Algorithm Analysis & Design (AAD)

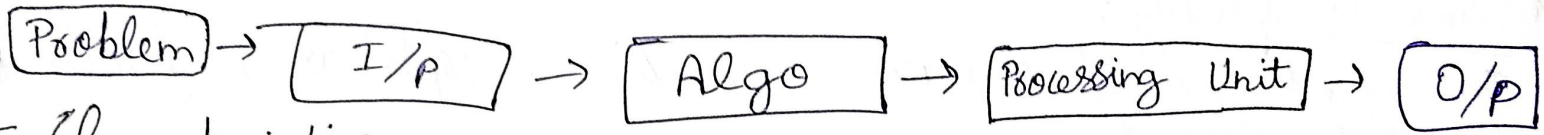
#Algorithm :-

- An Algo is a finite set of instructions that flow a particular task
- All Algo must satisfy the following criteria.

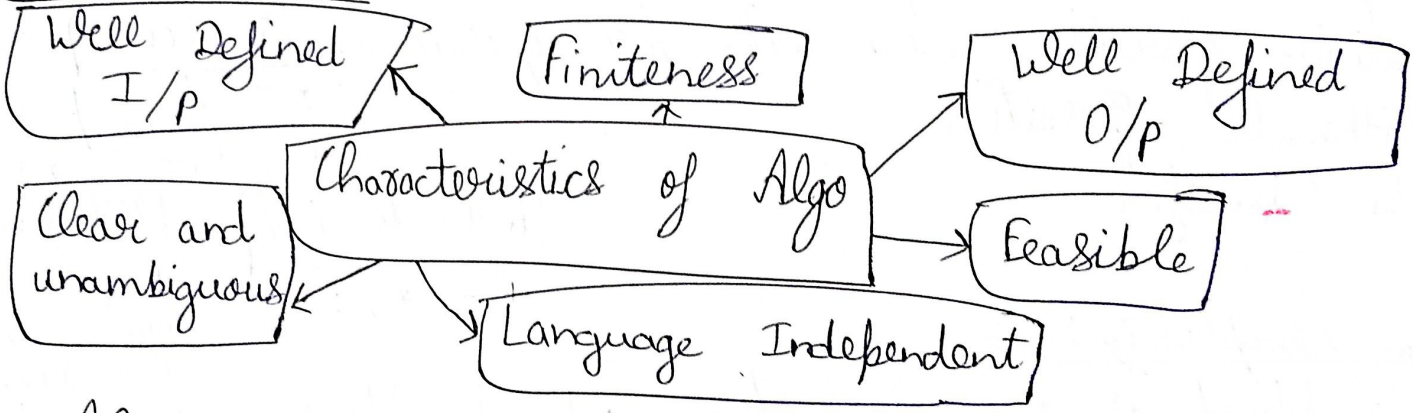
Rules :-

→ I/P : An Array of integers
→ O/P : Max int in the Array
→ Definitness : Algo must be clear, precise & unambiguous.
→ Finitness : Algo must terminate after a finite no of steps
→ Effectiveness : Capability of producing a desired result.

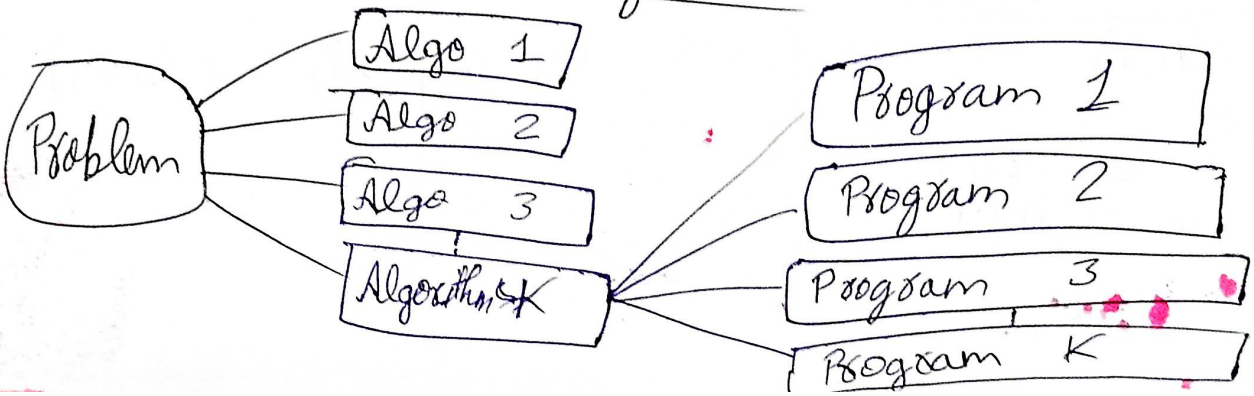
Data Flow :-



Characteristics :



Algorithm V/S Program



- Classification of Algo:

→ Recursive Algo
→ Backtracking Algo
→ Divide and Conquer Algo
→ Dynamic Programming Algo
→ Greedy Algo
→ Branch and Bound Algo
→ Brute Force Algo
→ Randomized Algo

+ Algo Analysis :-

→ I/p Size
→ Runing time (Worst case + Any case)
→ Order of growth function
(For e.g. → $5n^2 + 3n + 2 = O(n^2)$)

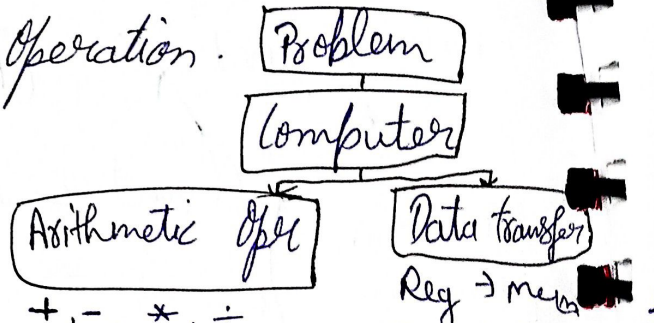
Linear Search
WC = O(N)
BC = O(1)
AC = $\frac{n+1}{2} = O(n)$ ①

* Time Complexity :-

The count no of premitive operation (Steps) as a function of input size ②

Premitive operation done with two operation.

→ Arithmetic Operation
→ Data Transfer



* Space Complexity :-

It is defined as a memory storage and its depend on the i/p size

Frequency Count Method :-

```
Algo Sum (A, n)
{
  Sum = 0
  For (i=0 ; i≤n ; i++) {
  Sum = Sum + A(i)
  }
  return Sum;
}
```

Time Complexity = O(n+1)

N)
1)
1 = O(n)

unction

ta transfer
→ Mem
→ Reg
→ Reg
i/p Size

1)

① main ()
{
int x, y, z ; → 1
x = y + z ; → 1
}

$T(n) = 1 + 1 = 2$
$\Rightarrow O(1)$

$3n^2 + 5n + 2$
$\Rightarrow O(n^2)$
∴ Higher Degree Conside

② main ()
{
int n ;
For (i = 0 ; i <= n ; i++) → n + 1
{
Statement → n
}
}

$T(n) = 2n + 1$
$\Rightarrow O(n)$

* Algo add (A, B, n)
{
For (int i = 0 ; i < n ; i++) ⟶ n + 1
{
For (int j = 0 ; j < n ; j++) ⟶ n (n + 1)
{
C[i, j] = A[i, j] + B[i, j] → n × n
}
}
}

