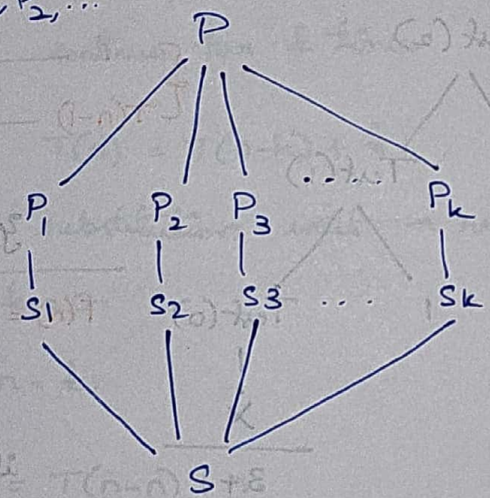
Strategies for solving a problem :-

* Greedy method
* Dynamic Programming
* Backtracking
* Branch and Bound
* Divide and Conquer.

DIVIDE AND CONQUER

A large problem of size n — P can be broken down to sub-problems $P_1, P_2, \ldots$



Combine the solutions of sub problems to get final solution.

Sub-problems is same as problem.

For example, if major problem is sort, sub problem is also sort.

So, the nature of the solution is recursive.

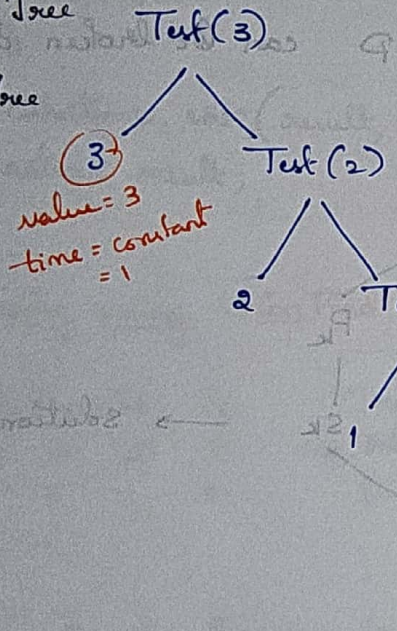
```
DAC(P) {
    if (small(P)) {
        S(P);
    } else {
        divide P into P1, P2, P3, ... Pk
        apply DAC(P1), DAC(P2), ...
        combine (DAC(P1), DAC(P2) ... )
    }
}
```

Problems :-

1. Binary Search
2. Finding Maximum and Minimum
3. Merge Sort
4. Quick Sort
5. Strassen's Matrix Multiplication.

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

Recursive Tree (or) Tracing Tree



$T(n)$ — void Test (int n) {

if (n>0) {

1 —— print n

$T(n-1)$ —— Test (n-1)

}

}

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

$3+1$

$n+1$ call.

$f(n) = n \Rightarrow O(n)$

If we include condition it will add +1 to the recurrence relation.

You can simply $T(n-1)+1$ instead of $T(n-1)+2$.

You can also write 'c'. $[T(n-1)+c]$.

Recurrence Relation:

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

Solve Recurrence Relation by Substitution method :

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

$T(n) = T(n-1) + 1$ — ① — 1st

$T(n-1) = T(n-2) + 1$ — ②

Substitute ② in ①,

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

$= T(n-2) + 2$ — ③ — 2nd

$T(n-2) = T(n-3) + 1$ — ④

Substitute ④ in ③,

$T(n) = T(n-3) + 3$ — 3rd

⋮

continue for k times.

$T(n) = T(n-k) + k.$ — k^{th}.

We continue substitution until n reaches to 0.

Assume $n - k = 0$.

$\therefore n = k.$

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

$= T(0) + n$

$T(n) = 1 + n. \Rightarrow \underline{\Theta(n)}$

$T(n) = T(n-1) + 1$ — ① — 1st

$T(n-1) = T(n-2) + 1$ — ②

Substitute ② in ①,

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

$= T(n-2) + 2$ — ③ — 2nd

$T(n-2) = T(n-3) + 1$ — ④

Substitute ④ in ③,

$T(n) = T(n-3) + 3$ — 3rd

⋮

continue for k times.

$T(n) = T(n-k) + k$. — k^{th}.

We continue substitution until n reaches to 0.

Assume $n - k = 0$.

$\therefore n = k$.

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

$= T(0) + n$

$T(n) = 1 + n \Rightarrow \underline{\Theta(n)}$.

Recurrence Relation Decreasing function $T(n) = T(n-1) + n$

Recurrence Relation :-

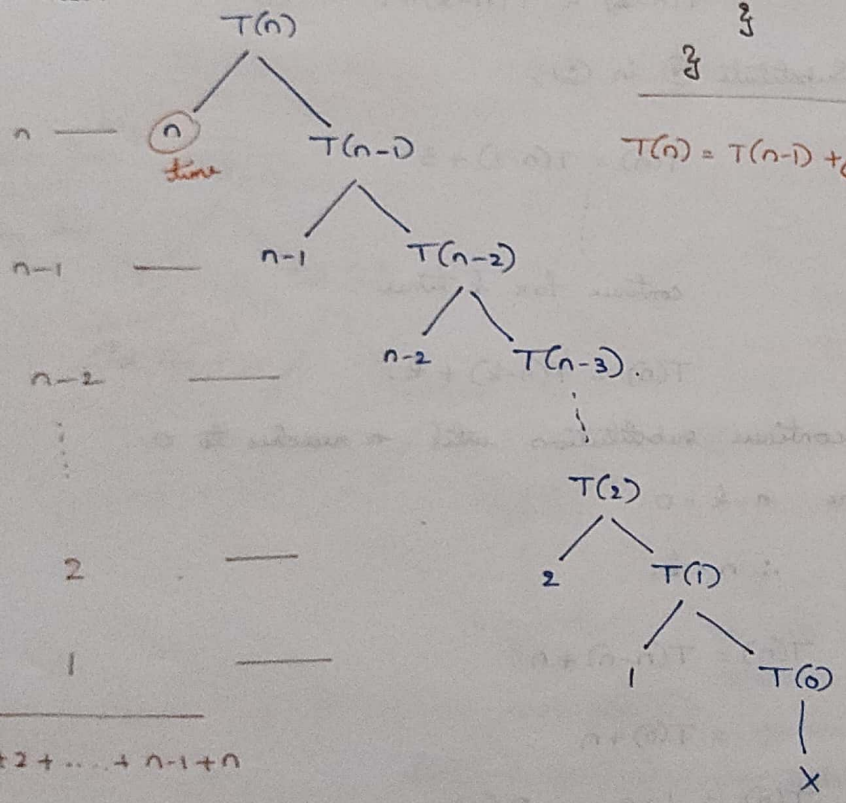
$$T(n) = T(n-1) + n$$

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

```
T(n) —  void Test(int n) {
1    ——   if(n>0) {
n+1  ——   for(i=0; i<n; i++)
n    ——   print n
T(n-1) —  Test(n-1)
        }
      }
```

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

Recursion Tree :-



$1 + 2 + \ldots + n-1 + n$

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

$$T(n) = \frac{n(n+1)}{2} = \underline{\Theta(n^2)}.$$

Back subst

Recurrence

a

Back substitution & Induction method:

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

$T(n) = T(n-1) + n$ ——— 1st

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

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

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

$= T(n-2) + (n-1) + n$ ——— 2nd

↳ avoid adding and making it $(2n-1)$.

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

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

$= T(n-3) + (n-2) + (n-1) + n$ ——— 3rd.

$\vdots$

$\vdots$

$= T(n-k) + (n-(k-1)) + (n-(k-2)) + \dots + n.$

assume $n - k = 0$

$k = n.$

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

$= T(0) + (1 + 2 + \dots + n)$

$T(0) = 1$

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

$\therefore T(n) = \frac{n(n+1)}{2} = \Theta(n^2).$

Recurrence Relation $T(n) = T(n-1) + \log n$

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

$T(n)$ —— void Test(int n) {

if(n>0) {

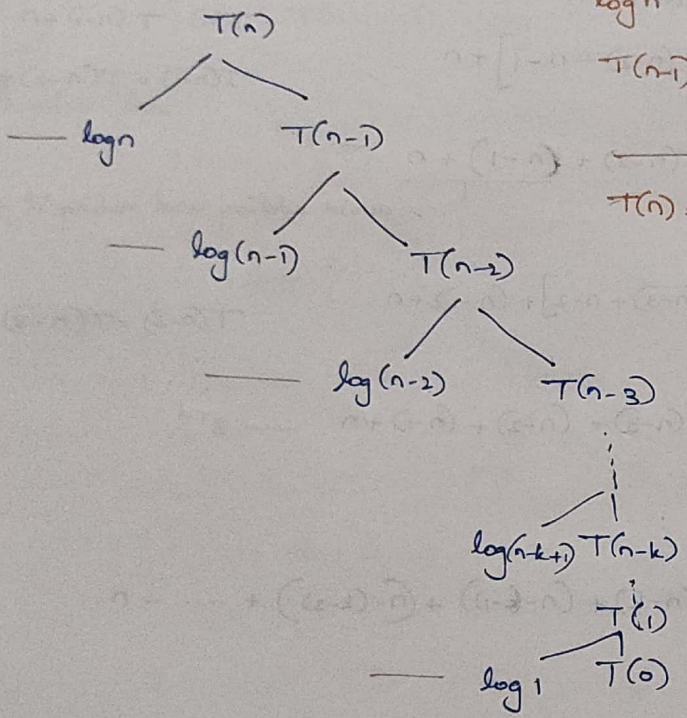
for(i=1; i<n; i*=2)

$\log n$ —— print i

$T(n-1)$ —— Test(n-1)

}

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



$\log n + \log(n-1) + \dots + \log 2 + \log 1$

$= \log[n \cdot (n-1) \cdot (n-2) \dots 2 \cdot 1]$

$= \log(n!)$

upper bound of $n! = n^n$.

- $O(\log n^n)$.
- $O(n \log n)$.

Induction

Induction & Substitution method :-

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

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

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

$$= [T(n-3) + \log(n-2)] + \log(n-1) + \log n.$$

$$\vdots$$

$$= T(n-k) + \log(n-(k-1)) + \ldots \log(n-1) + \log n.$$

$n - k = 0$

$n = k$

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

$$= T(0) + \log n!$$

$$= 1 + \log n!$$

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

$T(n) = T(n-1) + (1)^{\times n}$ — $O(n)$

$T(n) = T(n-1) + (n)^{\times n}$ — $O(n^2)$

$T(n) = T(n-1) + (\log n)^{\times n}$ — $O(n \log n)$

$T(n) = T(n-1) + (n^2)^{\times n}$ — $O(n^3)$.

$T(n) = T(n-2) + 1$ — $\frac{n}{2}\, O(n)$.

$T(n) = T(n-100) + n$ — $O(n^2)$.

$T(n) = 2T(n-1) + 1$ — ?

Back

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

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

$T(n)$ — Algorithm Test (int n) {

if (n>0) {

1 ——— print n

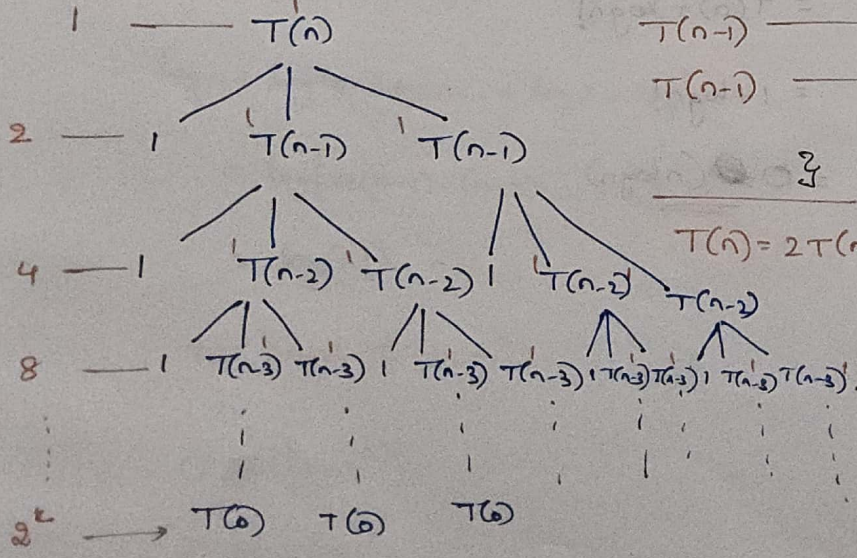
$T(n-1)$ ——— Test (n-1)

$T(n-1)$ ——— Test (n-1)

}

}

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



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

$a + ar + ar^2 + \ldots + ar^k = \frac{a(r^{k+1} - 1)}{r-1}$ $[a=1,\ r=2]$

Assume $n - k = 0 \Rightarrow k = n$

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

$= \underline{\Theta(2^n)}$.

