# Algorithm

## Chartercistics of Algorithm:

(i) Input (iii) Definiteness (v) Effectiveness

(ii) Output (iv) Finiteness

1. Time complexity

```
for (i=0; i<n; i++) {
    for (j=0; j<i; j++) {
        do...
    }
}
```

| i | j | no. of times |
|---|---|---|
| 0 | 0 × | 0 |
| 1 | 0 ✓ | 1 |
| | 1 × | |
| 2 | 0 ✓ | 2 |
| | 1 ✓ | |
| | 2 × | |
| 3 | 0 ✓ | 3 |
| | 1 ✓ | |
| | 2 ✓ | |
| | 3 × | |
| n | 0 ✓ | n |
| | 1 ✓ | |
| | ⋮ | |
| | n | |

T.C =

$\therefore 1+2+3+\cdots+n = \frac{n(n+1)}{2}$

$\therefore f(n) = \frac{n^2+n}{2}$

$\therefore O(n^2)$

---

2.

```
P = 0
for (i=1; P<=n; i++) {
    P = P+i
}
```

| i | P |
|---|---|
| 1 | 0+1 = 1 |
| 2 | 1+2 = 3 |
| 3 | 1+2+3 = 6 |
| 4 | 1+2+3+4 = 10 |
| ⋮ | ⋮ |
| k | 1+2+3+⋯k |

Assume, $P > n$

$\therefore P = \frac{k(k+1)}{2}$

$\therefore \frac{k^2(k+1)}{2} > n$

$\therefore k^2 > n$

$\therefore k > \sqrt{n}$

$\therefore$ T.C = $O(\sqrt{n})$

3. for (i=1; i<n; i = i*2) {
    do...
}

| i |
|---|
| 1 |
| 1×2 = 2 |
| 2×2 = $2^2$ |
| $2^2$×2 = $2^3$ |
| $2^3$×2 = $2^4$ |
| ⋮ |
| $2^k$ |

Assume, $i \geq n$

$\therefore i = 2^k$

$2^k \geq n$

let, $2^k = n$

$\therefore k = \log_2 n$

$\therefore O(\log_2 n)$

Extend for (i=1; i<n; i= i*2){
    do ---------
}

$\therefore \lceil \log n \rceil$

| n = 8 | n = 10 |
|---|---|
| i | i |
| 1 | 1 |
| 2 | 2 |
| 4 | 4 |
| 8× | 8 |
| 3 | 16× |
| | 4 |

$\log 8 = 3$

$\log_2 2^3 = 3$

$\log_2 10 = 3.2$

---

4. for ($i = n$; $i >= 1$; $i = \frac{i}{2}$){
    do... }

| i |
|---|
| n |
| n/2 |
| $n/2^2$ |
| $n/2^3$ |
| ⋮ |
| $n/2^k$ |

Assume, $i < 1$

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

$\therefore n = 2^k$

$\therefore k = \log_2 n$

$\therefore O(\log_2 n)$

5. for (i = 0; i*i < n; i++){

do - - -

}

Assume $i*i \geq n$

$\therefore i^2 \geq n$

let, $i^2 = n$

$\therefore i = \sqrt{n}$

$\therefore O(\sqrt{n})$

6. p = 0

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

p = p+1 ——— log n

}

for (j = 1; j < p; j = j*2) {

do - - -

} ⟶ log p

Here, $p = \log n$

$\therefore O(\log \log n)$

7. for (i = 0; i < n; i++) { ⟶ (n)

for (j = 1; j < n; j = j*2) { ~~n × log n~~

do - - -

} } ⟶ (log n)

∴ First loop = n times

Second ,, = log n ,,

$\therefore O(n \log n)$

So, for(i=0; i<n; i++) ... n [O(n)] → actually it will run for (n+1) times

for (i=0; i<n; i=i+2) ... $\frac{n}{2}$ [O(n)] — $\frac{n}{2}$ .. O(n)

for (i=n; i>1; i--) ... n [O(n)] — $\frac{n}{20}$ .. O(n)

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

for (i=1; i<n; i=i*3) ... $O(\log_3 n)$

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

## Analysis of if & while

1.
```
i=0 ---- 1
while (i<n) ---- n+1
{
    statement ---- n
    i++ ---- n
}
```

$f(n) = 3n+2$

$\therefore O(n)$

```
1          n+1          n
for(i=0; i<n; i++){
    statement — 1
}
```

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

---

2.
```
a=1
while(a<b){
    statement;
    a=a*2;
}
```

| a |
|---|
| 1 |
| $1\times2=2$ |
| $2\times2=2^2$ |
| $2^2\times2=2^3$ |
| : |
| $2^k$ |

(Assume it will run for k times)

Terminate; $a \geq b$

let, $a = 2^k$

$\therefore 2^k \geq b$

$2^k = b$

$\therefore k = \log_2 b$

$O(\log_2 b)$

3. `i = 1`
`k = 1`
`while (k < n) {`
`statement;`
`k = k + i;`
`i++;`
`}`

| i | k |
|---|---|
| 1 | 1 |
| 2 | 1+1 = 2 |
| 3 | 1+1+2 |
| 4 | 1+1+2+3 |
| 5 | 1+1+2+3+4 |
| ⋮ | ⋮ |
| m | 1+1+2+3+4 ... m |

