n + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \dots + \frac{n}{2^K}$$

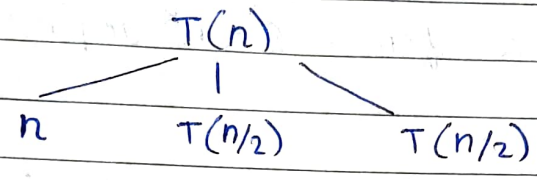
$$= n\left[1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \dots + \frac{1}{2^K}\right]$$

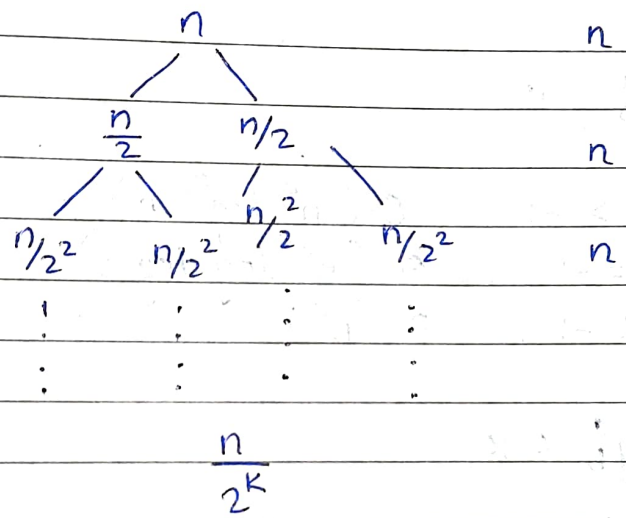
$$= n \sum_{i=0}^{K} \frac{1}{2^i}$$

$$= n \times 1$$

$$= O(n)$$

3. $T(n) = \begin{cases} 1 & n=1 \\ 2T(n/2) + n & n>1 \end{cases}$





k steps, time per step = n

$\therefore T(n) = nk$

Assume, $\frac{n}{2^k} = 1$,

$\Rightarrow n = 2^k$

$\Rightarrow k = \log_2 n$

$\therefore T(n) = n \log n$

$= O(n \log n)$

⇒ Master Theorem for dividing function:

$T(n) = aT(n/b) + f(n)$

Where, $a \geq 1$ & $f(n) = \Theta(n^k \log^p n)$

$b > 1$

Find,

(i) $\log_b^a$

(ii) k

Case 1: If $\log_b^a > k$, then $\Theta(n^{\log_b^a})$

Case 2: If $\log_b^a = k$,

(i) If $p > -1$, $\Theta(n^k \log^{p+1} n)$

(ii) If $p = -1$, $\Theta(n^k \log \log n)$

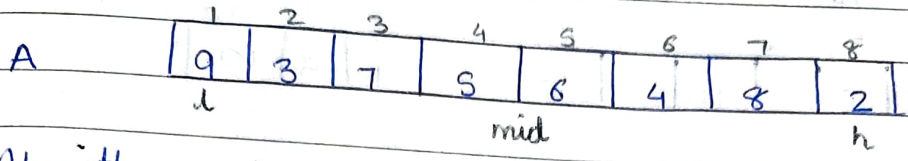
(iii) If $p < -1$, $\Theta(n^k)$

Case 3: If $\log_b^a < k$,

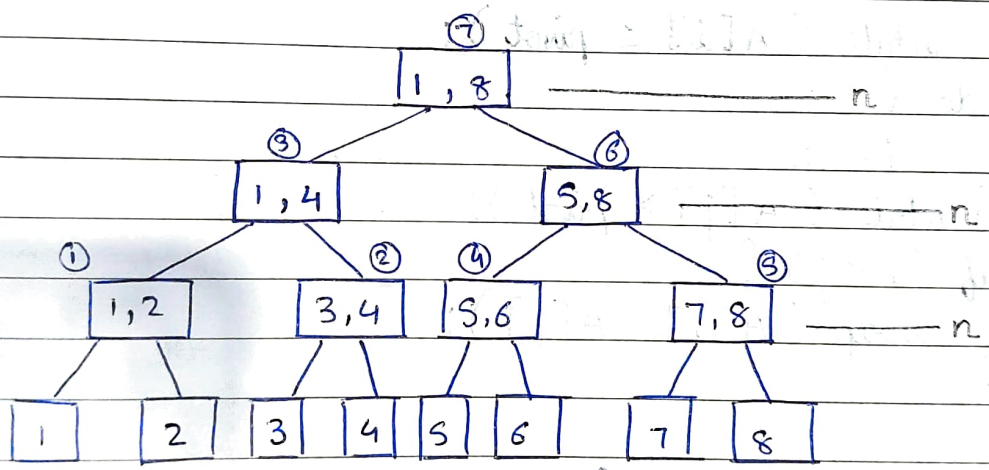
(i) If $p \geq 0$, $\Theta(n^k \log^p n)$

(ii) If $p < 0$, $O(n^k)$

⇒ Merge Sort:




```
Algorithm MergeSort (l, h) {                    ——— T(n)
    if (l < h) {                                ——— 1
        mid = (l+h)/2;                          ———
        MergeSort (l, mid);                     ——— T(n/2)
        MergeSort (mid+1, h);                   ——— T(n/2)
        Merge (l, mid, h);                      ——— n
    }
}
```



Time complexity : $O(n \log n)$

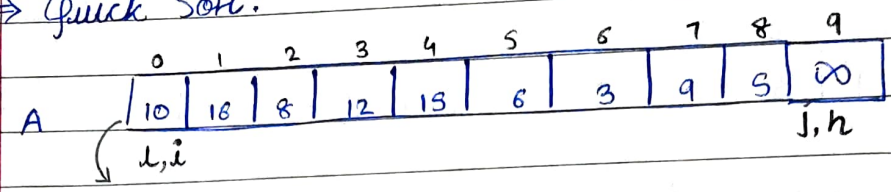
$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n/2) + n & n > 1 \end{cases}$$

$a = 2, \quad b = 2, \quad K = 1, \quad p = 0$

$\therefore \log_b^a = \log_2^2 = 1, \quad \because K = 1 \quad \therefore \log_b^a = K$

$\therefore \Theta(n \log n)$

⇒ Quick Sort:




```
Partition (l, h) {
    pivot = A[l];
    i = l; j = h;
    while (i < j) {
        do {
            i++;
        } while ( A[i] ≤ pivot );
        do {
            j--;
        } while ( A[j] > pivot );
        if (i < j) {
            swap (A[i], A[j]);
        }
    }
    swap (A[l], A[j]);
    return j;
}

QuickSort (l,h) {
    if ( l < h) {
        j = partition (l,h);
        QuickSort (l, j);
        QuickSort (j+1, h);
    }
}
```