Back Substitution method :-

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

$T(n) = 2T(n-1) + 1$ ——— (1)

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

$= 2^2 T(n-2) + 2 + 1$ ——— (2)

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

$= 2^3 T(n-3) + 2^2 + 2 + 1$ ——— (3)

⋮

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

$n - k = 0 \Rightarrow k = n$.

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

$= \underline{2^{n+1} - 1}$.

$= \underline{\Theta(2^n)}$.

Masters Theorem for Decreasing Functions

$T(n) = T(n-1) + 1$ — $O(n)$

$T(n) = T(n-1) + n$ — $O(n^2)$

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

$T(n) = 2T(n-1) + 1$ — $O(2^n)$.

$T(n) = 3T(n-1) + 1$ — $O(3^n)$

$T(n) = 2T(n-1) + n$ — $O(n 2^n)$.

General form of Recurrence Relation:-

$T(n) = aT(n-b) + f(n)$. [from above equations].

$a > 0$ $b > 0$ and $f(n) = O(n^k)$ where $k \geq 0$.

Case ①:

If $a = 1$ $O(n^{k+1})$ [∵ as $f(n)$ is multiplied by $n = n * f(n)$].

$O(n * f(n))$.

Case ②:

If $a > 1$ $O(a^{n/b} f(n))$. $= O(a^{n/b} f(n))$.

$O(n^k a^{n/b})$. $= O(n^k a^{n/b})$.

$\Rightarrow$ [If $T(n) = 2T(n-2) + 1$ then $O(2^{n/2})$.

Case ③:

If $a < 1$ $O(n^k)$

$O(f(n))$.

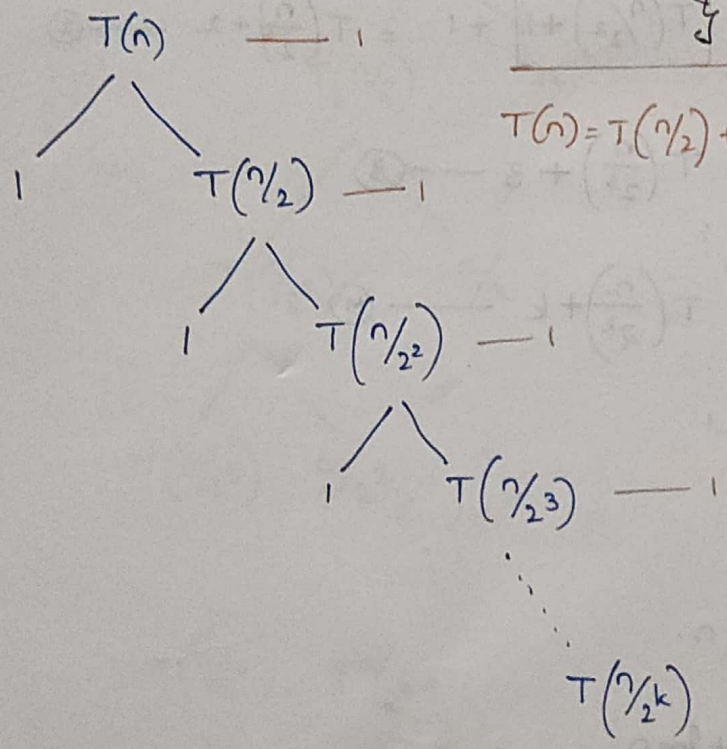
Recurrence Relation Dividing Function $T(n) = T(n/2) + 1$

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

```
T(n) ——— Algorithm Test(int n) {
            if (n>1) {
1 ———————      print n
T(n/2) ———————  Test(n/2)
            }
}
```

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

Recursion Tree method:-



Let's say, $\frac{n}{2^k} = 1$ & recursion stopped. $\underline{k \text{ steps}}$ → Total time.

$n = 2^k$

$k = \log_2 n$

$\underline{\Theta(\log n)}$.

Substitution Method :-

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

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

$$= \left[T\left(n/2^2\right) + 1\right] + 1 = T\left(\frac{n}{2^2}\right) + 2 \quad \text{—— (2)}$$

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

$$\vdots$$

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

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

$$n = 2^k$$

$$k = \log n.$$

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

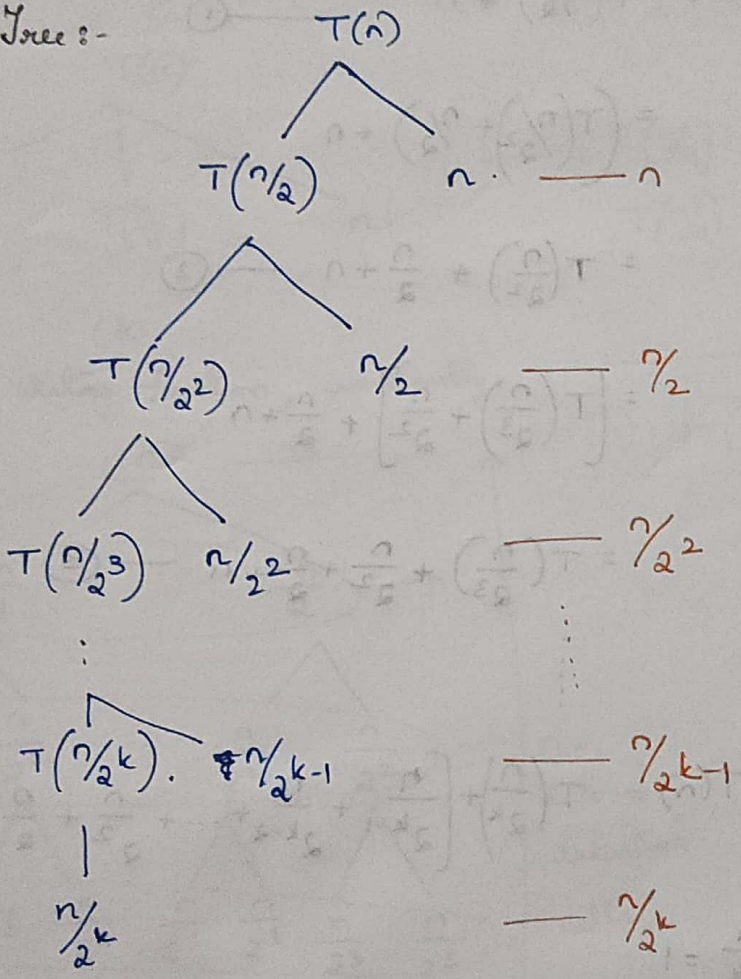
$$= 1 + \log n$$

$$\Theta(\log n)$$

Recurrence Relation Dividing $T(n) = T(n/2) + n$.

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

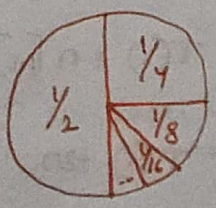
Recursion Tree :-



$$T(n) = n + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \dots + \frac{n}{2^k}$$

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

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



$\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \dots = 1.$

$T(n) = n$

$\therefore \Theta(n).$

Substitution method:-

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

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

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

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

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

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

$$\vdots$$

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

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

$k = \log n$

$$T(n) = T(1) + n\left[\frac{1}{2^{\log_k - 1}} + \frac{1}{2^{\log_k - 2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1\right]$$

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

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

$\Theta(n)$

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

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

$T(n)$ — `void Test(int n) {`

`if(n>1) {`

`for(i=0; i<n; i++){`

n — `stmt;`

`}`

$T(n/2)$ — `Test(n/2);`

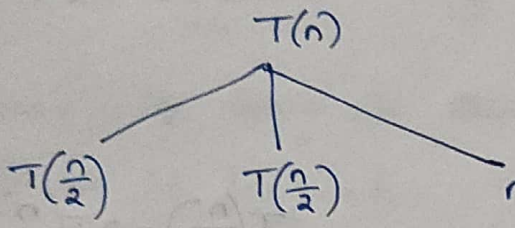
$T(n/2)$ — `Test(n/2);`

`}`

`}`

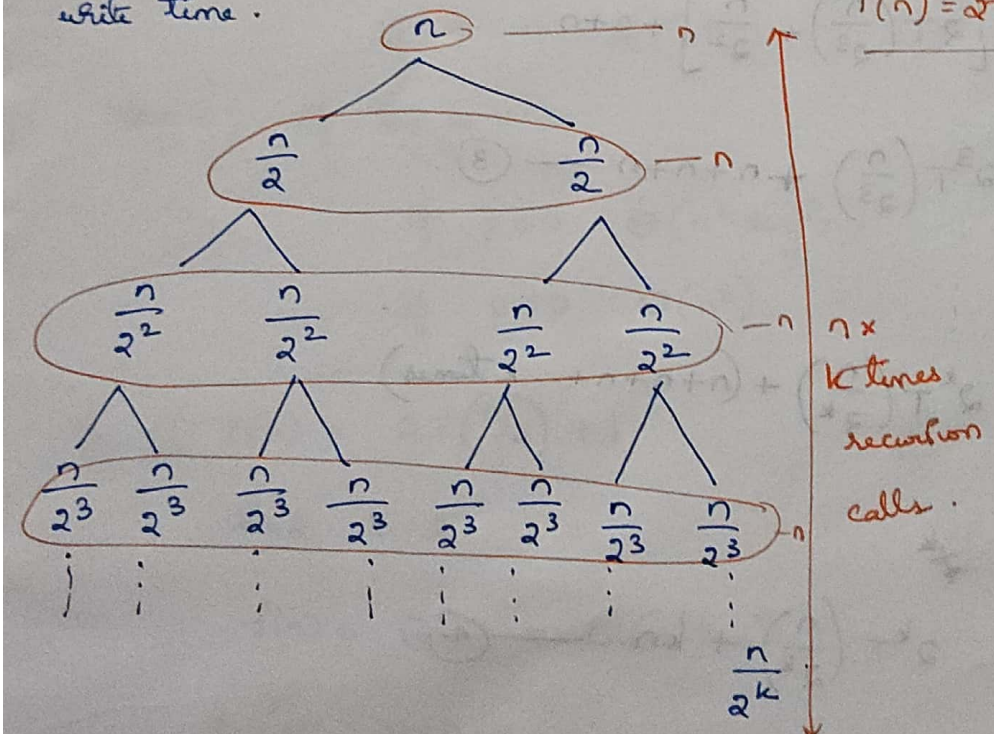
$T(n) = 2T(n/2) + n$

Recursion Tree Method :-



(2)

Instead of writing function, we directly write time.



$n * k$ times.

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

$k = \log n$.

$O(n \log n)$

Back Substitution Method :-

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

$T(n) = 2T(n/2) + n$ ——— ①

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

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

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

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

$\vdots$

$2^k T\left(\frac{n}{2^k}\right) + (n + n + n + \ldots \text{ k times})$

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

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

$k = \log n.$

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

$= n + n\log n$

$O(n\log n)$ & $\Theta(n\log n)$.

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

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

Masters Theorem in Algorithms for Dividing Function

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$a \geq 1 \quad b > 1 \quad f(n) = \Theta(n^k \log^p n)$.

We will find two values.

① $\log_b a$ ② k.

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

Case 2: if $\log_b a = k$

if $p > -1$ $\Theta(n^k \log^{p+1} n)$.

if $p = -1$ $\Theta(n^k \log \log n)$

if $p < -1$ $\Theta(n^k)$.

Case 3: if $\log_b a < k$

if $p \geq 0$ $\Theta(n^k \log^p n)$.

if $p < 0$ $\Theta(n^k)$.

Eg:- ① $T(n) = 2T(n/2) + 1$

$a = 2 \quad b = 2$.

$f(n) = \Theta(1) = \Theta(n^0 \log^0 n)$

$k = 0 \quad p = 0$.

① $\log_b a$ ② $k = 0$.

$= \log_2 2$

$= 1$

Case ① $\Theta(n^{0+1}) = \underline{\Theta(n)}$.

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

① $\log_2 4 = 2$ ② $k = 1$.

$p = 0$.

$2 > 1$

Case ①

$\underline{\Theta(n^2)}$.

③ $T(n) = 8T\left(\frac{n}{2}\right) + n$.

① $\log_2 8 = 3$. ② $k=1$ $p=0$.

$3 > 1$

Case ① $\Rightarrow \underline{\Theta(n^3)}$.

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

① $\log_b a = 3$ ② $k=2$ $p=0$

$3 > 2$

Case ① $\Rightarrow \underline{\Theta(n^3)}$

⑤ $T(n) = 9T\left(\frac{n}{3}\right) + 1$

① $\log_b a = 2$ ② $k=0, p=0$

$2 > 0$.

Case ① $\Rightarrow \underline{\Theta(n^2)}$.

⑥ $T(n) = 9T\left(\frac{n}{3}\right) + n$

① 2 ② $k=1$

$2 > 1$

$\underline{\Theta(n^2)}$.

⑦ $T(n) = 9T\left(\frac{n}{3}\right) + n^2$.

$\log_b a = 2 \mid 2 = 2. \mid p=0, k=2$

Case ② $\Rightarrow \Theta(n^2 \log^1 n)$

$= \underline{\Theta(n^2 \log n)}$

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

$\log_b a = 2$ $k=1, p=0$.