roughly, $\frac{m(m+1)}{2}$

Assume, $k \geq n$

Let, $\frac{m(m+1)}{2} \geq n$

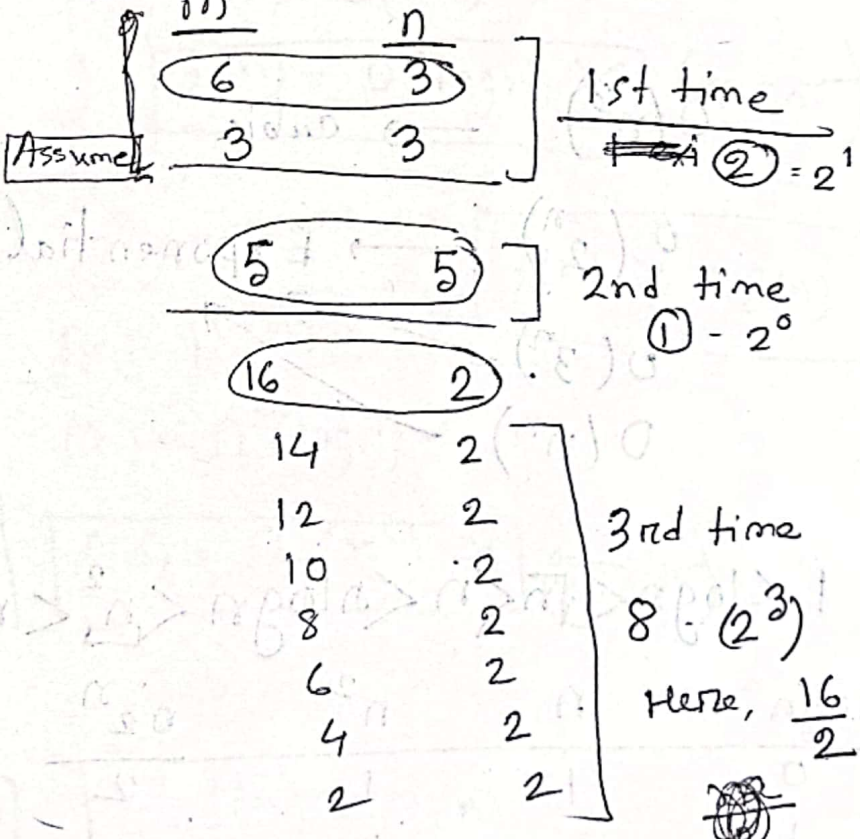
$\therefore m^2 \geq n$

$\therefore O(\sqrt{n})$

[using for,
`for (k=1; i=1; k<n; i++) {`
`Statement`
`k = k + i;`
]

4. `while (m != n)`
`{`
`if (m > n)`
`m = m - n;`
`else`
`n = n - m;`
`}`

$\therefore O(n)$




[minimum time = O(1)
maximum " = O(n)]

'gf' takey different time according to the condition.

Best case $O(1)$

worst " $O(n)$

## Types of time function

$O(1)$ ⟶ Constant ⟶ $f(n) = 2 \rightarrow O(1)$
" $= 1000$ ↗

$O(\log_{10} n)$ ⟶ logarithmic

$O(n)$ ⟶ Linear ⟶ $f(n) = 2n+3 \rightarrow O(n)$
" $= \frac{n}{100}$ ↗

$O(n^2)$ ⟶ Quadratic

$O(n^3)$ ⟶ cubic

$O(2^n)$ ⟶ Exponential

$O(3^n)$ ↗

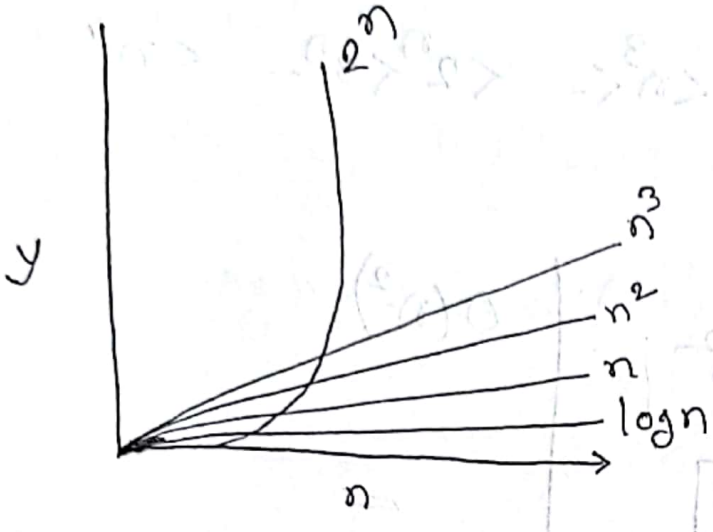
$O(n^n)$ ↗

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

| $\log n$ | $n$ | $n^2$ | $2^n$ |
|---|---|---|---|
| 0 | 1 | 1 | 2 |
| 1 | 2 | 4 | 4 |
| 2 | 4 | 16 | 16 |
| 3 | 8 | 64 | 256 |

Let, $n^{100} < 2^n$

At some point $2^n$ will be large and it will become larger later




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

Lower bound ($\Omega$) — $f(n)$ average bound ($\Theta$) — Upper bound ($O$)

**Big O**

Ex: $f(n) = 2n + 3$

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

$2n+3 \leq 5n, \quad n \geq 1$

$\therefore f(n) = O(n)$ ✓

$f(n) = O(n^2)$ ✓

$f(n) = O(2^n)$ ✓

$f(n) = O(\log n)$ ✗ (হবে না)

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

**Big Omega**

$f(n) = 2n+3$

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

$2n+3 \geq n \cdot 1$

$f(n)$ $g(n)$ $c$

$\therefore f(n) = \Omega(n)$ ✓

$f(n) = \Omega(\log n)$ ✓

$f(n) = \Omega(n^2)$ ✗ (হবে না)

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

**Big Theta**

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

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

$c_1$ $g(n)$ $f(n)$ $c_2$ $g(n)$

$\therefore f(n) = \Theta(n)$ ✓

$f(n) = \Theta(n^2)$ ✗

$f(n) = \Theta(\log n)$ ✗ (হবে না)