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

Space Complexit
$A = n^2$
$B = n^2$
$C = n^2$
$i = 1$
$j = 1$
$n = 1$

$= 3n^2 + 3$
$S(n) = O(n^2$

Algo Multiply (A, B, n)

```
{
For (i=0 ; i<n ; i++)          → n+1
  {
  For (j=0 ; j<n ; j++)        → n (n+1)
   {
    C[i, j] = 0 ;              → n × n
   For (k=0 ; K<n ; k++)       → n×n (n+1)
     {
      C[i, j] = C[i, j] + A[i, k] * B[k, j] → n×n×n
     }
  }
}
```

$$T(n) =$$

$$= n+1 + n^2+n + n^2 + n^3 + n^2 + n^3$$

$$= 2n^3 + 3n^2 + 3n + 1$$

$$\boxed{T(n) \Rightarrow O(n^3)}$$

$$S(n) = A + B + c + i + j + K + n$$

$$= n^2 + n^2 + n^2 + 1 + 1 + 1 + 1$$

$$= 3n^2 + 4$$

$$\boxed{S(n) = O(n^2)}$$

⊙ main ()

```
{
 int i, n;
 For ( int i=1 ; i<n ; i = i * 2)
    {
     Statement
    }
}
```

$n = 32$

$\because \log_2 32 = 5$

$\therefore \log_2 10 = 3.6$

$\Rightarrow 4$

$i = 1+2+4+8+16 \ldots 2^K = n$

$2^K = n$

$$\boxed{K = O(\log_2 n)}$$

⊙ main ()

```
{ int i, n;
For (i= n ; i≥1 ; i = log i/2)
   {
    Statement
   }
}
```

$n = 16$

$\Rightarrow 5$

$n = 15$

$\Rightarrow 4$

```
main ()
{
int i, j, n;                          → 1
for (int i = 1; i ≤ n; i++)           → n+1
  {
  for (int j = 1; j ≤ n; j = j * 2)   → n x log n
    { printF("GILA");                 → n x log n
    }
  }
}
```

$T(n) = 2 + n + 2n \log n$

$= n \log n$

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

$n = \theta$

$= \theta \log_2(2)^3$

$\Rightarrow 24$

```
main()
{
int P, i, j, n;                       → 1
int P = 0                             → 1
For (i = 1; i < n; i = i x 2)         → log n
  { P++;                              → n * log n
  }
For (j = 1; j < P; j = j x 2)         → log p   x n x log n
  {
  Statement;                          → log log n
  }
}
```

Peripheral device (like, Mouse, Keyboard, ...) — CPU

$T(n) = \underbrace{1+1}_{\text{Peripheral device}} + \underbrace{\log n}_{\text{CPU}}$

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

✱ Comparison of Function (Growth Function) :-

Formula of log

$\log(m \times n) = \log m + \log n$

$\log(m/n) = \log m - \log n$

$\log_a a = 1 \quad , \log 1 = a$

$\log_n m$

$m \log n$

$a \log_b n = n \log_b a$

✱ Typesial Time Functions :-

$O(1)$ = Constant time

$O(\log_2 n)$ = Logarithmic time

$O(n)$ → Linear time

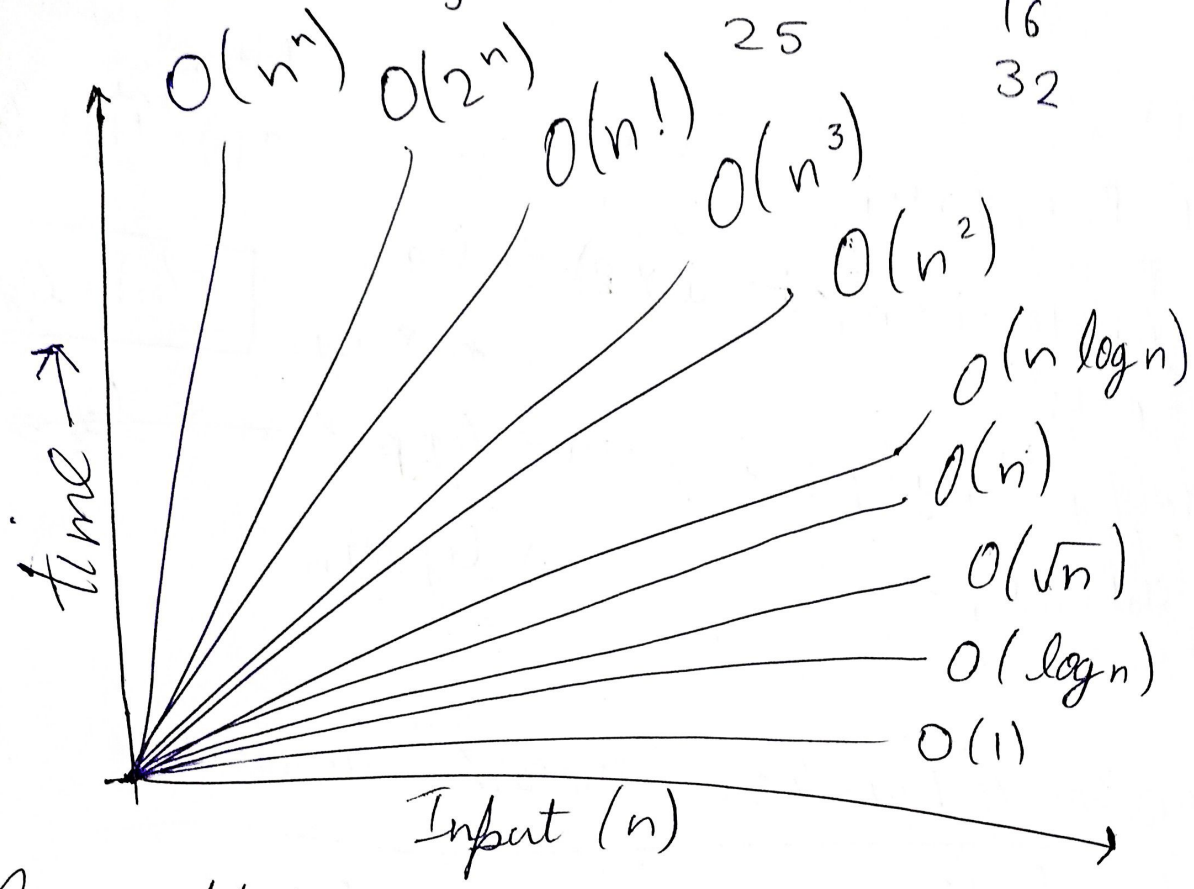
$O(n^2)$ → Quadratic time

$O(n^3)$ → Cubic time

$O(2^n)$ → Exponential time

$1 < \log n < n < n^2 < n^3 < 2^n \text{ ---- } < n^n$ Relation

	$\log_2 n$	n	n^2	2^n
n=1	0	1	1	2
n=2	1	2	4	4
n=3	1.3	3	9	8
n=4	2	4	16	16
n=5	2.5	5	25	32



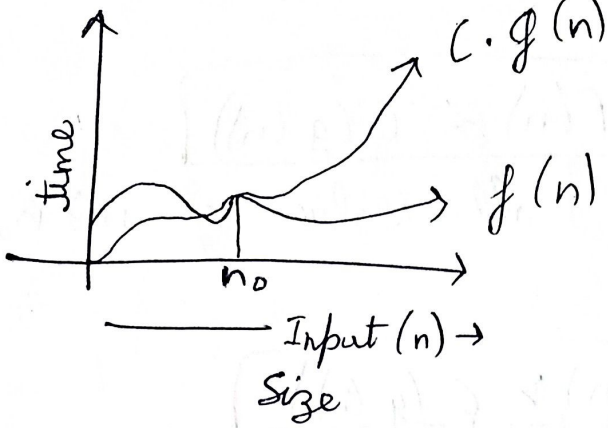
Asymptotic Notation :-

- → Big - Oh Notations (O) → Upper Bound
- → Big - Omega Notations (Ω) → Lower Bound
- → Theta Notation (θ) → Average Bound

Big - Oh Notation (O):-

$f(n) = O(g(n))$

$\boxed{f(n) \leq C \cdot g(n)}$ for some value of $C > 0$ && for all $n \geq 0$ n_0.