Case ① $\Rightarrow \underline{\Theta(n^2)}$.

⑨ $T(n) = 8T\left(\frac{n}{2}\right) + n \log n$

$\log_b a = 3$ $k=1, p=1$

$3 > 1$

Case ① $\Rightarrow \underline{\Theta(n^3)}$.

As long as, $(\log_b a) >$ (power of n (or k)), answer is $\underline{\Theta(n^{\log_b a})}$.

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

$\log_2 2 = 1 \quad k=1 \quad p=0.$

$\log_b a = k \Rightarrow$ Case (2) $\quad p > -1$

$\Theta(n^k \log^{p+1} n) = \underline{\Theta(n \log n)}.$

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

$\log_2 4 = 2 \quad k=2 \quad p=0$

case (2).

$\Theta(n^2 \log n)$

If $\log_b a = k$, then multiply 2nd term $(f(n))$ with $\log n$ & $p > -1$

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

$\underline{\Theta(n^2 \log^2 n)}.$

(13) $T(n) = 8T\left(\frac{n}{2}\right) + n^3$

$\underline{\Theta(n^3 \log n)}$

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

$\log_2 2 = 1 \quad k=1 \quad p=-1.$

$\underline{\Theta(n \log\log n)}.$

If $p=-1$ & $\log_b a = k$ (case 2), then multiply n^k with $\log\log n$

If $p<-1$ & case 2, then $\Theta(n^k)$.

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

$\log_2 2 = 1 \quad k=1 \quad p=-2. \ (p<-1).$

$\underline{\Theta(n)}.$

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

$\log_2 1 < 2$. $p = 0$.

$\Theta(n^2)$.

If case (3) or $\log_b a < k$

$p \geq 0$, 2nd term as it is.

$p < 0$, only n^k.

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

$\log_2 2 = 1 < k = 2$, $p = 2$

$\Theta(n^2 \log^2 n)$.

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

$\log_b a = 2$, $k = 3$. $p = -1$

$\log_b a \leq k \Rightarrow$ case (3).

$\Theta(n^3)$.

Examples for Master Theorem

Case ①:-

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

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

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

$T(n) = 8T\left(\frac{n}{2}\right) + n^2 \Rightarrow \Theta(n^3)$.

$T(n) = 16T\left(\frac{n}{2}\right) + n^2 \Rightarrow \Theta(n^4)$

$\log_b^a > k$

$\Theta(n^{\log_b^a})$.

Case ③:-

$T(n) = T\left(\frac{n}{2}\right) + n \Rightarrow \Theta(n)$.

$T(n) = 2T\left(\frac{n}{2}\right) + n^2 \Rightarrow \Theta(n^2)$.

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

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

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

Case ②:-

$T(n) = T\left(\frac{n}{2}\right) + 1 \Rightarrow \Theta(\log n)$.

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

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

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

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

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

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

Root Function (Recurrence Relation)

Recurrence Relation :-

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

$T(n)$ —— void Test (int n) {

if (n > 2) {

1 —— stmt,

$T(\sqrt{n})$ —— Test ($\sqrt{n}$);

}

}

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

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

$= T(n^{\frac{1}{2}}) + 1$ —— ①

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

$= T\left(n^{\frac{1}{2^3}}\right) + 3$ —— ③

⋮

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

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

$T(2^m) = T\left(\left(2^{m/2^k}\right)\right) + k$

$= T\left(2^{\frac{m}{2^k}}\right) + k.$

Assume $T\left(2^{\frac{m}{2^k}}\right) = T(2).$

$\frac{m}{2^k} = 1.$

$m = 2^k$ & $k = \log_2 m$

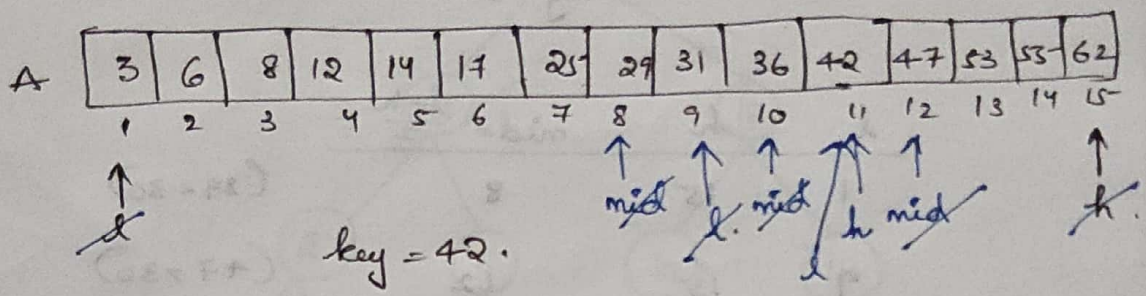
$= \log_2 \log_2 n$

$= \Theta(\log \log n)$

Binary Search Iterative Method

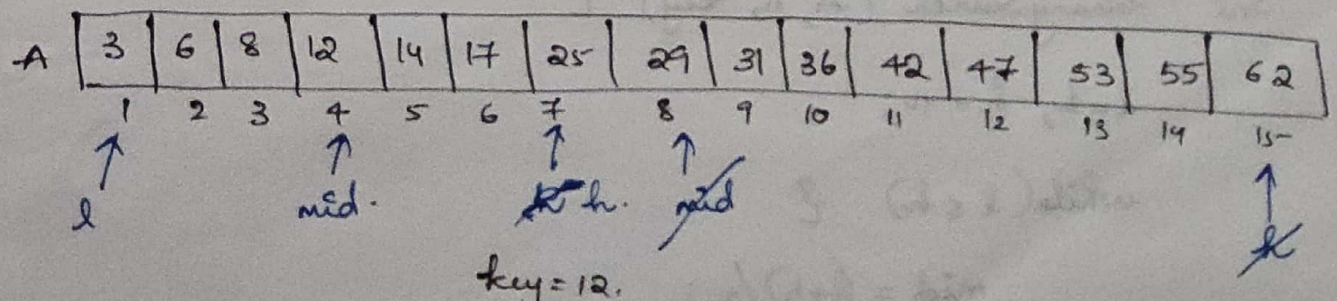
Binary Search follows Divide and Conquer Strategy.

To perform Binary Search, input must be in sorted order.



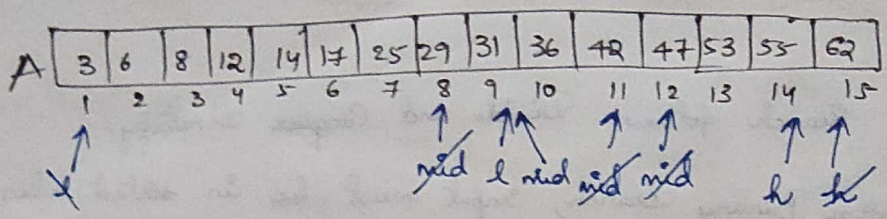
l	h	mid = $\left[\frac{l+h}{2}\right]$		
1	15	$\frac{1+15}{2} = 8$	(value = 29)	29 < 42
9	15	$\frac{9+15}{2} = \frac{24}{2} = 12$	(value = 47)	47 > 42
9	11	$\frac{9+11}{2} = 10$	(value = 36)	36 < 42
11	11	$\frac{11+11}{2} = 11$	(value = 42)	42 = 42

*



l	h	mid = $\frac{l+h}{2}$	
1	15	$\frac{1+15}{2} = 8$	(29 > 12)
1	7	$\frac{1+7}{2} = 4$	(12 = 12)

*



key = 30.

l	h	mid = $\frac{l+h}{2}$	
1	15	8	(29 < 30)
9	15	12	(47 > 30)
9	14	11	(42 > 30)
9	10	9	(31 > 30).
9	9	9	(31 > 30)
9	8	✗	

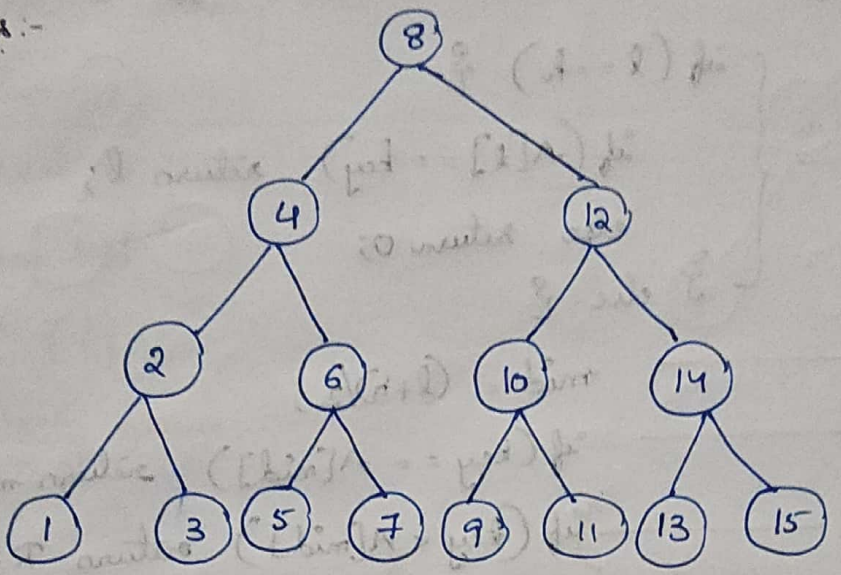
(l > h) → terminate
element not found.

Algorithm for Binary Search :-

```
int binarySearch (A, n, key) {
    l=1, h=n;
    while(l ≤ h) {
        mid = (l+h)/2;
        if (key == A[mid]) return mid;
        if (key < A[mid]) h = mid-1;
        else l = mid+1;
    }
    return 0;
}
```

A	3	6	8	12	14	17	25	29	31	36	42	47	53	55	62
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

mid values:-



logn.

Tree is multiplying at 4 levels ⇔ 4 comparisons.

$15+1 = 16$ $\quad \log_2 16 = 4$.

min — 1 $\quad O(1)$

Max — $\log n$ $\quad O(\log n)$.

Binary Search Recursive Method

$T(n)$ — Algorithm RBinSearch (l, h, key) {

1 — if (l == h) {
    if (A[l] == key) return l;
    else return 0;
  } else {
1 ——— mid = (l+h)/2;
1 ——— if (key == A[mid]) return mid;
$T(\frac{n}{2})$ ——— if (key < A[mid]) return RBinSearch(l, mid-1, key);
    else return RBinSearch(mid+1, h, key);
  }
}

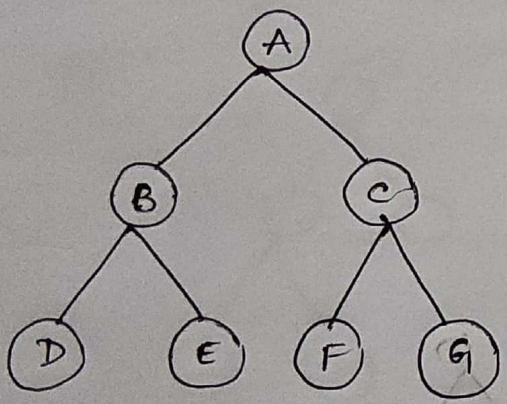
$$T(n) = \begin{cases} 1 & n = 1 \\ T\left(\frac{n}{2}\right) + 1 & n > 1 \end{cases}$$

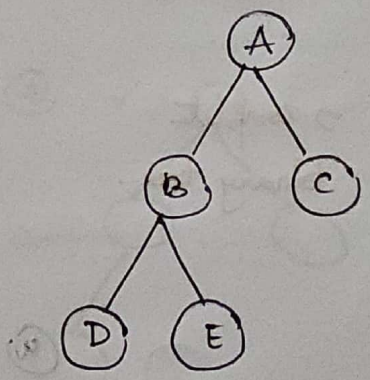
$\Theta(\log n)$ [from earlier chapters, above recurrence relation has time complexity of $\Theta(\log n)$]

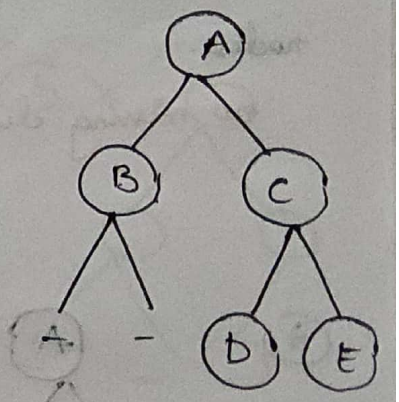
Heap - Heap Sort - Heapify - Priority Queues.

1. Array Representation of Binary Tree.
2. Complete Binary Tree.
3. Heap.
4. Insert & Delete.
5. Heap Sort.
6. Heapify.
7. Priority Queue.