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

**Ex:1** $f(n) = 2n^2 + 3n + 4$

$2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2$

$2n^2 + 3n + 4 \leq 9n^2 \quad [n \geq 1]$

$\uparrow\uparrow$

$c\ g(n)$

$O(n^2)$

Again, $2n^2 + 3n + 4 \geq 1 \cdot n^2$ | $\Omega(n^2)$

Again,

$1 \cdot n^2 \leq 2n^2 + 3n + 4 \leq 9n^2$

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

$: \theta(n^2)$

**Ex-2:** $f(n) = n^2 \log n + n$

$n^2 \log n + n \leq 10n^2 \log n$

$1 \cdot n^2 \leq n^2 \log n + n \leq 10n^2 \log n$

$O(n^2 \log n)$

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

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

**Ex÷ 3** $f(n) = n!$

$= n \times (n-1) \times (n-2) \cdots \times 3 \times 2 \times 1$

$\therefore 1 \times 1 \times 1 \cdots \times 1 \leq 1 \times 2 \times 3 \cdots \times n \leq n \times n \times n \times n$

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

$O(n^n)$

$\Omega(1)$

[Here no $\theta$ is found]

Ex-4 $f(n) = \log n!$

$\log(1 \times 1 \times 1 \cdots \times 1) \leq \log(1 \times 2 \times 3 \cdots \times n) \leq \log(n \times n \times n \cdots \times n)$

$1 \leq \log n! \leq \log n^n \quad (n \log n)$

$O(n \log n)$
$\Omega(1)$
[no theta]

~~Ex-5~~

## Properties of Asymptotic notation

### ~~General properties~~

1. General properties: if $f(n)$ is $O(g(n))$ then

$af(n)$ is $O(g(n))$ [Here same for $\Theta(g(n))$ & $\Omega(g(n))$]

Ex: $f(n) = 2n^2 + 5$ is $O(n^2)$

then $7 \cdot f(n) = 7(2n^2 + 5)$

$= 14n^2 + 35$ is also $O(n^2)$

2. Reflexive: if $f(n)$ is given, then $f(n)$ is $O(f(n))$

Ex: $f(n) = n^2$ $\therefore O(n^2)$

(iii) Trasitive: if $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$

then $f(n) = O(h(n))$

Ex: $f(n) = n$ | $g(n) = n^2$ $h(n) = n^3$

$n$ is $O(n^2)$ . $n^2$ is $O(n^3)$

$n$ is $O(n^3)$

(it is true only for $\Theta$)

(v) symmetric: if $f(n)$ is $\Theta(g(n))$ then

$g(n)$ is $\Theta(f(n))$

Ex: $f(n) = n^2$ $g(n) = n^2$

$f(n) = \Theta(n^2)$ $g(n) = \Theta(n^2)$

(it is true for $O$ & $\Omega$)

(v) Transpose symmetric: if $f(n)$ is $O(g(n))$

then $g(n)$ is $\Omega(f(n))$

Ex: $f(n) = n$ $g(n) = n^2$

then $n$ is $O(n^2)$

$n^2$ is $\Omega(n)$

(VI) if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

$g(n) \leq f(n) \leq g(n)$

$\therefore f(n) = \Theta(g(n))$

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

$d(n) = O(e(n))$

then $f(n) + d(n) = O(\max(g(n), e(n))$

Ex

Assume:

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

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

$\therefore f(n) + d(n) = n + n^2$

$= O(n^2)$

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

$d(n) = O(e(n))$

then $f(n) * d(n) = O(g(n) * e(n))$

## Comparison of functions

1. $\log ab = \log a + \log b$
2. $\log \frac{a}{b} = \log a - \log b$
3. $\log a^b = b \log a$
4. $a^{\log_c b} = b^{\log_c a}$
5. $a^b = n$ then $b = \log_a n$

**Ex-1** $f(n) = n^2 \log n$ $\quad g(n) = n(\log n)^{10}$

Apply log,

$\log[n^2 \log n]$ $\quad \log[n(\log n)^{10}]$

$\log n^2 + \log \log n$ $\quad \log n + \log(\log n)^{10}$

$2\log n + \log\log n$ $\quad \log n + 10 \log(\log n)$

$\therefore f(n) > f(n)$

**Ex-2**: $f(n) = 3n^{\sqrt{n}}$ $\quad g(n) = 2^{\sqrt{n}\log n}$

$3n^{\sqrt{n}}$

$3n^{\sqrt{n}}$

$3n^{\sqrt{n}}$

$\log_a^{(2)} n^{\sqrt{n}} \to b$ [Rule 4]

$a \to 2$

$\left(n^{\sqrt{n}}\right)^{\log_2 2}$

$n^{\sqrt{n}}$

$\therefore f(n) > g(n)$

**Ex-3** $f(n) = n^{\log n}$ $\quad g(n) = 2^{\sqrt{n}}$

Apply log

~~Log [n log~~

$\log(n^{\log n})$ $\quad \log 2^{\sqrt{n}}$

$\log n \cdot \log n$ $\quad \sqrt{n} \log_2 2$

$(\log n)^2$ $\quad \sqrt{n}$

Apply log Again ~~2 log~~ $\log(\log n)^2$ $\quad \log \sqrt{n}$ ~~log n~~

$2 \log(\log n)$ $\quad \frac{1}{2} \log n$

$\therefore f(n) < g(n)$

**Ex-4** $f(n) = 2^{\log n}$ $\quad g(n) = n^{\sqrt{n}}$

Apply log,

$\log n \cdot \log_2 2$ $\quad \sqrt{n} \log n$

$\log n$ $\quad \sqrt{n} \log n$

$\therefore f(n) < g(n)$

**Ex-5** $f(n) = 2n \qquad g(n) = 3n$

$f(n) \neq g(n)$

But if we compare with Order, both are same and as we didn't applied log.