	$f(n) = n^2$	$g(n) = 2^n$
n=1	1	2
n=2	4	4
n=3	9	8
n=4	16	16
n=5	25	32
n=6	36	64

	$F(n) = 3n+2$	$g(n) = n$
1	5	1
2	8	2
3	11	3
4	14	4
5	17	5
6	20	6

$3n + 2 \leq Cn$

$3 \times 1 + 2 \leq 5 \times 1$ $\quad \therefore C = 5$

$5 \leq 5$

$\boxed{\text{Min equality Satisfied}}$

$\boxed{C = 5}$

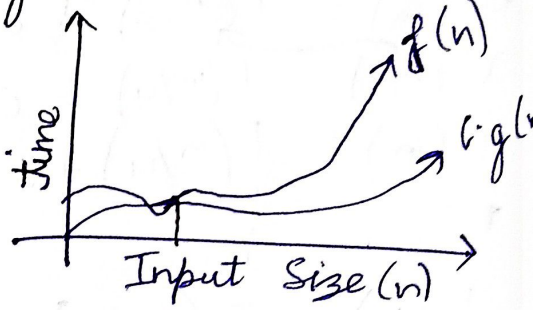
Big - Omega Notation (Ω):-

$f(n) \geq C \cdot g(n)$

for some value of $C > 0$ && for all $n \geq n_0$

$\boxed{F(n) = \Omega(g(n))}$

	$F(n) = 3n+2$	$g(n) = n$
n=1	5	1
n=2	8	2
n=3	11	3
n=4	14	4



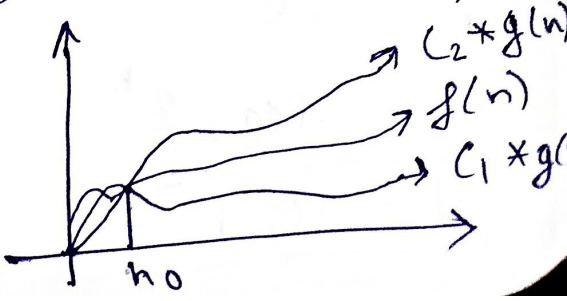
Theta Notation (Θ):-

The Functions & $f(n) = \Theta(g(n))$; if some +ve Cont^n C_1, C_2 && Such that

$\boxed{C_1 * g(n) \leq f(n) \leq C_2 * g(n)}$

Ω $\quad f$ $\quad O$

Θ



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

$1 * n \leq 2n + 3 \leq 5 * n$

$c_1 = 1$
$c_2 = 5$
$n_0 = 1$

$f(n) = \theta(g(n))$

* Small O Notation :-

$f(n) = o(g(n))$ if $\boxed{f(n) < c(g(n))}$

For all value of $c > 0$ && all value of $n \geq n_0$

* Small omega Notation (ω) :-

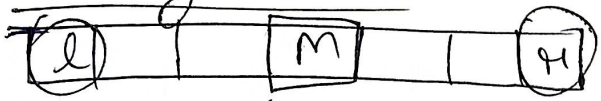
$f(n) = \omega(g(n))$ if $\boxed{f(n) > c(g(n))}$

For all value of $c > 0$ && for all value of $n \geq n_0$

Recurrences Method :-

- Subsitution Method
- Iteration Method (Back Subsitution Summation Method)
- Recursion Free Method
- Master theorem Method

- Binary Search :-



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

$T(n/2) = T(n/4) + c$ (:: Iteration Method)

$T(n) = T(n/4) + c + c$

$\vdots$

$T(n) = T\left(\frac{n}{2^k}\right) + kc$

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

$n = 2^k$

$\log n = \log_2 2^k$

$\boxed{K = \log n}$

```
while (l ≤ q1)
  {
  mid = (l + q1)/2  → c
  if{
    l = mid + 1  → n/2
  }else {
    q1 = mid - 1  → n/2
  }
```

Substitution Method :-

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

We have to show that it is asymptotic bound by $O(\log n)$.

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

$T(n) \leq C \cdot \log(n)$ $\quad (\because f(n) \leq C \cdot g(n))$

$\leq C \cdot \log(n/2) + 1$

$\leq C[\log n - \log 2] + 1$

$= C \log n - C \log 2 + 1$

$= C \log n - C + 1$

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

Consider the recursive

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

Find the asymptotic bound on $T(n)$.

We guess the soln is $O(n \log n)$, thus for a Const

$T(n) \leq C \cdot n \log n$

$= \leq 2C(n/2) \log(n/2) + n$

$= \leq cn \log n - cn \log 2 + n$

$= \leq cn \log n - cn + n$

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

Sorting Method :-

- Bubble Sort $[O(n^2)]$
- Insertion Sort $[O(n^2)]$
- Selection Sort $[O(n^2)]$
- Quick Sort $[O(n^2)]$
- Heap Sort $[O(n \log n)]$
- Merge Sort $[O(n \log n)]$

} Comparision Sort

Linear Sorting

- Radix Sort $= O(n)$
- Bucket Sort $= O(n)$
- Count Sort $= O(n)$

} Non-comparision Sort

- Shell Sort $= O(n)$

} Comparision Sort

* Bubble Sort :-

$\Rightarrow \frac{n(n+1)}{2}$ $\quad T(n) = O(n^2)$

```
Bubble Sort (A, n)
{
  for (i=0; i<n-1; i++)        ⇒ n+1
  {
    for (j=0; j<n-1-i; j++)    ⇒ n x n
    {
      if (A[j] > A[j+1])
      {
        temp = A[i];
        A[i] = A[j+1];
        A[j+1] = temp;
```