Representation of a Binary Tree







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

A	B	C	D	E

A	B	C	-	-	D	E

if a node is at index - i

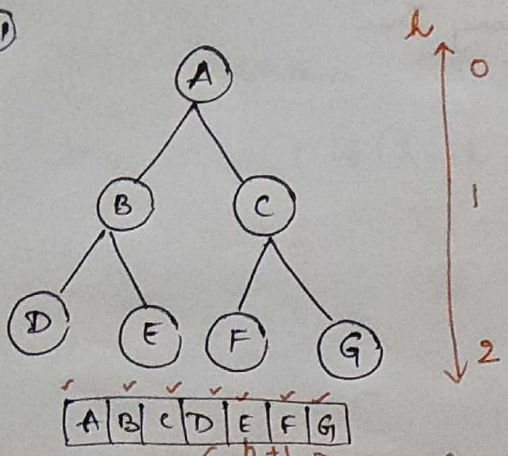
it's left child is at - 2 * i

it's right child is at - 2 * i + 1

it's parent is at - $\frac{i}{2}$

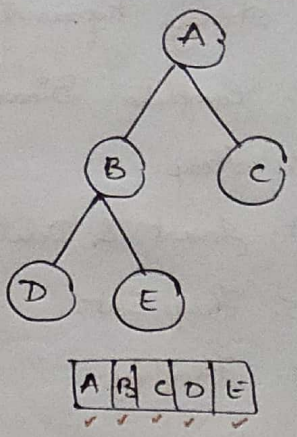
Full Complete Binary Tree

(1)



maximum: $(2^{h+1} - 1)$ nodes

(2)

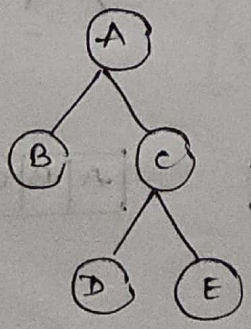


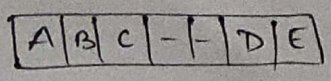
no missing elements ⇒ complete binary tree.

A full binary tree is a binary tree with maximum number of nodes.

no missing elements ⇒ complete binary tree.

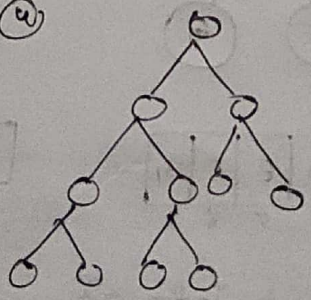
(3)





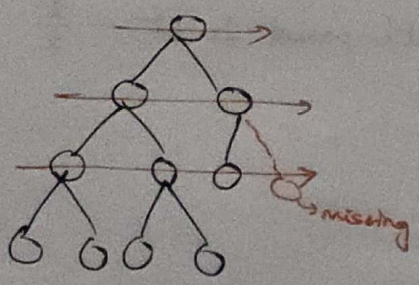
missing elements ⇒ not a complete binary tree.

(4)



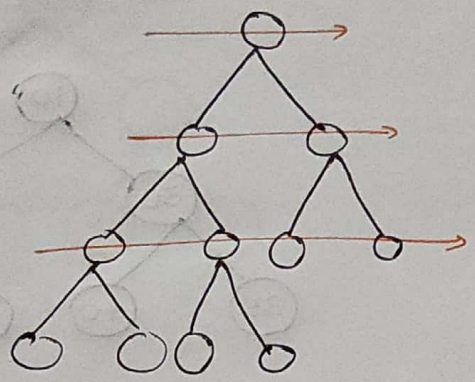
complete binary tree.

(5)



check level-by-level to find complete BT

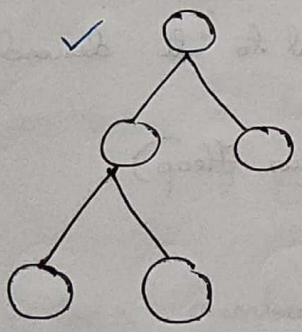
* A complete binary tree of height (h) is a full binary tree upto height (h-1) and in the last level, elements are filled from left to right.



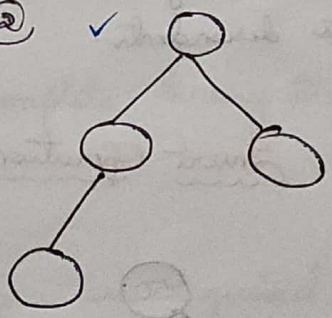
[full binary until h=2]

at h = 3; nodes filled from left to right.

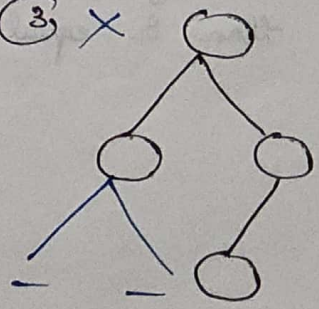
(1) ✓



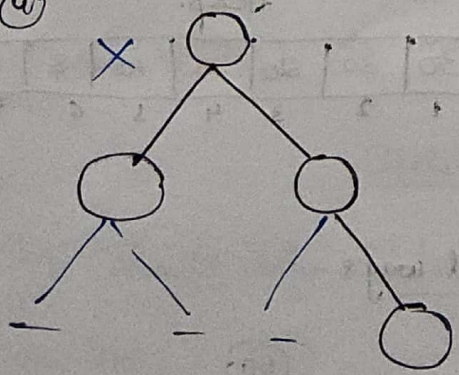
(2) ✓



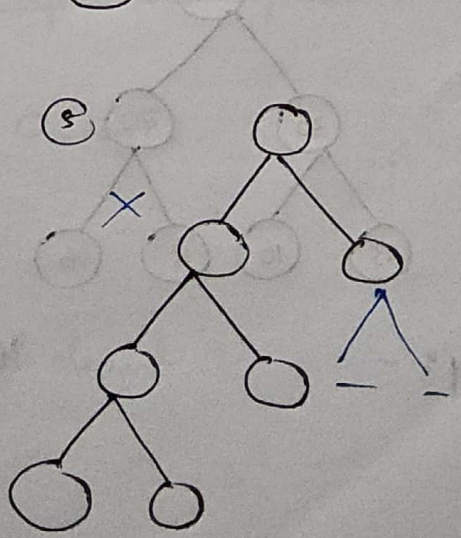
(3) ✗



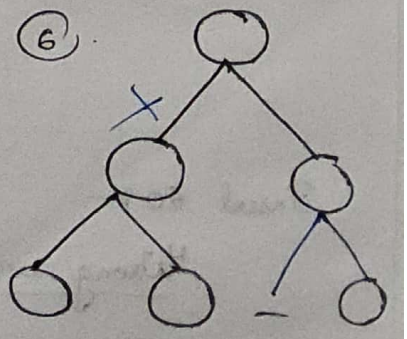
(4) ✗



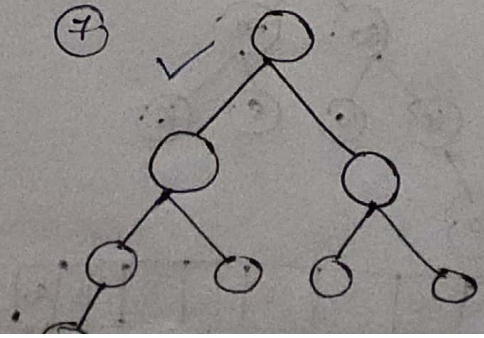
(5) ✗



(6) ✗



(7) ✓

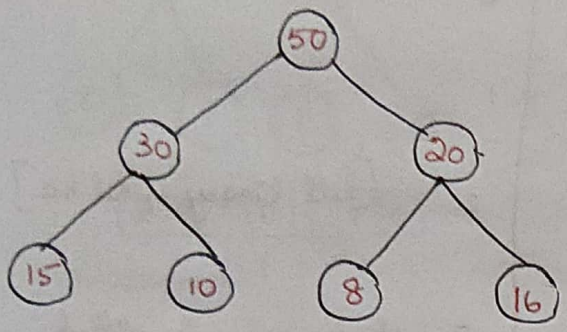


Always, The height of a complete binary tree is minimum $\Rightarrow \log n$.

Heap

Heap is a complete binary tree.

Max Heap

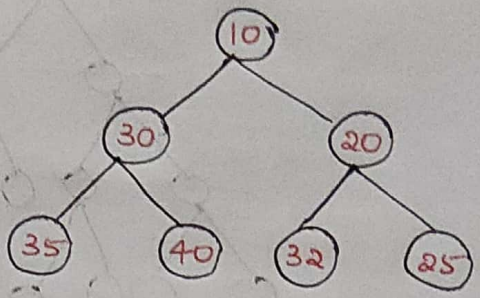


H

1	2	3	4	5	6	7
50	30	20	15	10	8	16

Every node has element larger than or equal to its descendents.

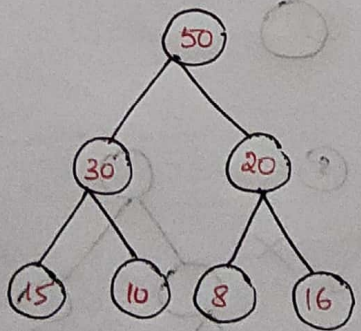
Min Heap



1	2	3	4	5	6	7
10	30	20	35	40	32	25

Every node has element smaller than or equal to its descendents.

Insert Operation (of Max Heap)

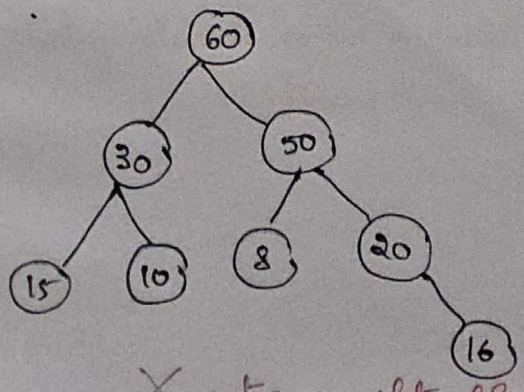


Heap.

1	2	3	4	5	6	7
50	30	20	15	10	8	16

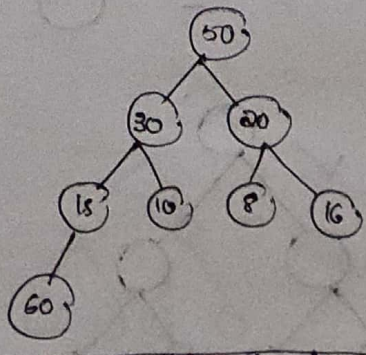
Insert 60 :-

Wrong way :-



✗ not a complete binary tree.

Correct way :-

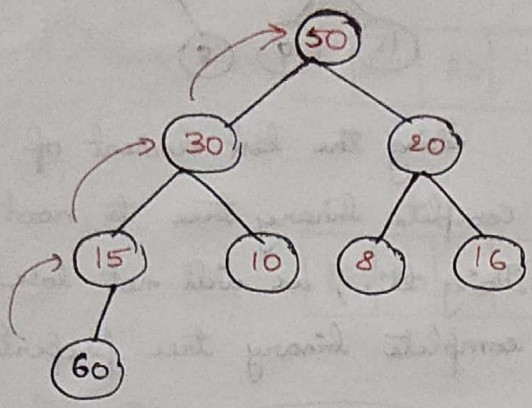


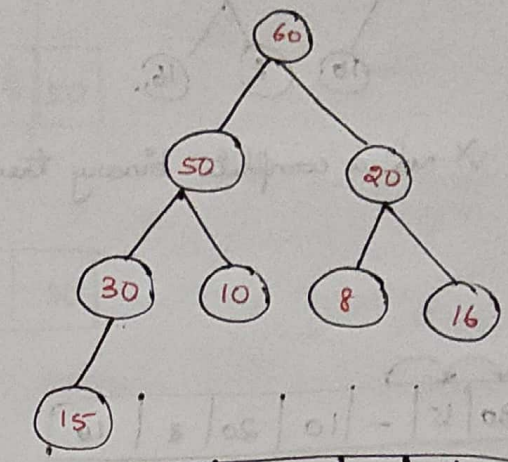
1	2	3	4	5	6	7	8
50	30	20	15	10	8	16	60

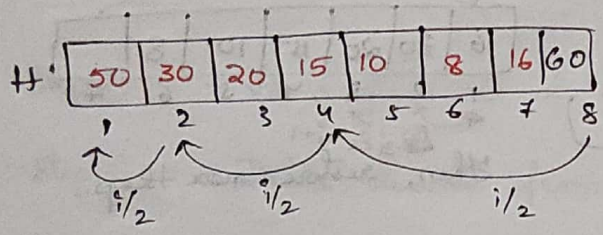
(15) — 4.

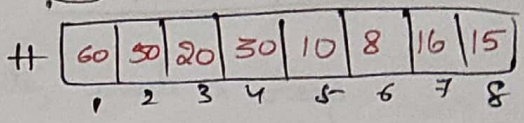
$2 * 4 = 8$.

(60) — 8.









Time taken = height of complete binary tree

= $O(\log n)$.