**Ex-6** $f(n) = 2^n \qquad g(n) = 2^{2n}$

$n \log_2 2 \qquad \Rightarrow 2n \log_2 2$

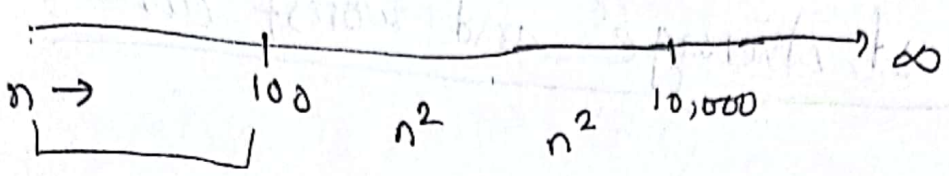
$n \qquad\qquad 2n$

$\therefore f(n) < g(n)$

Here we applied log, so we can't ignore decide that they are same. if log is applied there must be one bigger value if they are unequal.

**Ex-7** $g1(n) = \begin{bmatrix} n^3 & n < 100 \\ n^2 & n \geq 100 \end{bmatrix}$

$g2(n) = \begin{bmatrix} n^2 & n < 10,000 \\ n^3 & n \geq 10,000 \end{bmatrix}$




$g_1(n)$ greater

$g_1(n) > g_2(n)$

$g_1 = g_2$

$g_2(n) > g_1(n)$

Here $g_2(n)$ is always bigger after 10,000.

so, hence $g_2(n) > g_1(n)$

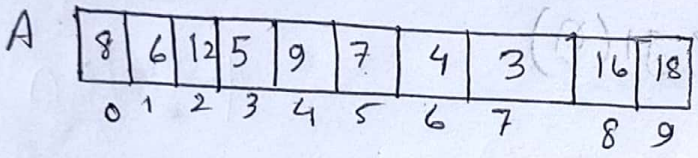
## True or False

1. $(n+k)^m = \Theta(n^m)$ ✓ $(n+3)^2 = \Theta(n^2)$ (3 ignorable)
2. $2^{n+1} = O(2^n)$ ✓ $2 \times 2^n$ (2 ignorable)
3. $2^{2n} = O(2^n)$ ✗ $4^n > 2^n$
4. $\sqrt{\log n} = O(\log \log n)$ ✗
5. $n^{\log n} = O(2^n)$ ✓ $\log n \times \log n$ $n$

# Best, Average and worst case

1. Linear search
2. Binary search Tree

## Linear search




$B(n) = O(1)$

$w(n) = O(n)$

$A(n) = O\left(\frac{n+1}{2}\right)$

$= O(n)$

Best case – searching key element present at first index

Best case time $\longrightarrow 1 \quad (O(1))$

$B(n) = O(1)$

worst case – searching key element present at last index

worst case time $= n \quad O(n)$

$w(n) = O(n)$

Average case $\longrightarrow$ $\frac{\text{all posible case time}}{\text{no. of cases}}$

Averag time $= \frac{1+2+3+\cdots+n}{n} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2}$

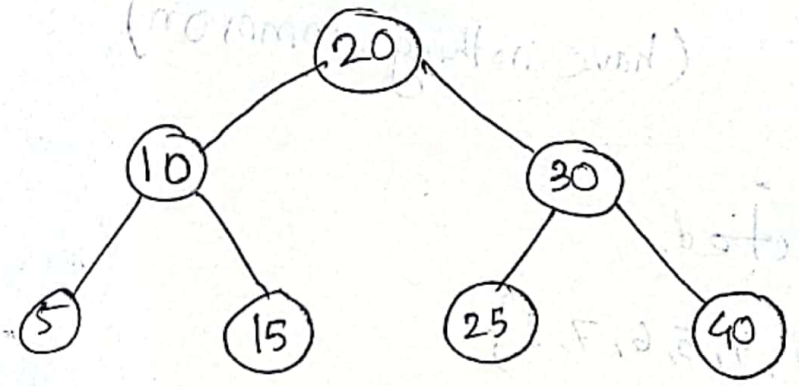
$B(n) = 1$ $\quad w(n) = n$

$B(n) = O(1)$ $\quad w(n) = O(n)$

$B(n) = \Omega(1)$ $\quad w(n) = \Omega(n)$

$B(n) = \Theta(1)$ $\quad w(n) = \Theta(n)$

## Binary search tree




height = log n

Best case → searching for root element

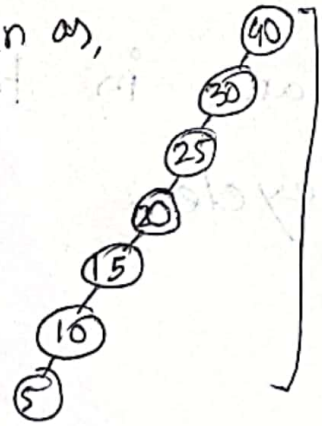
Best case time → $B(n) = 1$

worst case → searching for leaf element

worst case time → $W(n) =$ height of tree

So,
min $w(n) = \log n$
max, $w(n) = n$

This can be written as,

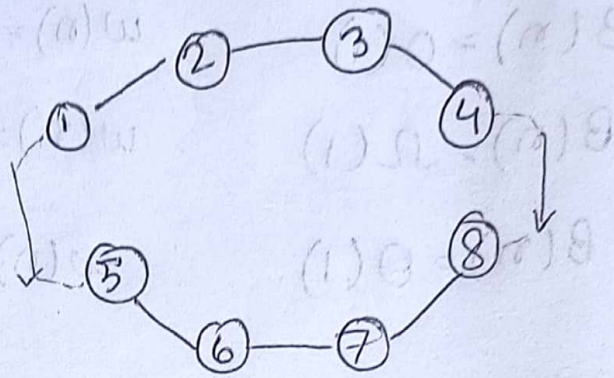



height = n

# Disjoint sets

1. Find
2. Union

] operation




$S_1 = \{1, 2, 3, 4\}$

$S_2 = \{5, 6, 7, 8\}$

$S_1 \cap S_2 = \phi$ . So, these are disjoint set (have nothing common)

$S_1$ $S_2$

↓ ↓

Assume (4, 8) connected.