0	1	2	3	4
9	7	0	2	5

Pass1 :
7 9 0 2 5
7 0 9 2 5
7 0 2 9 5
7 0 2 5 9

Pass 2 :
7 0 2 5 9
7 2 0 5 9
7 2 5 0 9

Pass 3 :
7 2 5 0 9
2 7 5 0 9
2 5 7 0 9

Pass 4 : 2, 5, 7, 0 9

⇒ 10 Swap & Sort

Q. $T(n) = \begin{cases} 1, & n = 1 \\ T(n-1) + c, & \text{when } n > 1 \end{cases}$

∴ By using Iterative Method

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

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

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

$T(n-2) = T(n-3) + c$ ~~$+ c + c$~~

$T(n) = T(n-3) + c + c + c$

$T(n-3) = T(n-4) + c$

$T(n) = T(n-4) + c + c + c + c$

⋮ k time

$T(n) = T(n-k) + kc$

$n - k = 1$

$\boxed{k = n-1}$

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

$= T(1) + nc - c.$

$= 1 + nc - c$

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

Q. $T(n) = \begin{cases} 1, & n = 1 \\ 2T(n/2) + n, & \text{where } n > 1 \end{cases}$ Merge Sort

∴ By using Iterative Method

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

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

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

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

$= T(n) + n + n$

$T(n/4) = 2T(n/8) + n/4$

$T(n) = 2^2(2T(n/8) + n/4) + n + n$

$T(n) = 2^3 T(n/8) + n + n + n$

⋮ k time

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

$n/2^k = 1 \Rightarrow n = 2^k$

$\boxed{k = \log_2 n}$

$= 2^{\log n}(1) + n \log n$

$= n^1 \times (1) + n \log n$

$\boxed{T(n) = O(n \log_2 n)}$

Q: $T(n) = \begin{cases} 1 & , n = 0 \\ T(n-1) + n & , n > 1 \end{cases}$ Quick Sort

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

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

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

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

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

$T(n-3) = T(n-4) + n - 3$

$T(n) = T(n-3) + 4n - 6$

$\vdots$ k time

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

$n - k = 0 \quad ; \quad n = k$

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

$= T(0) + 1 + 2 + 3 + \ldots n$

$= 1 + 2 + 3 + \ldots n$ (∵ Neglect the value of $T(0)$)

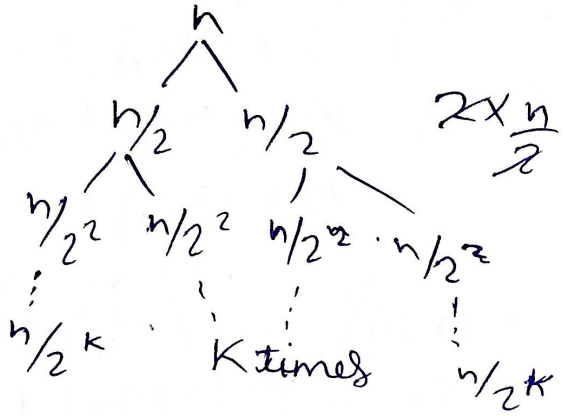
$= \frac{n(n+1)}{2}$

$\boxed{T(n) = O(n^2)}$

Recursive Tree Method:

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

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



$n/2^k = 1$

$n = 2^k$

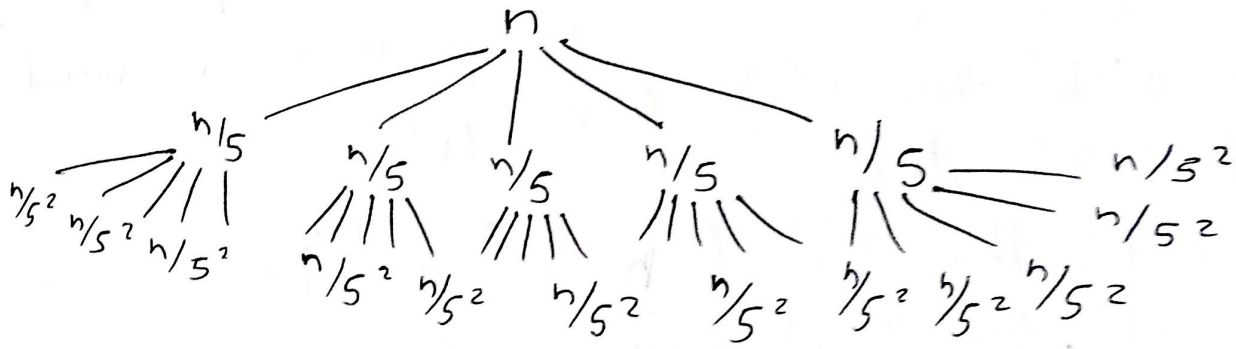
$K = \log_2 n$

$= n + n + n + \ldots kn$

$\Rightarrow O(n \log_2 n)$

Q. $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 5T(n/5) + n \end{cases}$

$= T\left(\frac{n}{5}\right) + T\left(\frac{n}{5}\right) + T\left(\frac{n}{5}\right) + T\left(\frac{n}{5}\right) + T\left(\frac{n}{5}\right) + n$



$n/5^k = 1$

$n = 5^k$

$k = \log_5 n$

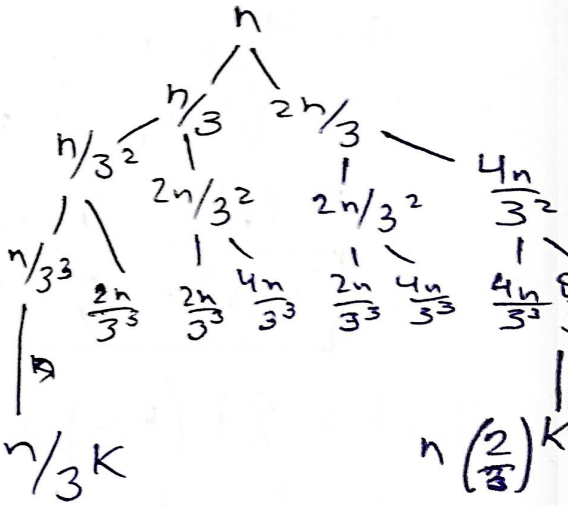
$n + n + n + n \ldots + kn$

$\Rightarrow O(n \log_5 n)$

Q. $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$

$T\left(\frac{n}{3}\right) = T\left(\frac{n}{3^2}\right) + T\left(\frac{2n}{3^2}\right) + \frac{n}{3}$

$T\left(\frac{2n}{3}\right) = T\left(\frac{2n}{3^2}\right) + \left(\frac{4n}{3^2}\right) + \frac{2n}{3}$



$= \frac{n}{3} + \frac{2n}{3} = n$

$= \frac{n}{3^2} + \frac{2n}{3^2} + \frac{2n}{3^2} + \frac{4n}{3^2} = n$

$= \frac{n}{3^3} + \frac{2n}{3^3} + \frac{2n}{3^3} + \frac{4n}{3^3} + \frac{2n}{3^3} + \frac{4n}{3^3} + \frac{4n}{3^3} + \frac{8n}{3^3} = n$

$n/\left(\frac{3}{2}\right)^k \Rightarrow \left(\frac{3}{2}\right)^k = n$

$k = \log_{\frac{3}{2}} n$

$\Rightarrow O(n \log_{\frac{3}{2}} n)$

Master Method :-

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$, with $a \geq 1$ and $b \geq 1$

$T(n) = aT\left(\frac{n}{b}\right) + \theta\left(n^{k} \log^{p} n\right)$, where, $a \geq 1$, $b > 1$, $k \geq 0$ and p is a real number