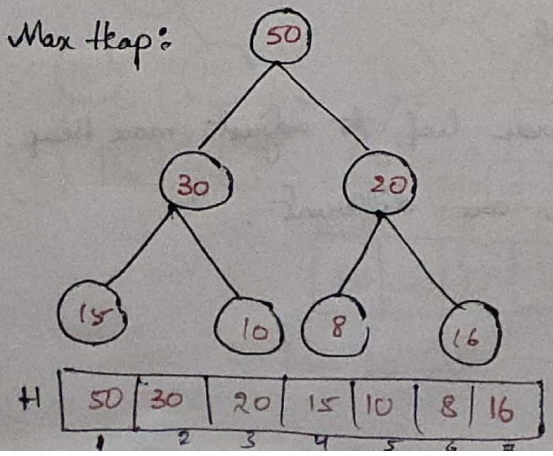
$\Rightarrow \log n$ number of swaps are required.

min $\Rightarrow O(1)$ to max $\Rightarrow O(\log n)$.

Delete

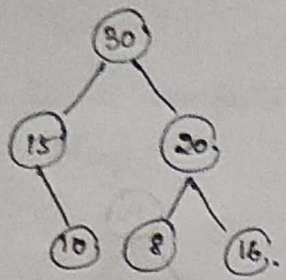
We should only delete root element.

Max Heap:

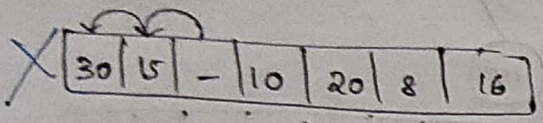


Remove 30 :-

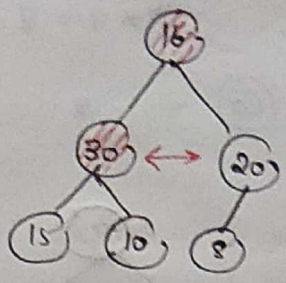
Wrong approach :-



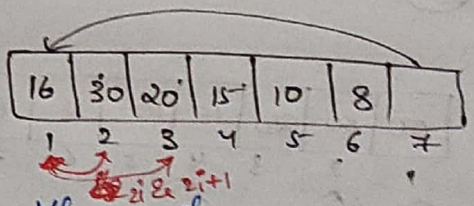
X not a complete binary tree.



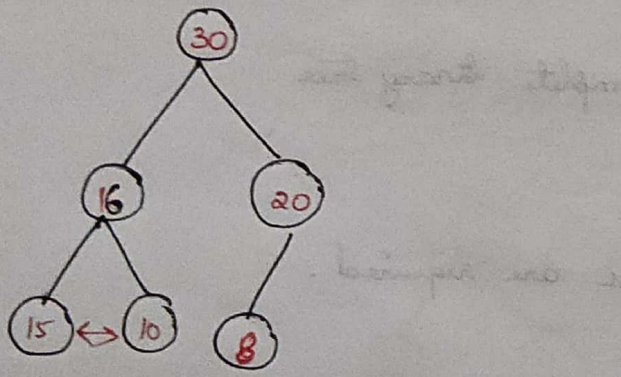
Right approach :-

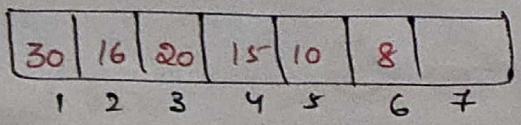


bring the last element of complete binary tree to root. Doing this, we will not lose complete binary tree property.



Then, restore max Heap.





① We always delete root element

② Last leaf node takes place of root

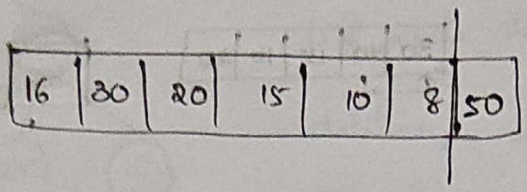
③ We push elements downwards towards leaf to adjust max Heap.

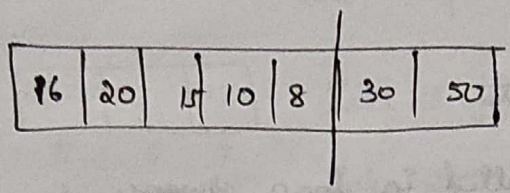
④ For insertion & deletion, directions are different.

max time. $\log n$

Heap Sort

Delete an element and fill the empty position of that array then the array gets sorted.





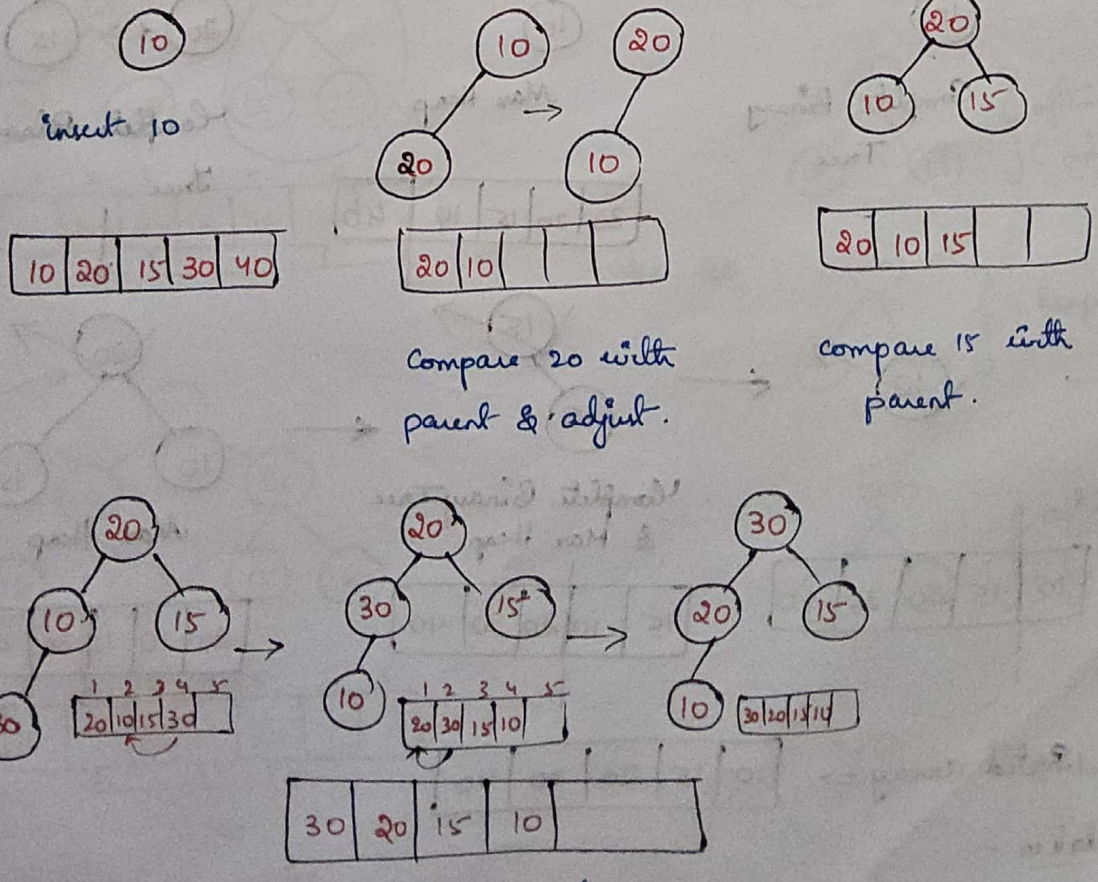
Two Steps:-

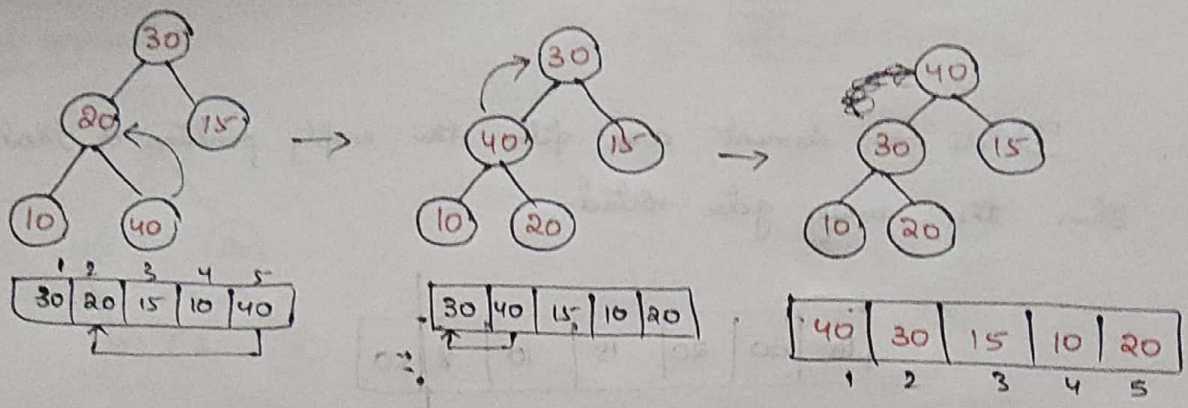
Step①:- For a given set of elements, create a Heap.

Step②:- Delete elements from Heap to sort them.

Step 1:-

Create Heap:-





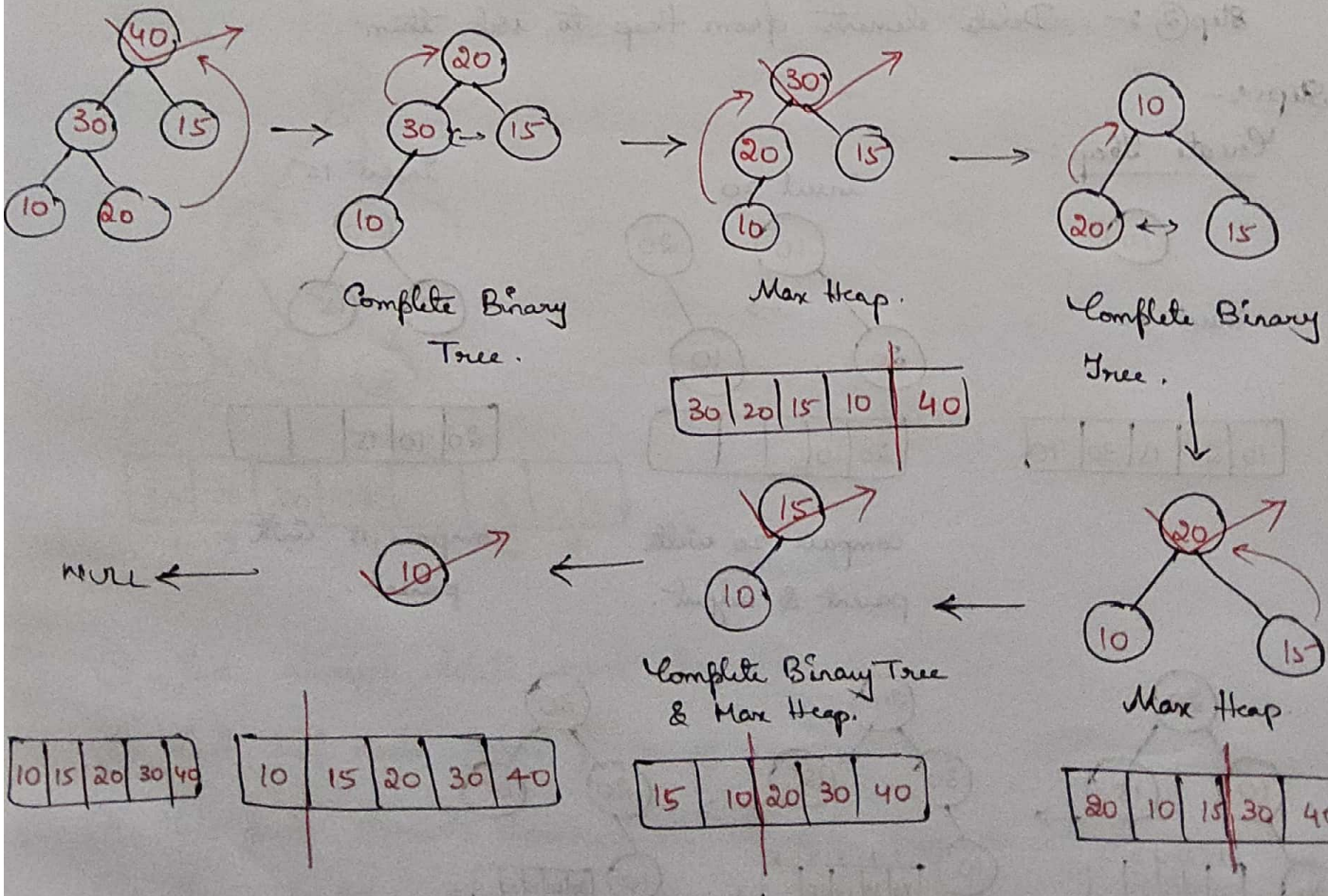
Time taken :-

Inserted 'n' elements.