$S_3 = S_1 \cup S_2 = \{1, 2, 3, 4, 5, 6, 7, 8\}$

Here $S_1$, $S_2$ are two different sets and its elements are connected. So, there is union operation between ~~the~~ $S_1$ & $S_2$

Here, (1, 5) are connected.

$S_3 = \{1, 2, 3, 4, 5, 6, 7, 8\}$

As ~~the~~ both of them are in the same set, so there is a cycle.

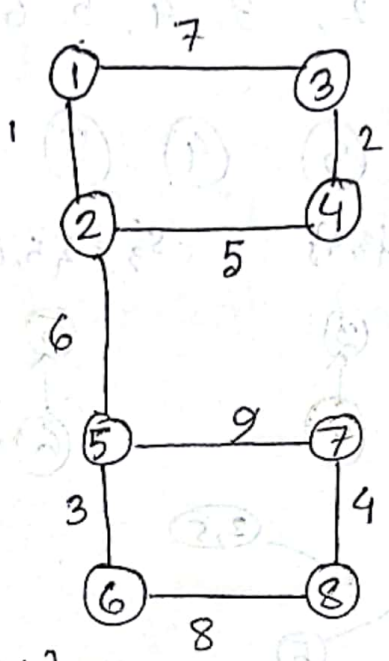
Ex] $u = \{1, 2, 3, 4, 5, 6, 7, 8\}$

① (1,2)

$S_1 = \{1, 2\}$

② (3,4)

$S_2 = \{3, 4\}$




③ (5,6)

$S_3 = \{5, 6\}$

④ (7,8)

$S_4 = \{7, 8\}$

⑤ (2,4)

$S_5 = S_1 \cup S_2$

$= \{1, 2, 3, 4\}$

⑥ (2,5)

$S_6 = S_5 \cup S_3$

$= \{1, 2, 3, 4, 5, 6\}$

⑦ (1,3)

Here (1,3) are both available in $S_6$. So if we take this, there will be a cycle in $S_6$

⑧ (6,8)

$S_7 = S_6 \cup S_4$

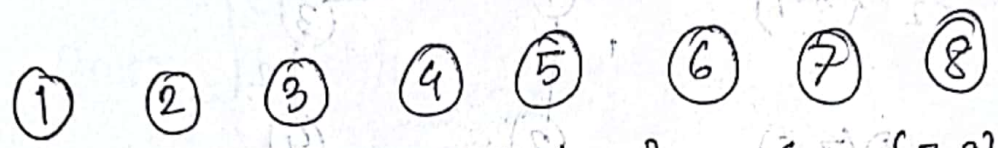
$= \{1, 2, 3, 4, 5, 6, 7, 8\}$

⑨ 5,7

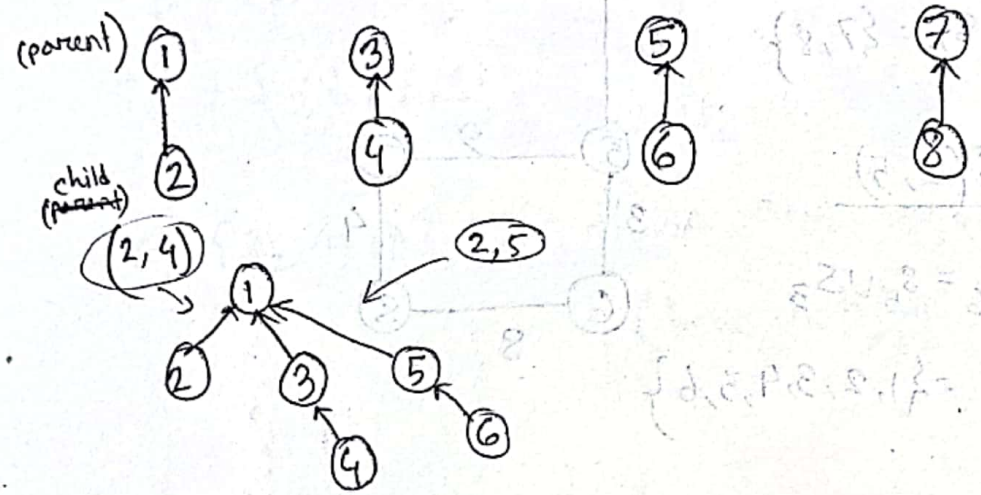
Here (5,7) are available in $S_7$. So, it is also not taken

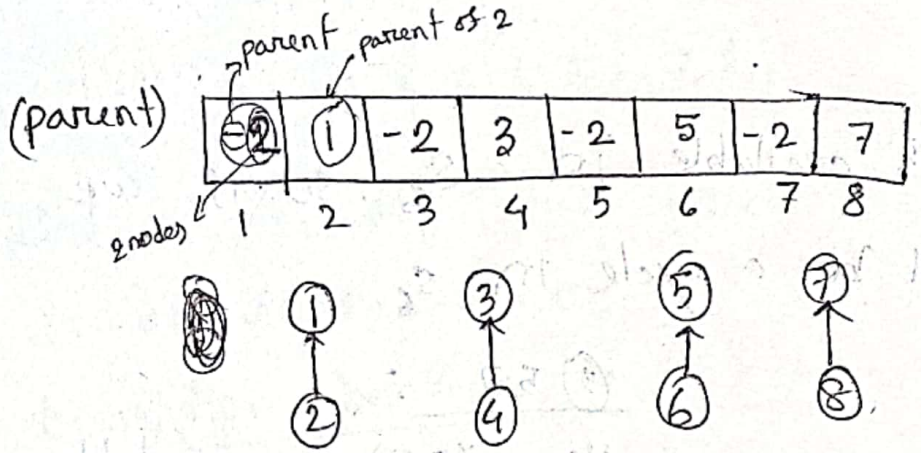
# How to represent previous example in array