Case 1: if $a > b^k$ then $T(n) = \theta(n^{\log_b a})$

Case 2: if $a = b^k$ then

(a) $p > -1$, then $T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

(b) $p = -1$, then $T(n) = \theta(n^{\log_b a} \times \log\log n)$

(c) $p < -1$, then $= \theta(n^{\log_b a})$

Case 3: if $(a < b^k)$

(a) if $p \geq 0$ then $T(n) = \theta(n^k \log^p n)$

(b) if $p < 0$ then $T(n) = \theta(n^k)$

Q. $T(n) = 4T(n/2) + n$

$= 4T(n/2) + n^1 \log^0 n$ $\quad (\because \log 0 = 1)$

$a = 4$, $b = 2$, $k = 1$, $p = 0$

$(a > b^k) = (4 > 2^1)$; Case 1 proved

$T(n) = \theta(n^{\log_b a}) = \theta(n^{\log_2 4}) = \theta(n^{\log_2 2^2})$

$\boxed{T(n) = \theta(n^2)}$

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

$a = 2$, $b = 2$, $k = 1$, $p = 0$

$(a = b^k) = (2 = 2^1)$

$T(n) = \theta(n^{\log_b a} \times \log^{p+1} n) = n^{\log_2 2} \times \log^{0+1} n$

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

Q. $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$

$= 2T\left(\frac{n}{2}\right) + n \log n^{-1}$

$a = 2 \;,\; b = 2 \;,\; K = 1 \;,\; p = -1$

$(a = b^K) = (2 = 2^K)$

$T(n) = n^{\log_b a} \log \log n$

$= n^{\log_2 2} \log \log n$

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

Q. $T(n) = 2T(\sqrt{n}) + \log n$

$n = 2^m \Rightarrow m = \log_2 n$

$T(2^m) = 2T(2^{m/2}) + \log 2^m$

$= 2T(2^{m/2}) + m$

$S(m) = 2T(2^{\frac{\log m}{2}}) + m \log_2 n$

$a = 2 \;,\; b = 2 \;,\; K = 0 \;,\; p = 0$

$(2 > 2^0) = (a > b^K)$

if $p > -1$

$T(m) = O(m^{\log_b a} \log m^{p+1})$

$= m \log^{2^2}_2 \log m^{0+1}$

$= m \log m$

$\Rightarrow O(\log_2 n \log \log_2 n)$

Q. $T(n) = 2T(\sqrt{n}) + 1$

$n = 2^m$, $m = \log_2 n$, $1 = \frac{n}{2^m}$

$= 2T(2^m) + m^{-1} \log_2 n$

~~$S(m) = 2S($~~

$a = 2$, $b = 2$, $K = -1$, $P = 1$

$(a \geq b^K) = 2 > 2^{-1}$

if $P > -1$

$T(n) = O(m^{\log_b a} \log m^{P+1})$

$= m^{\log_2 2} \log m^{1+1}$

$= \log n \log m^2$

$= \log n \, 2$

$\Rightarrow 2 \log n$

$S(m) = 2S(\frac{m}{2}) + m^0 \log n^0$

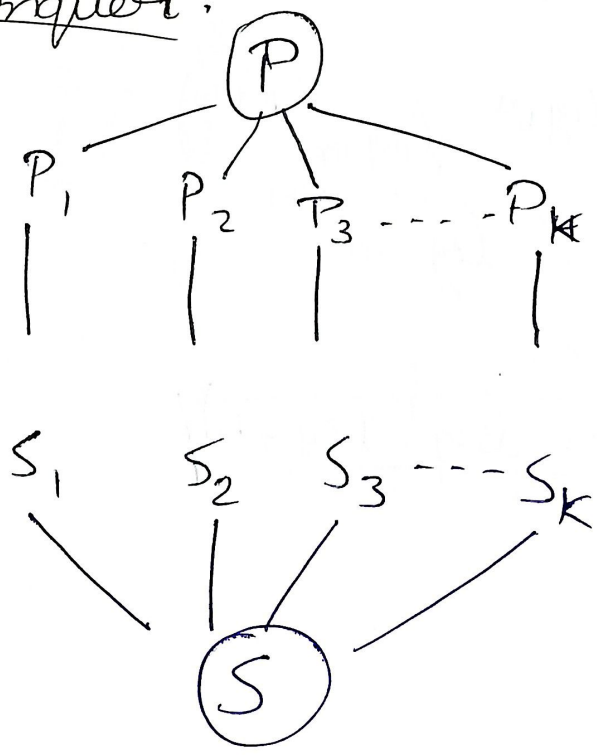
Sorting Techniques :-

* Divide & Conquer :

→ Divide

→ Conquer

→ Combine



- Merge Sort
- Quick Sort

Merge sort:

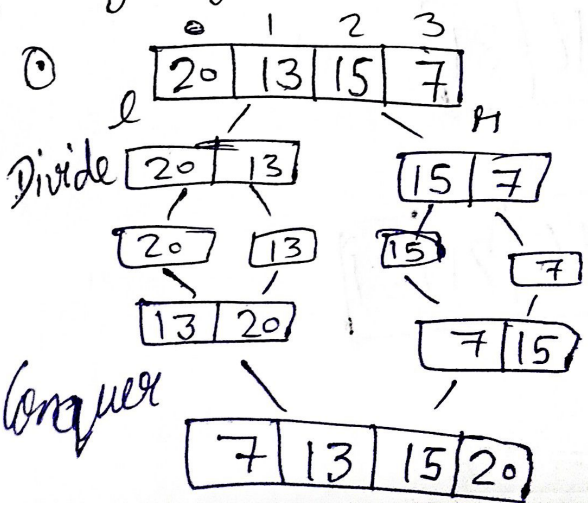
```
Merge (A, B, q, r)
1. n1 = q - p + 1
2. n2 = r - q
3. Let L[1 --- n1+1] and R[1 --- n2+1] (New array)
4. For (i = 1 to n1)
5. L[i] = A[p + i - 1]
6. For (j = 1 to n2)
7. R[j] = A[q + j]
8. L[n1+1] = ∞
9. R[n2+1] = ∞
10. i = 1 , j = 1
11. for (k = p to r)
12. if (L[i] ≤ R[j])
13. A[k] = L[i]
14. i = i + 1
15. else A[k] = R[j]
16. j = j + 1
```

Time Complexity = n + n + n
= 3n

$$T(n) = O(n)$$

Space Complexity = 2n + 2

$$S(n) = O(n)$$



$$mid = \frac{l + r}{2} = \frac{0 + 3}{2}$$

$$= 1.5 \Rightarrow 1$$

Merge Sort (A, P, q)

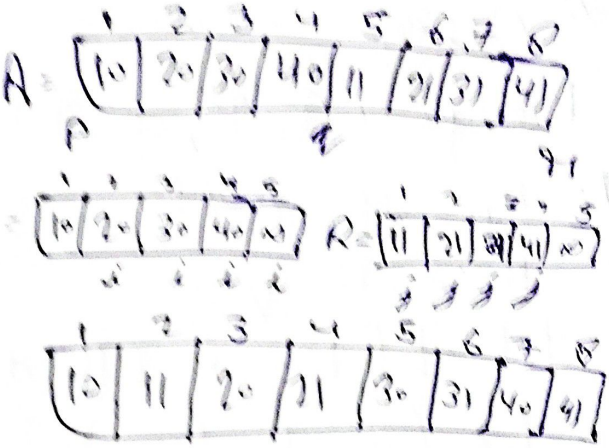
if (P < q)