Each insertion resulted in $\log n$ swaps

$O(n \log n)$ → to build max Heap.

Step 2 :-

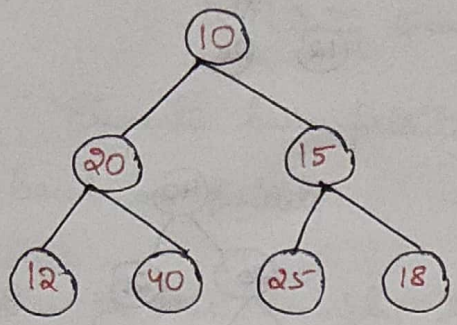


Sorted array => 10 | 15 | 20 | 30 | 40

Time taken :-

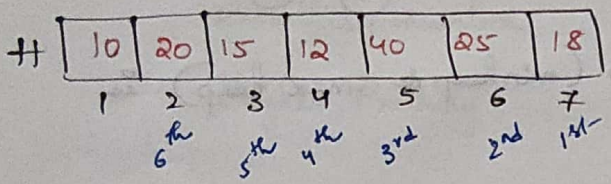
$O(n \log n)$ → to delete

Heapify



Similar to Deletion step where after deletion, we push elements towards leaf node to rebuild max heap after (or adjust) deletion, we consider from bottom the heap.

look downwards there are no children.



Take 18 first, it itself is a heap.

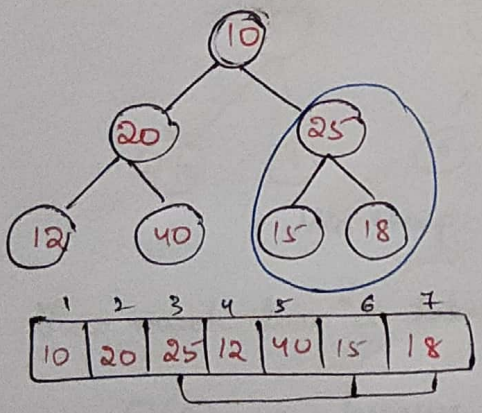
Take 25 " "

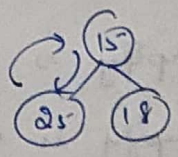
40 " "

12 " "

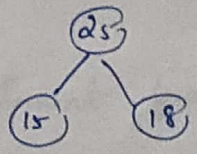
⋮

Now take 15, there are children.



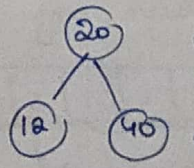


} a heap.

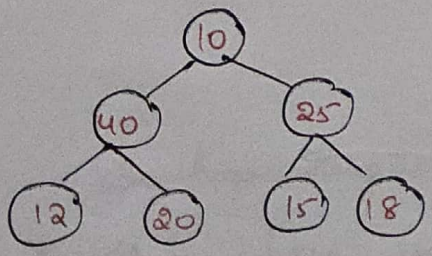


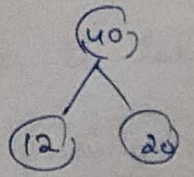
} after adjustment

take 20,



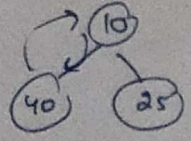
} a heap



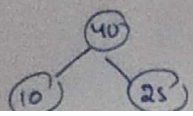


} after adjustment.

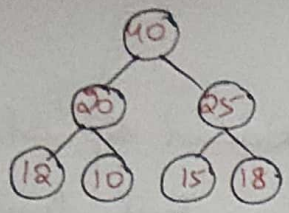
take 10,

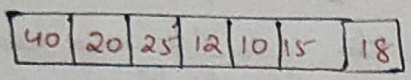


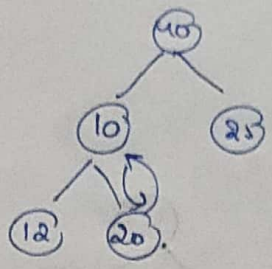
} a heap

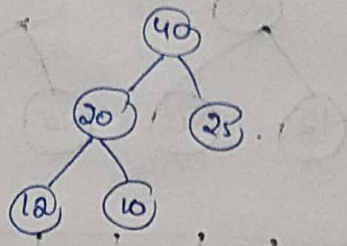


} after adjustment









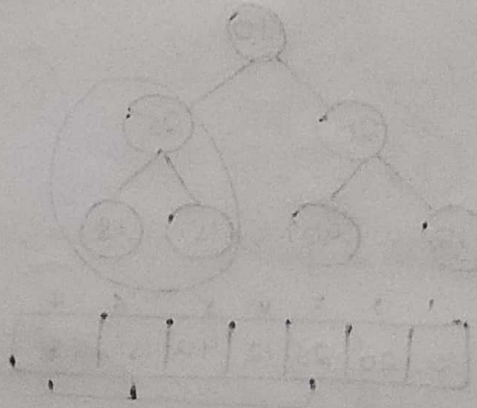
This procedure of building heap (min heap or max heap) is called Heapify.

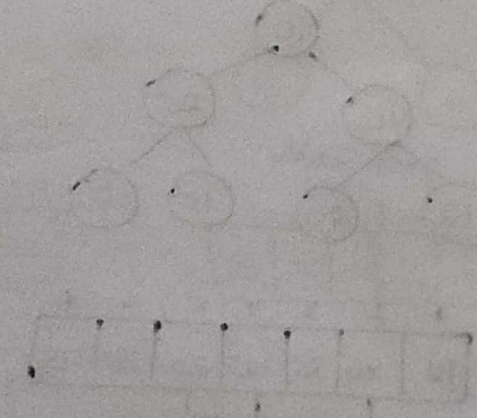
Time taken ⇒ $O(n)$.

Create Heap ⇒ $O(n \log n)$

Heapify ⇒ $O(n)$

So Heapify is faster.





Priority Queue

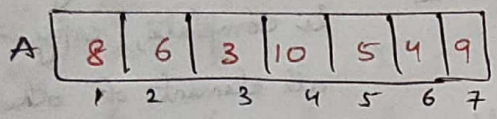
Queue – FIFO

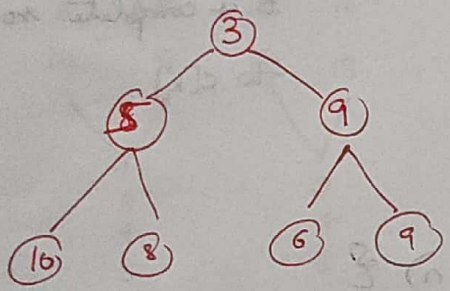
Priority Queue – not strictly FIFO.

Elements have priority and elements are inserted and deleted based on priority.

* In this example, the element value itself is priority.

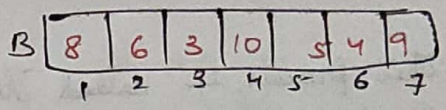
Smaller number
Higher Priority

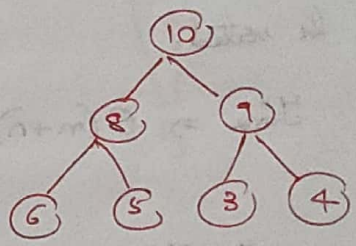




Min Heap.

Larger number
Higher Priority

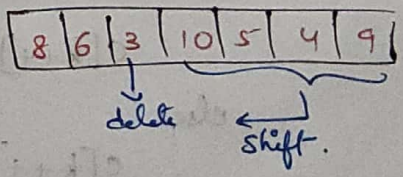




Max Heap.

How much time it takes to delete an element from an array?

O(n) → because if you remove 3, you need to shift all elements to left.



So here Heap is best data structure as it takes O(log n).

* If you have smaller number as high priority, use min heap.
* If you have larger number as high priority, use max heap.

Two Way Merge Sort - Iterative Method

Merge :-

A	B	C
i → 2	5 ← j	2 ← k
i → 8	9 ← j	5 ← k
i → 15	12 ← j	8 ← k
i → 18	17 ← j	9 ← k
i →	← j	12 ← k
		15 ← k
		17 ← k
		18 ← k

Compare A[i] & B[j] and place smaller element at C[k] and increment pointer of smaller element.

2 < 5 ⇒ C[k] = 2

i points to 8.

When one array traversal is complete, copy remaining all elements of other array.

B is complete so copy 18 to C[k].

This process is called "Merge".
If two lists are already sorted, result will be sorted.

Time ⇒ $\Theta(m+n)$.

```
Algorithm Merge (A, B, m, n) {
    i=1, j=1, k=1
    while(i<=m && j<=m) {
      if(A[i] < B[j])
            C[k++] = A[i++];
      else
            C[k++] = B[j++);
    }
    while(i<=m) {
        C[k++] = A[i++];
    }
    while(j<=n) {
        C[k++] = B[j++];
    }
}
```

Merge (more than two lists) :-

A	B	C	D
4	3	8	2
6	5	10	4
12	9	16	18

2 3 4 4 ....

At a time 4 lists are being merged ⇒ 4-way merging.

For M lists, M-way merging.

For 2 lists, 2-way merging

Ways of merging :-

*

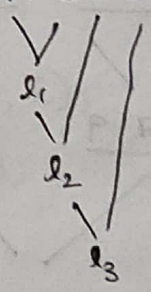
A	B	C	D
4	3	8	2
6	5	10	4
12	9	16	18

3 4 5 6 9 12

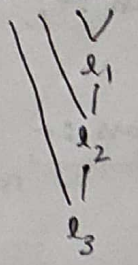
2 4 8 10 16 18

2 3 4 ...

*



*



There are 2 different merge sorts :-

① Two-way merge sort → Iterative

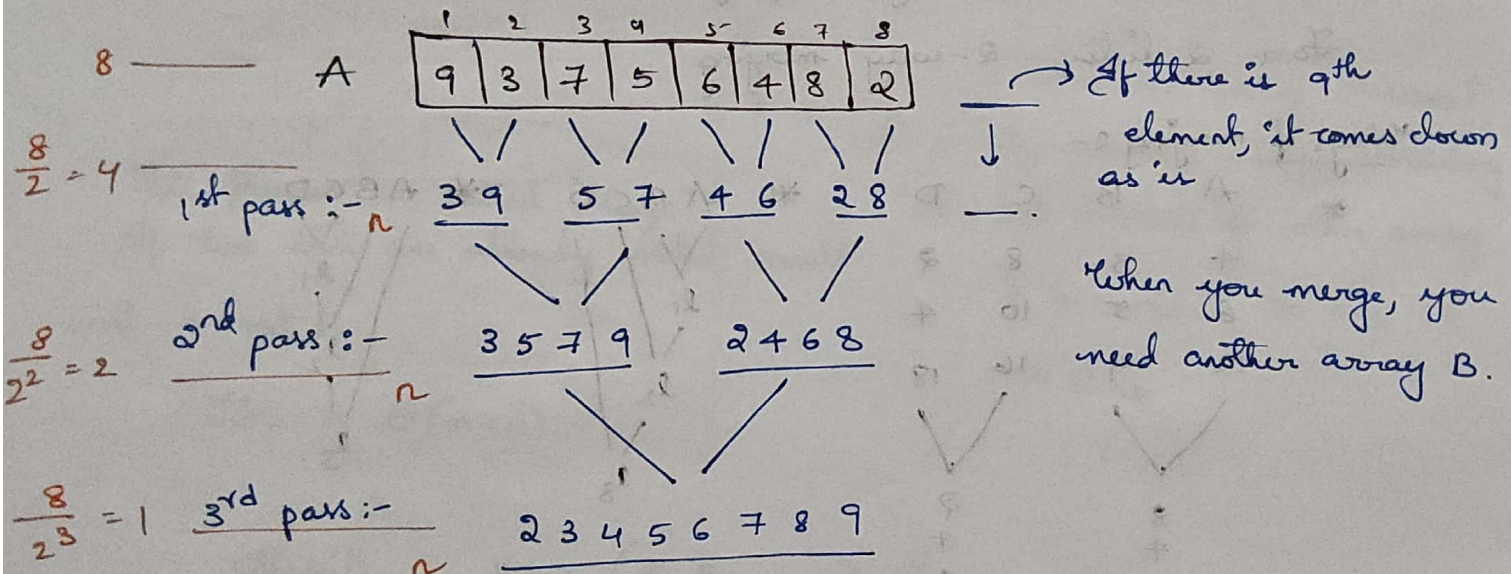
② Merge sort → Recursive.

2 way Merge Sort :-

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

To perform merge sort, we need two separate arrays but we don't even have them. Hence, we assume each element in the given array is a list and it has one element which means each list is sorted.