$u = \{1, 2, 3, 4, 5, 6, 7, 8\}$

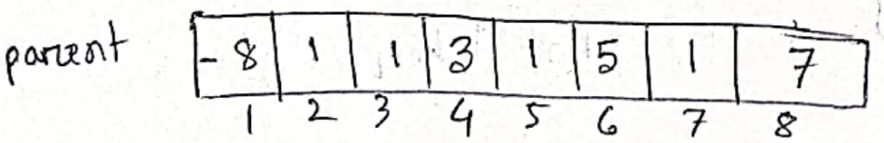


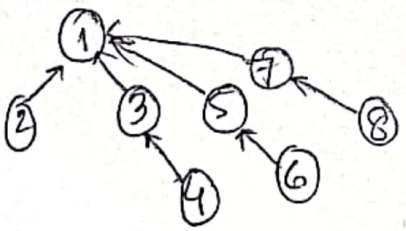

$S_1 = \{1, 2\}$ $S_2 = \{3, 4\}$ $S_3 = \{5, 6\}$ $S_4 = \{7, 8\}$



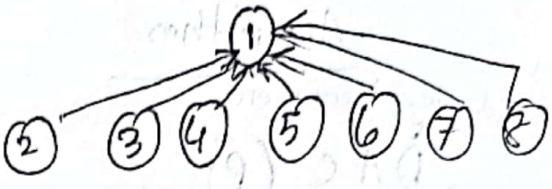



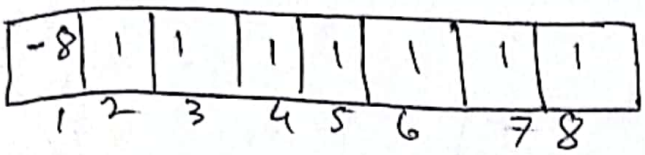


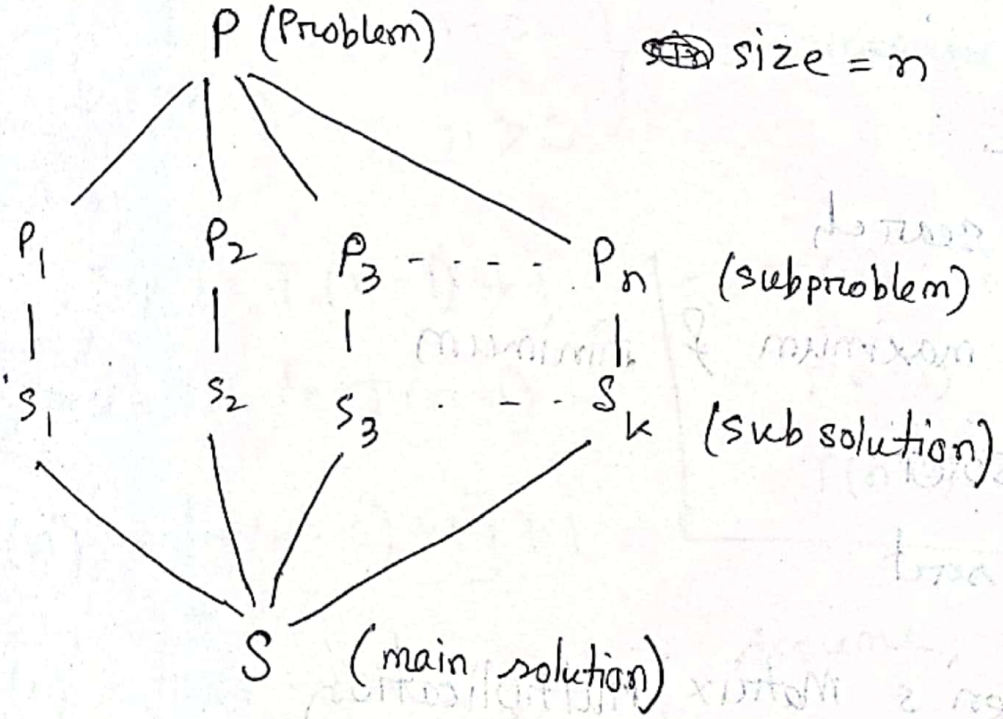
if we reprent this as




here, all nodes root is 1. it is called collpsing find. Here in constant time we can know parent of any child.

## Divide and Conquer




(Recurive solution)