$q = \left(\frac{P+q}{2}\right)$

Merge Sort (A, P, q) → n/2

Merge Sort (A, q+1, q) → n/2

Merge Sort (A, P, q, q+1) → O(n)



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

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

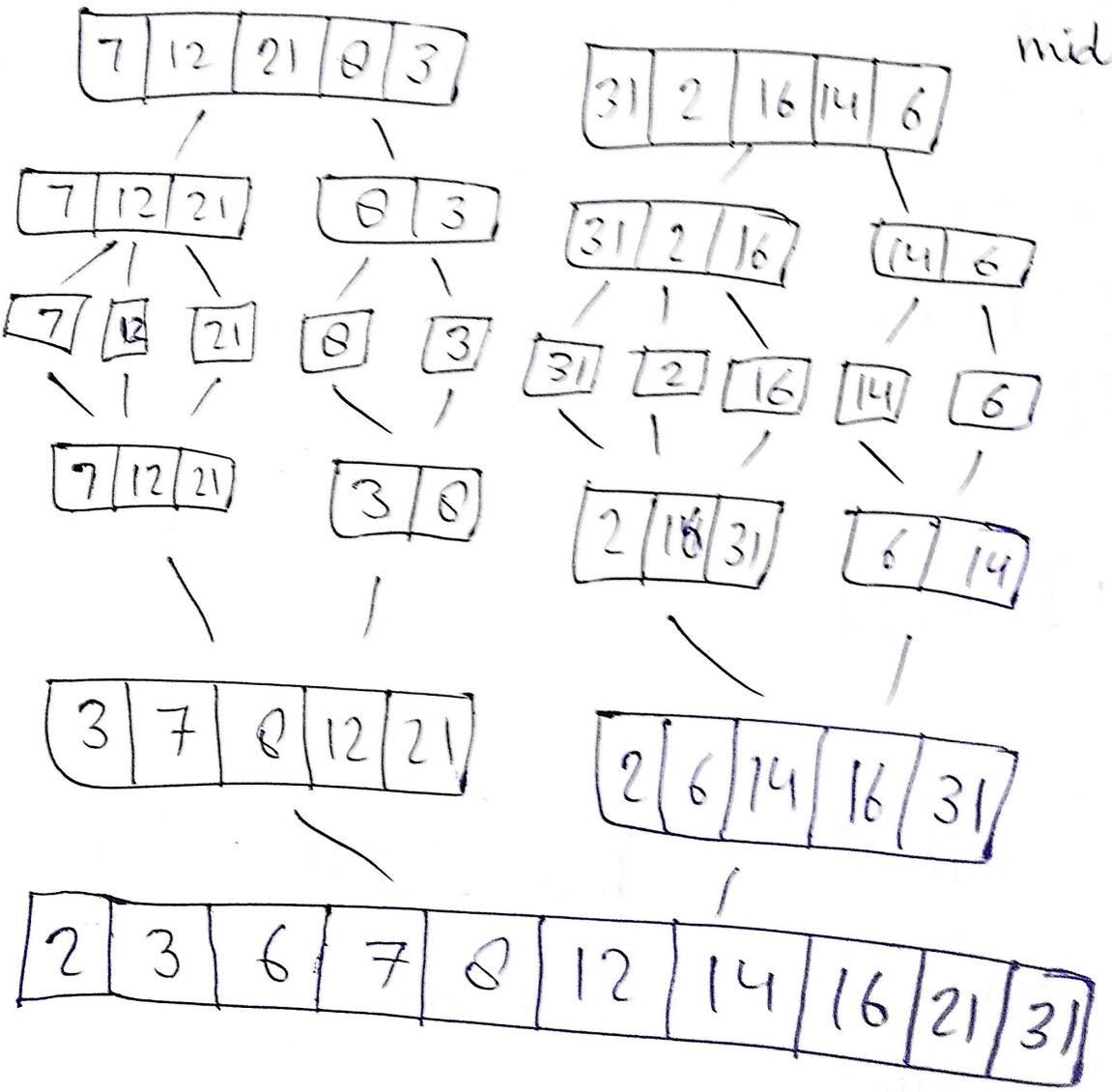
Note : Merge Sort apply in a long array data

Q. 7 12 21 0 3 31 2 16 14 6 → Apply Merge Sort

mid

$mid = \frac{l+h}{2} = \frac{1+10}{2}$

$= \frac{11}{2} \Rightarrow 5$



Q- $f(n) = n$ ...
$g(n) = n - 10$ Find out c & n_0.
$f(n) = O(g(n))$

Soln: $F(n) \leq c \cdot g(n)$ (∴ For same value $c > 0$ & $n > n_0$)

$F(n) \leq c \cdot g(n)$

$n + 10 \leq 2(n-10)$ c

~~$20 \leq 2n + c$~~ $n+10 \leq 2n - 20$ (∵ Let $c = 2$)

$30 + 10 \leq 2(30) - 20$

$40 \leq 40$

$n \geq 30$

$n_0 = 30$ $c = 2$

Q- $T(n) = \begin{cases} 1, & n = 1 \\ 4T(n/2) + n, & \text{where } n > 1 \end{cases}$

Soln: $O(n^2)$ Master & iterative

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

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

$T(n) = 4\left[4T(n/4) + n/2\right] + n$

$= 4^2 T\left(\frac{n}{4}\right) + 2n + n$

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

$T(n) = 4\left[4T(n/8) + n/8\right] + n/2$

$= 4^2 T\left(\frac{n}{8}\right) + \frac{n}{2} + \frac{n}{2}$

$T(n) = 4^2 T\left(\frac{n}{2^k}\right) + \frac{2^k - 1}{2^{k-1}}$

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

$n = 2^k$

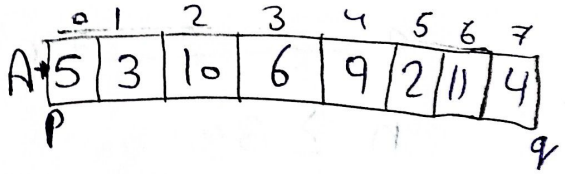
$k = \log_2 n$

$T(n) = 16\, T\left(\frac{n}{2^{\log n}}\right) + \frac{2^{\log n} - 1}{2^{\log n - 1}} = 16\, T\left(\frac{n}{n}\right) + \frac{n-1}{2^{n-1}}$ $T(n) = O(n \log n)$

Quick Sort :-

→ Choose a pivot element
→ Partition of array
→ Recursively sort the subarray


```
partition (A, p, q)
{
  x = A[p]
  i = p
  for (j = p+1 , j ≤ q , j++)
  {
    if (A[j] ≤ x)
    {
      i = i+1;
      swap (A[i] , A[j])
    }
  }
  swap (A[i], A[p])
}
```

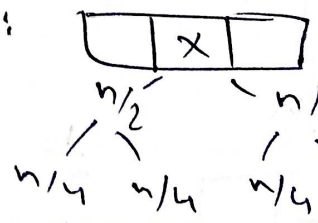



```
Quick Sort (A, p, q) {
  if (p < q) {
  m = partition (A, p, q)
    QuickSort (A, p, m-1)
    QuickSort (A, m+1, q)
  }
}
```

$T(n) = O(n \log_2 n)$

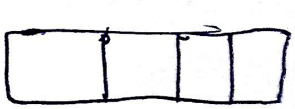
$T(n) = T(n-1) + n \Rightarrow O(n^2)$

Best Case:



Best Case $T(n) = O(n \log n)$

Worst Case:



Both sides not equal

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

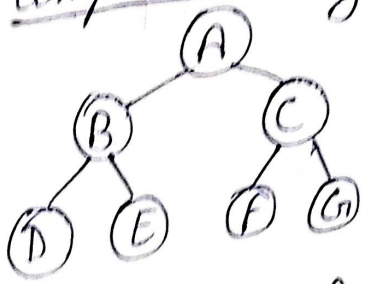
$T(n) \Rightarrow O(n^2)$

Heap Tree:

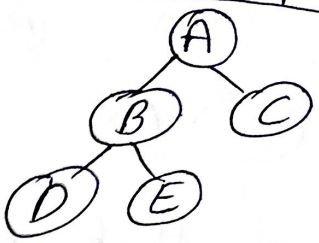
Heap Tree:

CBT
ACBT

Complete Binary Tree:



Almost Complete Binary Tree:



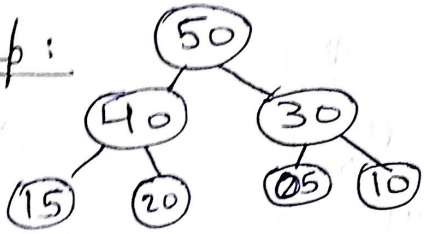
1	2	3	4	5	6	7
A	B	C	D	E	F	G

Left Child : $2i$

Right Child : $2i+1$

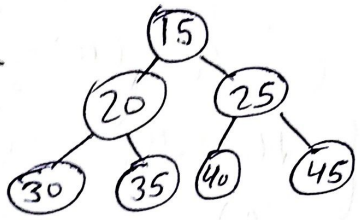
Parent (i) : $\left[\frac{i}{2}\right]$

1- Max Heap:



1	2	3	4	5	6	7
50	40	30	15	20	05	10

2- Min Heap:



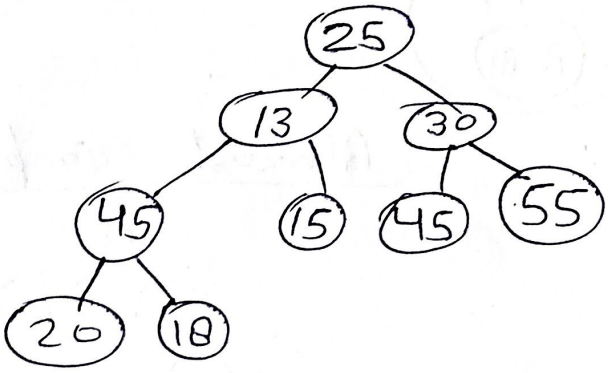
1	2	3	4	5	6	7
15	20	25	30	35	40	45

* Max - Heapify (A, i) :-

```
1. l = left (i)          (∵ 2i)
2 r = right (i)          (∵ 2i+1)
3 if (l ≤ heap.size and A[l] > A[i])
4 largest = l
5 else largest = i
6 if (r ≤ A.heap.size and A[r] > A[largest])
7 largest = r
8 if largest = i
9 Exchange (A[i], A[largest])
10 Max-Heapify (A, largest)
```

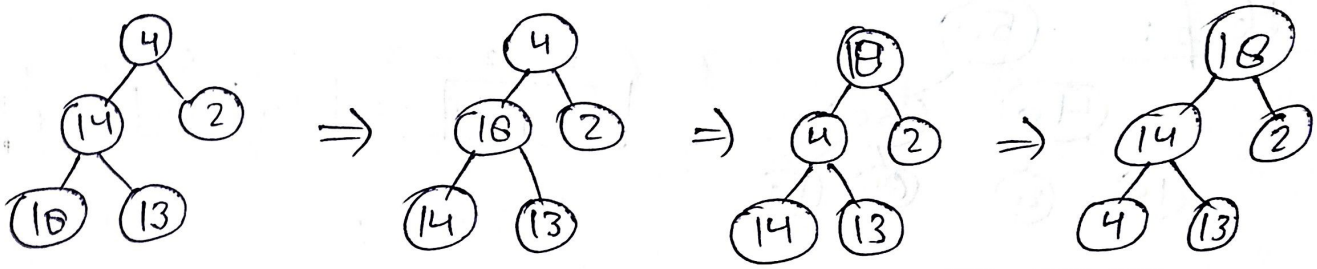
$$T(n) = O(\log_2 n)$$

Q: 25, 13, 30, 45, 15, 20, 18, 45, 55

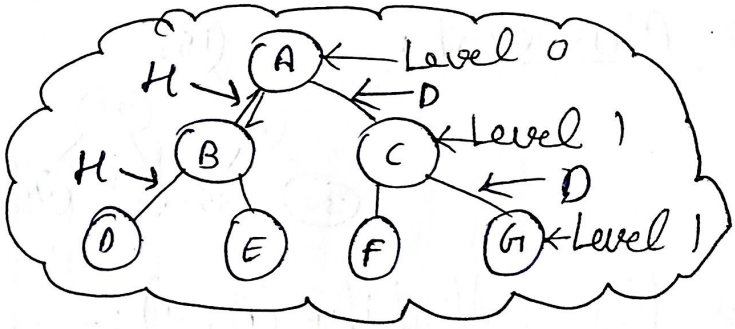


Build Max-Heap (A):-

1- A.Size = A.Length
2- For $i = \left[\frac{A.Length}{2}\right]$ to 1
3- Max-Heapify (A, i)



Height = Bottom to top
Depth = Top to Bottom
Level = 3



Note : $H = D = L$
NNode = $2^{h+1} - 1$
Leaf Node = 2^h
Internal Node = $2^h - 1$

$\left[\frac{n}{2^h+1}\right] = \left[\frac{15}{2^{0+1}}\right] = 7.5 = 8$

$\left[\frac{15}{2^{1+1}}\right] = \left[\frac{15}{4}\right] = 3.4 = 4$

$\left[\frac{15}{2^{2+1}}\right] = \left[\frac{15}{8}\right] = 1.875 = 2$

$$\sum_{h=0}^{\log_2 n} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h)$$

$$\sum_{h=0}^{\log n} \left\lceil \frac{n}{2^h \times 2} \right\rceil \times h = \frac{n}{2} \sum_{h=0}^{\log_2 n} \frac{h}{2^h}$$

$$\Rightarrow \frac{n}{2} \sum_{h=0}^{\infty} \frac{h}{2^h} = 2 \quad \Rightarrow \frac{n}{2} \times 2$$