We will follow 2-way merge sort to merge them into a single list



$8 = 2^3$

$\log_2 8 = 3 \Rightarrow \log_2 n$ passes

no. of passes = $\log n$

Each pass has n operations (& elements)

$\Theta(n \log n)$

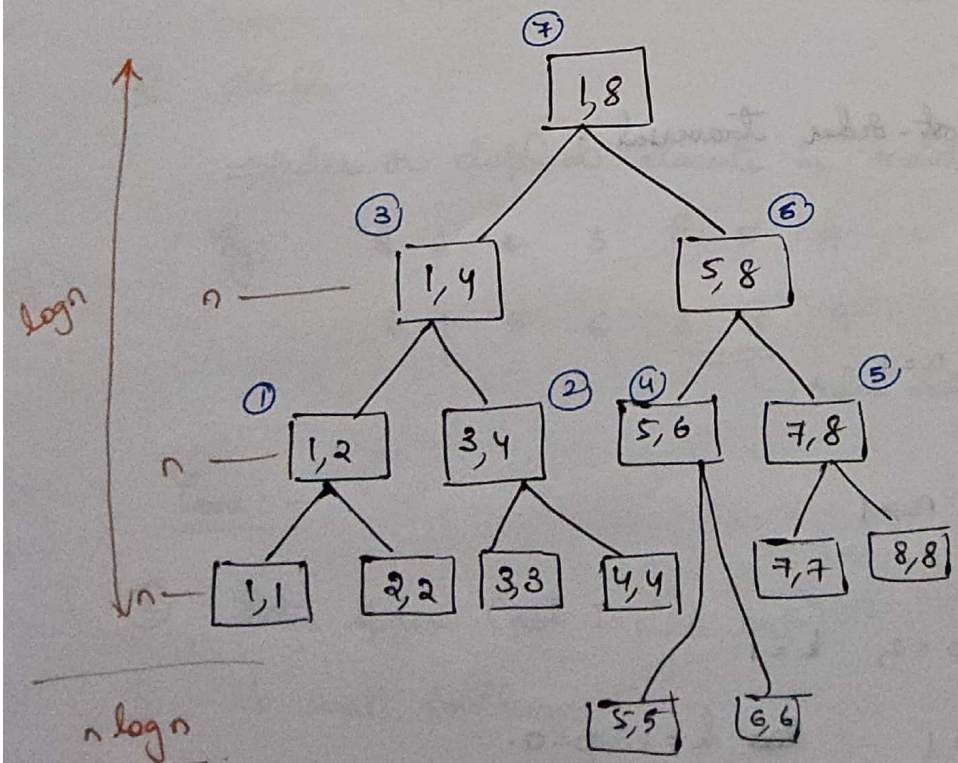
Merge Sort Algorithm

This is a recursive method and follows "Divide & Conquer" strategy.

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

Small problem = single element (which is already sorted).

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

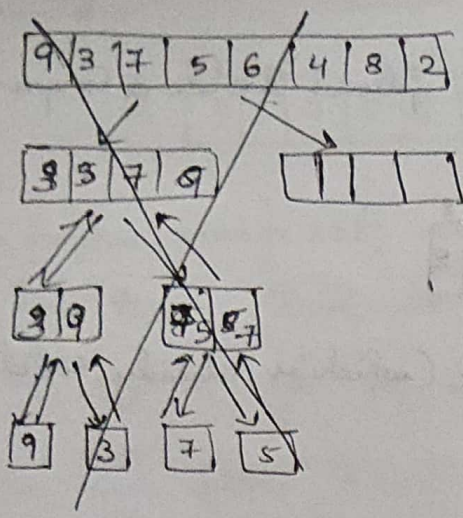


1	2	3	4	5	6	7	8
9	3	7	5	6	4	8	2

① 3,9 ② 5,7 ④ 4,6 ⑤ 2,8

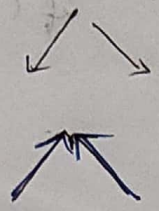
③ 3,5,7,9 ⑥ 2,4,6,8.

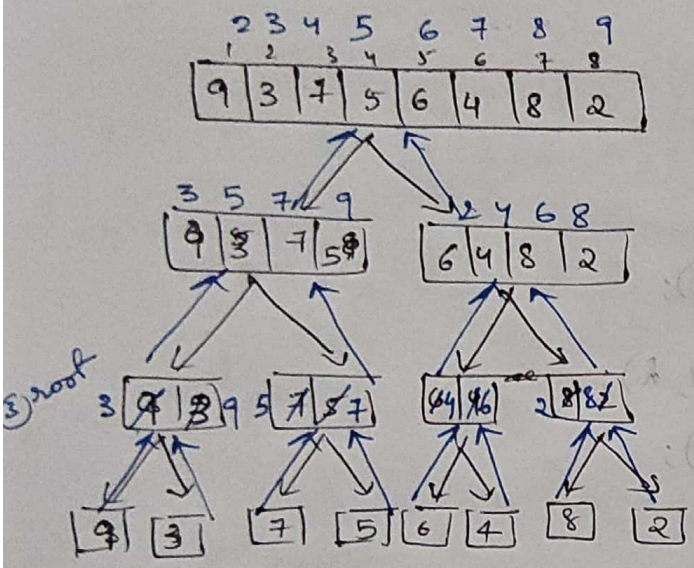
⑦ 2,3,4,5,6,7,8,9.



T(n) – Algorithm MS(l, h) {
if (l < h) {
1 —— mid = (l+h)/2;
$T(\frac{n}{2})$ —— MS(l, mid);
$T(\frac{n}{2})$ —— MS(mid+1, h);
n —— merge(l, mid, h);
}
}

$T(n) = 2T(\frac{n}{2}) + n$.





① left ② right

Merging is done in post-order traversal :

Time Complexity :-

Recurrence Relation $\Rightarrow T(n) = \begin{cases} 1 & n = 1 \\ 2T(\frac{n}{2}) + n & n > 1 \end{cases}$

Master's Theorem $\Rightarrow$ $a = 2,\ b = 2,\ k = 1$

$\log_b a = 1$ ~~log~~ $k = 1$. $p = 0$.

Case ② .. $p > -1 \Rightarrow$ multiply with $\log n$.

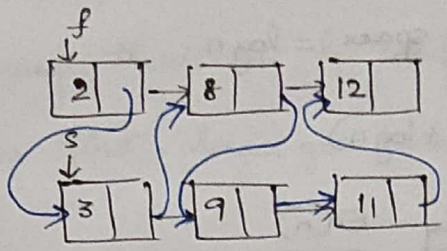
$\therefore \Theta(n \log n)$

MergeSort In-depth Analysis

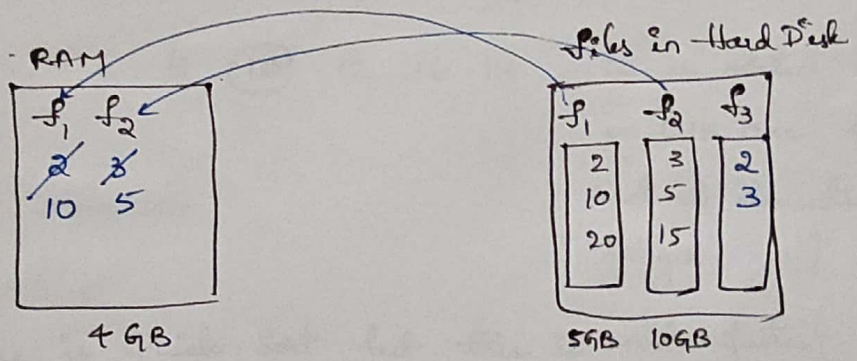
* Pros and Cons of Merge Sort :-

Pros :-

1. Large size list
2. Linked List



3. External Sorting.



4. Stable.

⇒ Order of duplicate elements is maintained.

Eg: 8 6 4 3 8 5 9

3 4 5 6 8 8 9

↳ order maintained (stable).

Cons :-

1. Extra space (not in place sort).
2. No small problem.

Insertion Sort — $O(n^2)$ ⇒ stable.

Merge Sort — $O(n \log n)$ ⇒ stable.

Merge Sort is slow for $n \leq 15$.

Bubble Sort — $O(n^2)$ ⇒ stable

For small size, you can use insertion sort & bubble sort.

But insertion sort is suitable for LinkedList like Merge Sort.

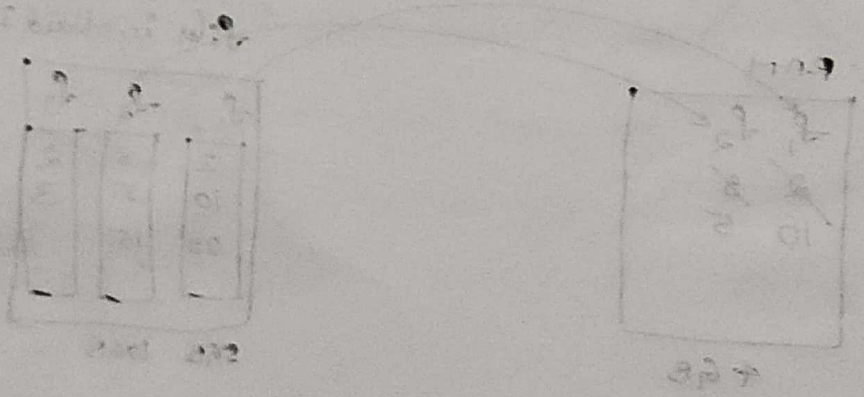
③ Recursive (needs extra stack space).

Extra space = n.

Stack space = log n.

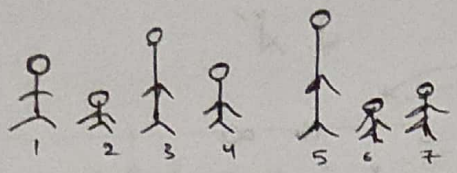
O(n + log n).

asymptotically $\underline{\Theta(n)}$



Quick Sort Algorithm

Idea:-



Students of different heights

If students are asked to arrange themselves according to heights,

Student 6 comes to front.
Student 5 comes to last.
Rest of students arrange themselves by comparing each other.
Student shorter comes first, larger goes back.

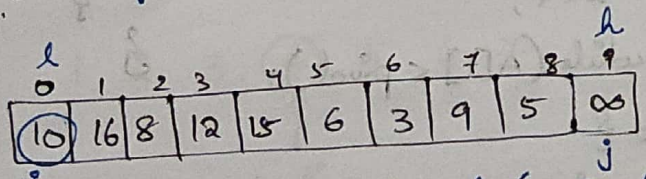
Eg:- (10) 80 90 60 30 20 — 10 is sorted as 10 is smallest

6 3 5 4 2 1 (9) — 9 is sorted as 9 is largest.

7 6 4 (10) 12 16 14 — 10 is sorted as all elements on LHS are smaller than 10 and all elements on RHS are larger.

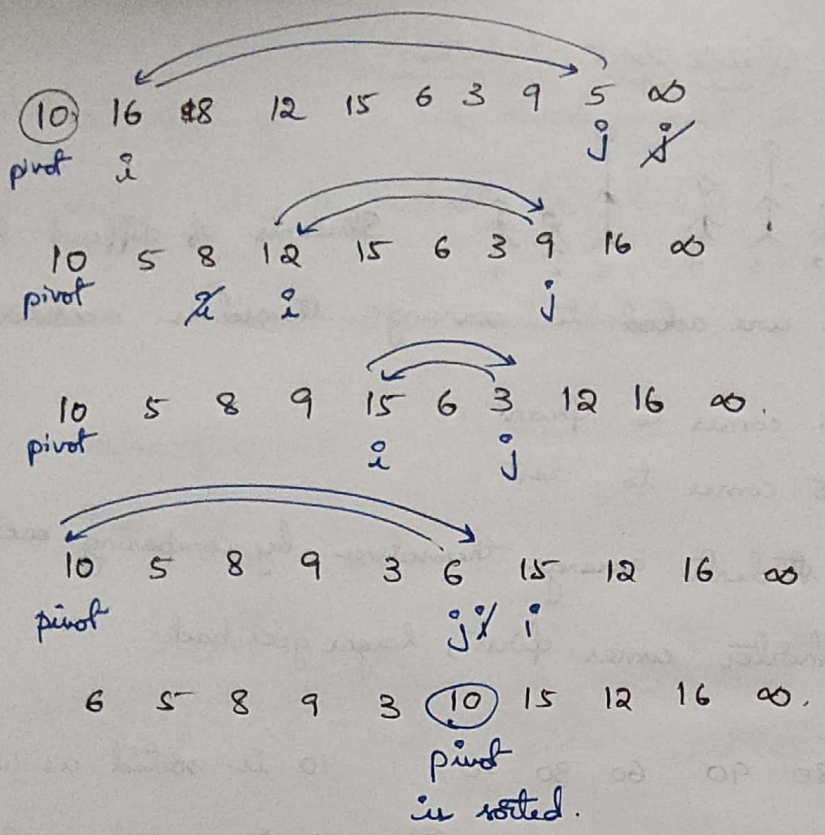
The name is Quick Sort but this is not fastest sorting technique. It is meant to quickly arrange elements (like in students example).