## Algorithm

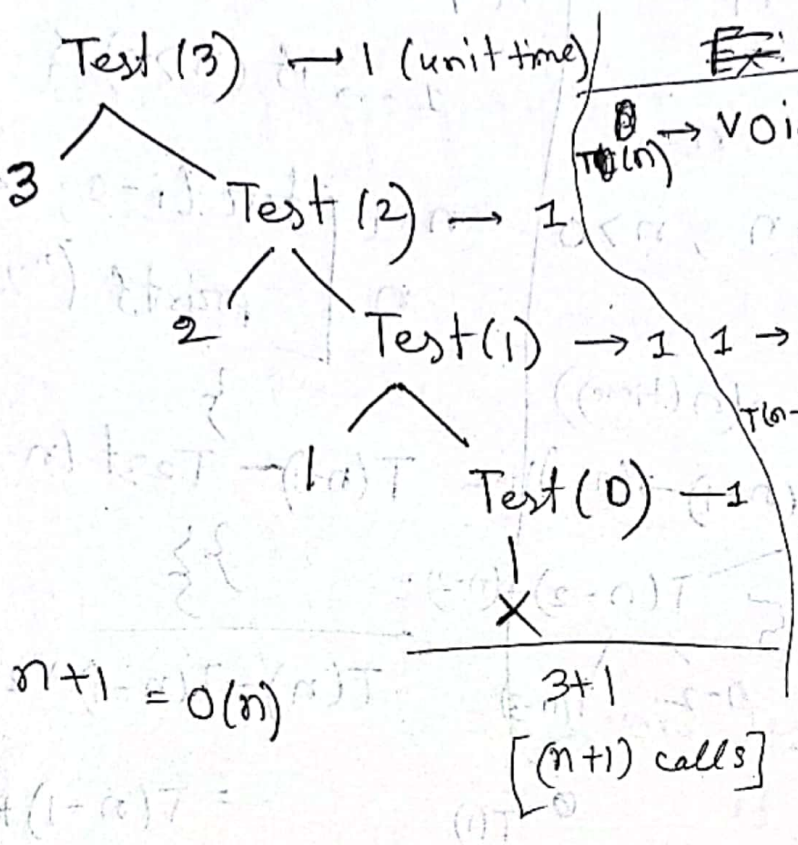
(divide and conquer)

```
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 & minimum
3. Merge sort
4. Quick sort
5. Strassen's Matrix Multiplication

# Recurrence Relation




3+1

[(n+1) calls]

$\therefore f(n) = n+1 = O(n)$

~~Ex~~

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

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

Ex

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

**solve:** $T(n) = T(n-1) + 1$

① Substitute $T(n-1)$

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

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

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

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

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

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

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

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

Assume $n - k = 0$

$\therefore n = k$

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

$= T(0) + n$

$\therefore \boxed{T(n) = 1 + n}$

$O(n) / \Theta(n)$

2.

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

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

$T(n) \rightarrow$ void Test (int n)

1 → if (n > 0) {

n+1 for (i = 0; i < n; i++) {

n printf ("%d", n);

}

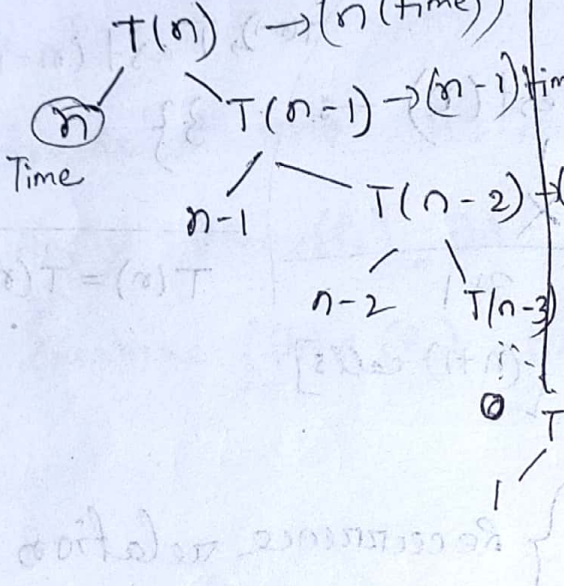
$T(n-1)$ Test (n-1);

}

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

$= T(n-1) + n$

solve:




$\therefore$ Total time $= 1 + 2 + \cdots + n = \frac{(n+1)\,n}{2}$

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

$O(n^2)$

**Substitution method:**

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

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

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

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

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

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

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

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

Assume, $n-k=0$

$\therefore n=k$

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

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

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

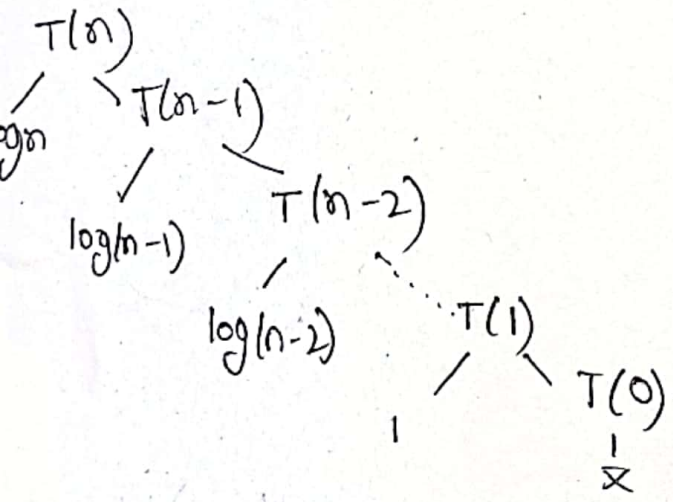
$\therefore O(n^2) \;/\; \Theta(n^2)$

3.

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