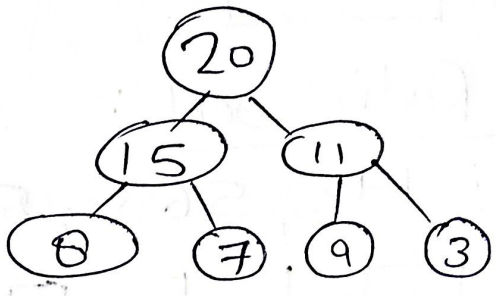
Build Max-Heap

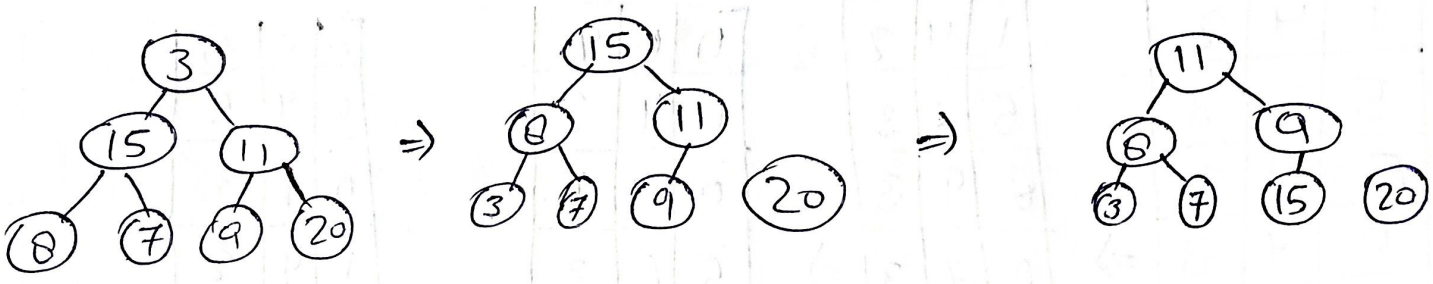
$T(n) \Rightarrow O(n)$

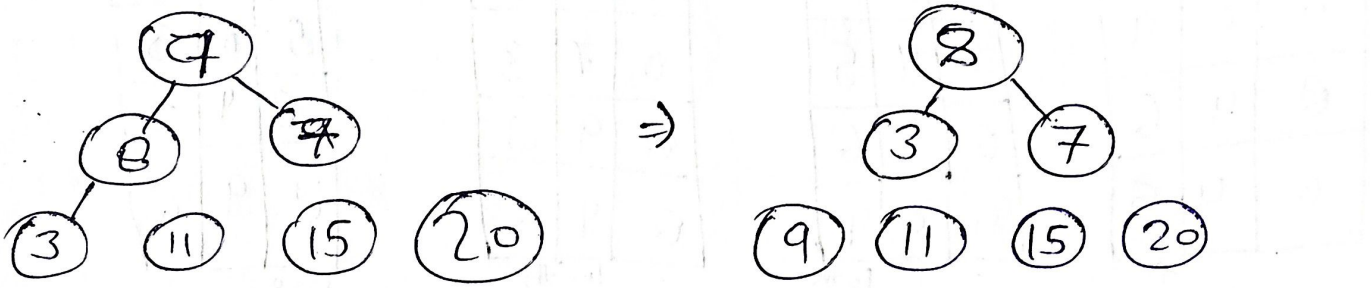
$S(n) \Rightarrow O(1)$

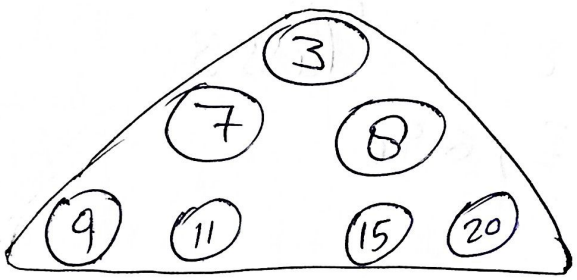
Heap-Sort (A) :-

1. Build Max-Heap (A)
2. For i = A.Length to 2 {
3. Exchange A[1] with A[i]
4. A.heap-Size = A.heap-Size - 1
5. Max Heapify (A, 1)









$$= O(n) + O(n-1) + O(n \log n)$$

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

Heap Sort

Linear time Sorting Method :-

⇒ Counting Sort :

→ Given Input Size n

→ Take range K (1 ... K)

Q: 2, 1, 2, 3, 1, 3, 2, 4, 5, 4 → Highest element + 1 = n

⇒ 1 to K

xx	2	1
xxx	3	2
xx	2	3
xx	2	4
x	1	5
		6

No of Occurence in particular element

⇒ 1, 1, 2, 2, 2, 3, 3, 4, 4, 5

$T(n) \Rightarrow O(n+k)$

⇒ Radix Sort :

Eg: 142, 563, 893, 73, 9, 896, 45

1	4	2
5	6	3
8	9	3
0	7	3
0	0	9
8	9	6
0	4	5

↑ Unit

⇒

1	4	2
5	6	3
8	9	3
0	7	3
0	4	5
8	9	6
0	0	9

10th Pos

Pass 1

⇒

0	0	9
1	4	2
0	4	5
5	6	3
0	7	3
8	9	3
8	9	6

100th Pos

Pass 2

0	0	9
0	4	5
0	7	3
1	4	2
5	6	3
8	9	3
8	9	6

1000 Pos

Pass 3

9, 45, 73, 142, 563, 893, 896

$T(n) = O(nk)$

Bucket Sort

It works based on Floating Point Number b/w 0.0 to 1.0

Eg: 0.73, 0.13, 0.64, 0.39, 0.20, 0.89, 0.53, 0.42, 0.06, 0.94

Length of Bucket × Key ⇒ 10 × 0.73 = 7.3

0	0.06
1	0.13
2	0.20
3	0.39
4	0.42
5	0.53
6	0.64
7	0.73
8	0.89
9	0.94

Case 1: $T(n) = O(n)$

Case 2: 0.39, 0.31, 0.34, 0.33

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

⇒ Shell Sort: (modify of insertion sort)

0	1	2	3	4	5	6	7	8
23	29	15	19	31	7	9	5	2

$Gap = \left\lfloor \frac{0+8}{2} \right\rfloor = 4$

23	29	15	19	31	7	9	5	2
23	7	15	19	31	29	9	5	2
23	7	9	19	31	29	15	5	2
23	7	9	5	31	29	15	19	2
23	7	9	5	2	29	15	19	31
2	7	9	5	23	29	15	19	31

Pass 2: $Gap = \left\lfloor \frac{0+4}{2} \right\rfloor = 2$

2	5	9	7	23	29	15	19	31
2	5	9	7	15	29	23	19	31
2	5	9	7	15	19	23	29	31

Pass 3: $Gap = \left\lfloor \frac{0+2}{2} \right\rfloor = 1$

2	5	7	9	15	19	23	29	31
2	5	7	9	15	19	23	29	31

Time Complexity -

$T(n) = O(n^{1.2} \text{ to } n^2)$

Worst case: $O(n^2)$

[∵ Gap High, T(n) low]

Stable & Non-Stable Sorting:

If a Sorting Algo, after the sorting the elements does not change the sequence of similiar element in which they appear in array, it is k.a. Sorting Stable Technique. Otherwise, this is a unstable.

⇒ Stable Sort: Insertion Sort, Bubble Sort Merge Sort, Counting Sort, Radix Sort, Bucket Sort.

⇒ Unstable Sort: Quick Sort, Selection Sort Heap Sort, Shell Sort

Inplace and Outplace Sorting Method:-

Sorting technique may require some extra space for comparision & of temporary storage for few data elements.

⇒ Inplace → Bubble, Insert, Selection, Quick, Heap Sort

⇒ Outplace → Merge Sort, Bucket, Counting and Radix Sort

E.g. → (Stable) $3, 2^A, 2^B, 5$ (Bubble Sort)

$2^A, 3, 2^B, 5$

⇒ $2^A, 2^B, 3, 5$ } Pass 1