Quick Sort:- follows Divide & Conquer strategy.



pivot = 10. ⇒ bring smaller elements (<10) to the left [j is used for this]
larger elements (>10) to the right [i is used for this]

Increment i until element greater than pivot is found.
Decrement j until element less than pivot is found.
swap elements at i and j.
Continue until i ≤ j.

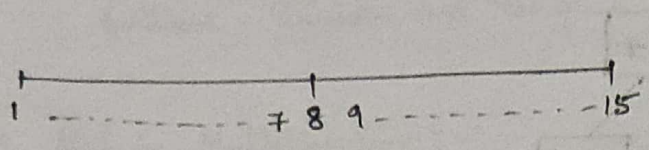


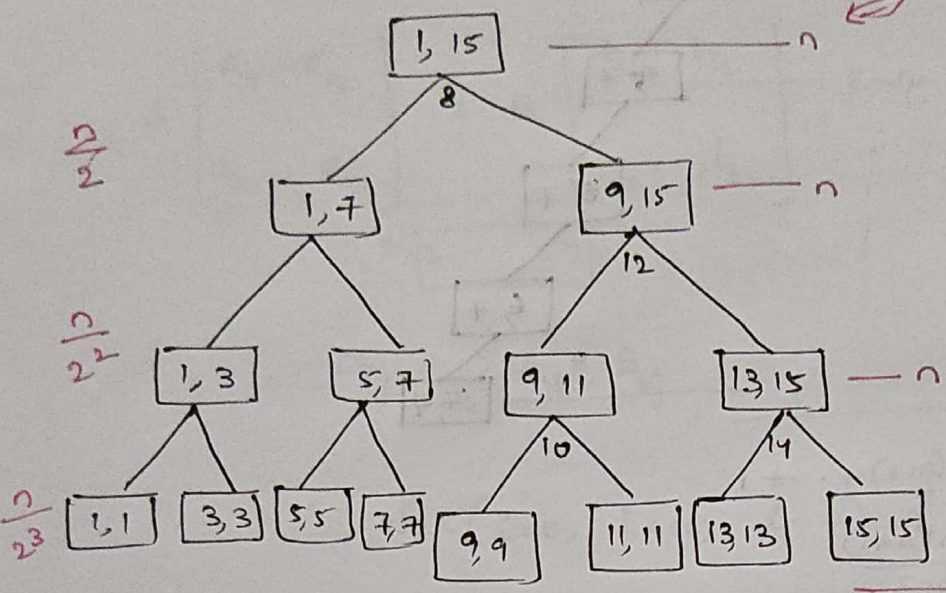
Perform Quick Sort recursively on left and right parts.

```
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);
    }
}
```

Quicksort Analysis






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

$O(n \log n)$

Best case :-

If partitioning is always in the middle $O(n \log n)$.

Above tree explanation is Best Case.

Median :- median is the middle element of a sorted list.

1 2 3 (4) 5 6 7

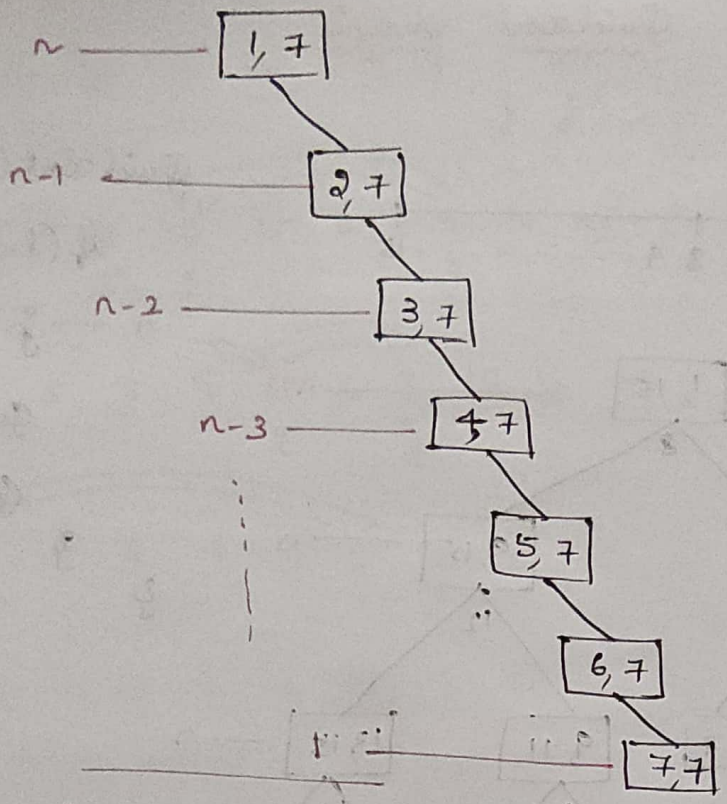
median

There is no guarantee that the pivot is always a median.

Worst case :-

(2)	4	8	10	16	18	17
pivot i, j	i	i	i	i	i	i
j						

Here partitioning always happens at the beginning of list since j traverses all towards the beginning as there is no element smaller than pivot



$n + (n-1) + (n-2) + \cdots + 1$

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

$O(n^2)$

If list is already sorted, pivot is always at beginning and the partition will be whole array except pivot whose size will always be less than 1 $(n, n-1, \ldots 1)$. The time complexity of Quick Sort for an already sorted array is $O(n^2)$ which is a big disadvantage.

Ways to remove worst case of Quick Sort :-

1. Select middle element as pivot (for few arrangements, worst case : $O(n^2)$)
2. Select random element as pivot. (for few arrangements, worst case : $O(n^2)$)

Always, worst case : $O(n^2)$.

Space complexity :- Size of Stack (best to worst) : $\log n$ to n.

Strassen's Matrix Multiplication

follows Divide and Conquer strategy.

① Matrix Multiplication :-

$$A\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times B\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = C\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$2 \times 2 \qquad 2 \times 2 \qquad 2 \times 2.$

$$C_{ij} = \sum_{k=1}^{n} A_{ik} * B_{kj}$$

```
for(i=0; i<n; i++) {
    for(j=0; j<n; j++) {
        C_ij = 0;
        for (k=0; k<n; k++) {
O(n^3)  ------------  C_ij += A_ik * B_kj
        }
    }
}
```

② Using Divide & Conquer Strategy :-

Small problem => multiplying 2×2 matrix.

$$A\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times B\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = C\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21}$$
$$c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$
$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21}$$
$$c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$$

For this smallest problem, we are not using two for loops. We directly solve by using above formulas. This is a constant time function.

8 multiplications is also constant.

If we consider 1×1 matrix multiplication as smallest problem:

$$A = [a_{11}] \qquad B = [b_{11}]$$

$$C = [a_{11} * b_{11}]$$

This is already used in 2×2

For problems larger than 2×2 :-

Divide and make matrices of 2×2.

If after division, dimension is smaller than 2×2, use zeroes to make dimension 2×2.

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] \quad \begin{matrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{matrix}$$

$$\frac{4}{2} \times \frac{4}{2}$$

$$B = \left[\begin{array}{cc|cc} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ \hline b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{array}\right] \quad \begin{matrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{matrix}$$

$$\frac{4}{2} \times \frac{4}{2}$$

```
Algorithm MM (A, B, n) {
    if (n ≤ 2) {
        c = 4 formulas
    } else {
        mid = n/2
```

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

$$MM(A_{11}, B_{11}, \tfrac{n}{2}) + MM(A_{12}, B_{21}, \tfrac{n}{2})$$
$$MM(A_{11}, B_{12}, \tfrac{n}{2}) + MM(A_{12}, B_{22}, \tfrac{n}{2})$$
$$MM(A_{21}, B_{11}, \tfrac{n}{2}) + MM(A_{22}, B_{21}, \tfrac{n}{2})$$
$$MM(A_{21}, B_{12}, \tfrac{n}{2}) + MM(A_{22}, B_{22}, \tfrac{n}{2}).$$

Proof :-

$$\boxed{C_{11}} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

(i)

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix} = \begin{bmatrix} \overbrace{\begin{matrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{matrix}}^{C_{11}} & C_{13} & C_{14} \\ C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \\ C_{41} & C_{42} & C_{43} & C_{44} \end{bmatrix}$$

$C_{11} \Rightarrow$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} \quad [A(\text{row}1)\ B(\text{col}1)]$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} \quad [A(\text{row}1)\ B(\text{col}2)]$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} \quad [A(\text{row}2)\ B(\text{col}1)].$$
$$c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} \quad [A(\text{row}2)\ B(\text{col}2)].$$

(ii)

$C_{11} \Rightarrow$ $\quad C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$$= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} + \begin{bmatrix} a_{13} & a_{14} \\ a_{23} & a_{24} \end{bmatrix} \begin{bmatrix} b_{31} & b_{32} \\ b_{41} & b_{42} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix} + \begin{bmatrix} a_{13}b_{31} + a_{14}b_{41} & a_{13}b_{32} + a_{14}b_{42} \\ a_{23}b_{31} + a_{24}b_{41} & a_{23}b_{32} + a_{24}b_{42} \end{bmatrix}$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} + a_{14}b_{41} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} + a_{14}b_{42} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} + a_{24}b_{42} \end{bmatrix}$$

↑ & ⇑ are equal.

Hence proved
☺

Time Complexity :-

Recurrence Relation:-

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 8T\left(\frac{n}{2}\right) + n^2 & n > 2 \end{cases}$$

n^2 → time taken to add elements. This is not scalar addition but matrix addition of size n^2 where n^2 elements are added.

Master Theorem :- $a = 8 \quad b = 2 \quad k = 2 \quad p = 0.$

$\log_b^a = 3. \qquad k = 2.$

Case ① $\Theta(n^{\log_b^a})$.

$= \Theta(n^3)$.

↳ Approach 1: 3 for loops ⇒ $\Theta(n^3)$

↳ Approach 2: Divide & Conquer ⇒ $\Theta(n^3)$ [uses extra space].

Approach 1 better than Approach 2.

Strassen's approach:-

→ Majority of ~~mul~~ operations are multiplications.

→ For 2×2 multiplication, we have 8 multiplications.

→ Strassen proposed 7 multiplications instead of 8.

$$P = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$$

$$Q = (A_{21} + A_{22}) \cdot B_{11}$$

$$R = A_{11} \cdot (B_{12} - B_{22})$$

$$S = A_{22} \cdot (B_{21} - B_{11})$$

$$T = (A_{11} + A_{12}) \cdot B_{22}$$

$$U = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$$

$$V = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$$

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$

$$C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$$

$$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$$

$$C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$$

There are 8 matrix multiplication (not just scalar multiplications).

$$C_{11} = P + S - T + V$$

$$C_{12} = R + T$$

$$C_{21} = Q + S$$

$$C_{22} = P + R - Q + U.$$

Recurrence Relation:-

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 7T\left(\frac{n}{2}\right) + n^2 & n > 2 \end{cases}$$

$\log_2 7 = 2.81 \qquad k = 2.$

$$O(n^{\log_2 7}) = O(n^{2.81}).$$

How to remember Strassen's formulas?

$P = [A_{11} + A_{22}][B_{11} + B_{22}]$

$Q = (A_{21} + A_{22})\ B_{11}$

$R = A_{11}\ (B_{12} - B_{22})$

$S = A_{22}\ (B_{21} - B_{11})$

$T = (A_{11} + A_{12})\ B_{22}$

$U = (A_{21} - A_{11})\ (B_{11} + B_{12})$

$V = (A_{12} - A_{22})\ (B_{21} + B_{22})$.

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$.

For $C_{11}, C_{12}, C_{21}, C_{22}$:-

① $C_{12} \Rightarrow RAT \Rightarrow R+T$

② $C_{21} \Rightarrow Q + S$ [use previous letters of R & T].

③ Use "P" on both C_{11} & C_{22}

④ $R+T$, $Q+S$ use $+S, +R$ on C_{11}, C_{22} and then $-T, -Q$ on C_{11}, C_{22}

⑤ Place $+V, +U$ in C_{11}, C_{22}.

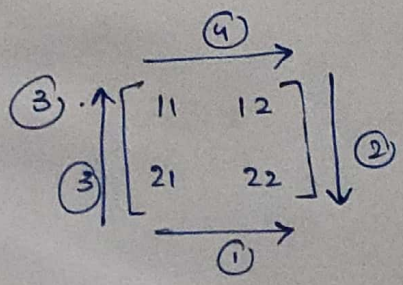
For P, Q, R, S, T, U, V :-

① P is easy $(A_{11} + A_{22})(B_{11} + B_{22})$

② $B_{11}\ A_{11}\ A_{22}\ B_{22}$ → put these in Q R S T

B as second term

A as first term.



Remember this arrow directions ① ② ③ ④

If there is B already, put $A_{21} + A_{22}$, $A_{11} + A_{12}$

A already, put $B_{12} - B_{22}$, $B_{21} - B_{11}$

④ For U and V, take S and R subtractions, replace B with A.

Then $(B_{11} + B_{12})$ $(B_{21} + B_{22})$