```
T(n) → void Test (int n)
       if (n>0) {
         for (i=1; i<n; i=i*2) {
           logn ——— printf ("%d", i) }
         T(n-1) ——— Test (n-1); }}
```

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




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

$= \log [n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1]$

$= \log n!$

$\therefore \boxed{O(n \log n)}$

Again, by substitution

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

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

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

~~$T(n-2) + \log$~~

$T(n) = T(n-3) + \log(n-2) + \log(n-1) + \log n$ --- (iii)

$T(n) = T(n-k) + \log 1 + \log 2 + \cdots + \log(n-1) + \log n$

$n - k = 0$

$n = k$

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

$= 1 + \log n!$

$\therefore O(n \log n)$

shortcut

① $T(n) = T(n-1) + 1 \quad \cdots \quad O(n)$

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

③ $T(n) = T(n-1) + \log n \quad \cdots \quad O(n \log n)$

④ $T(n) = T(n-1) + n^2 \quad \cdots \quad O(n^3)$

⑤ ~~$T(n) = T(n-2) + 1$~~

$\frac{n}{2} \quad O(n)$

⑥ ~~$T(n) =$~~

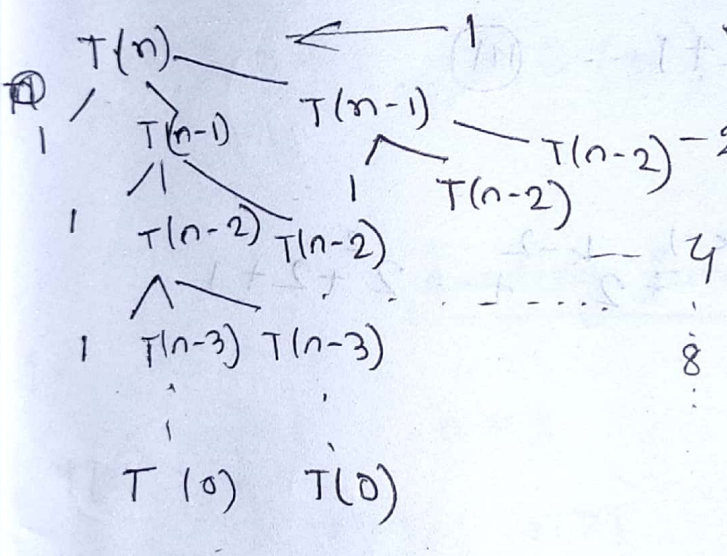
⑤ $T(n) = T(n-2)+1$ ... $\frac{n}{2}$ $O(n)$

⑥ $T(n) = T(n-100)+n$ ... $O(n^2)$

⑦ $T(n) = 2T(n-1)+1$ ... (1) $O(2^n)$

This type:

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




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

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

$\therefore 1+2^1+2^2+2^3+\cdots+2^k = 2^{k+1}-1$

$\left[a+ar+ar^2+ar^3+\cdots+ar^k = \frac{a(r^{k+1}-1)}{r-1}\right]$

Assume, $n-k=0$

$n=k$

$\therefore 2^{n+1}-1$

$\therefore O(2^n) / \Theta(2^k)$

Again,

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

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

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

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

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

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

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

Assume, $n - k = 0$

$$n = k$$

$\therefore$ ~~$2^n T(0) + 1 + 2 + \cdots 2^n$~~

$$2^n \cdot T(0) + 1 + 2 + 2^2 + \cdots + 2^{k-1}$$

$$= 2^n \times 1 + 2^k - 1$$

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

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

$\therefore O(2^n)$

# Master Theorem for decreasing function

$$T(n) = aT(n-b) + f(n)$$ → General form of recurrence relation

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

if $a = 1$, $O(n^{k+1})$ / $O(n * f(n))$

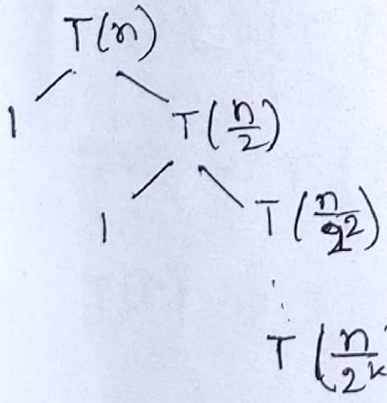
if $a > 1$, $O(n^k \cdot a^{n/b})$

if $a < 1$ $O(n^k)$ / $O(f(n))$

## dividing function

1.

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




T(n) → Algorithm Test (int n)

```
if (n>1) {
1 → printf("%d", n)
T(n/2) → Test (n/2);
}}
```

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

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

$\Rightarrow n = 2^k$

$\Rightarrow k = \log_2 n$

$\therefore O(\log n)$

By substitution,

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

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

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

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

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

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

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

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

$n = 2^k$

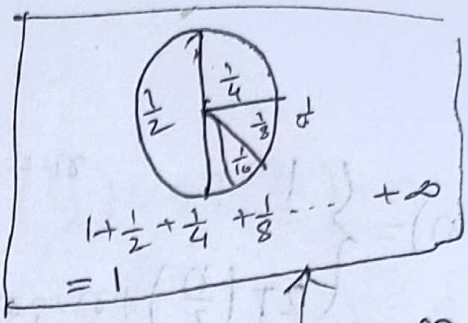
$\therefore k = \log_2 n$

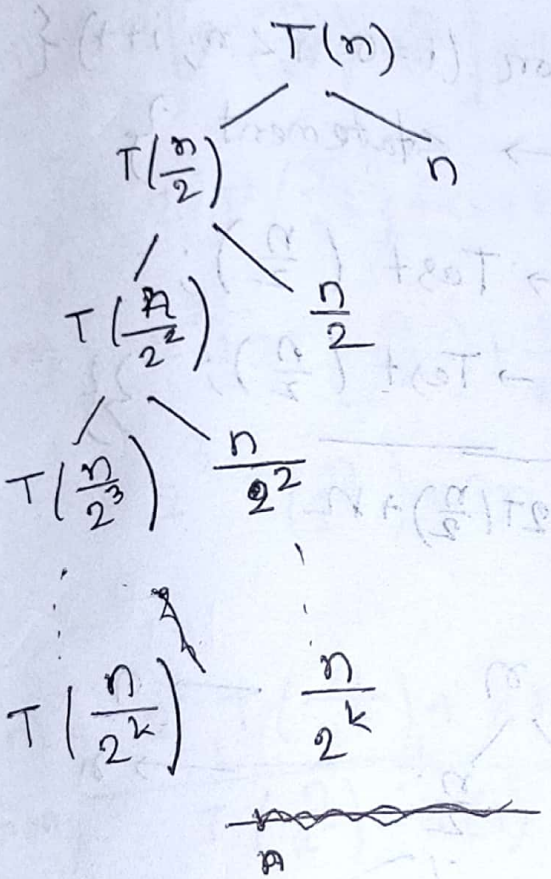
$\therefore T(n) = T(1) + \log n$

$\therefore = 1 + \log n$

$\therefore O(\log n)$

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




$$n + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^k}$$

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

$$= n \left[\sum_{i=0}^{k} \frac{1}{2^i} = 1\right]$$

$$= n \times 1$$

$$= n$$

$$\therefore O(n)$$

By substitution,

$$T(n) = T\left(\frac{n}{2}\right) + n \quad \cdots (1)$$

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

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

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

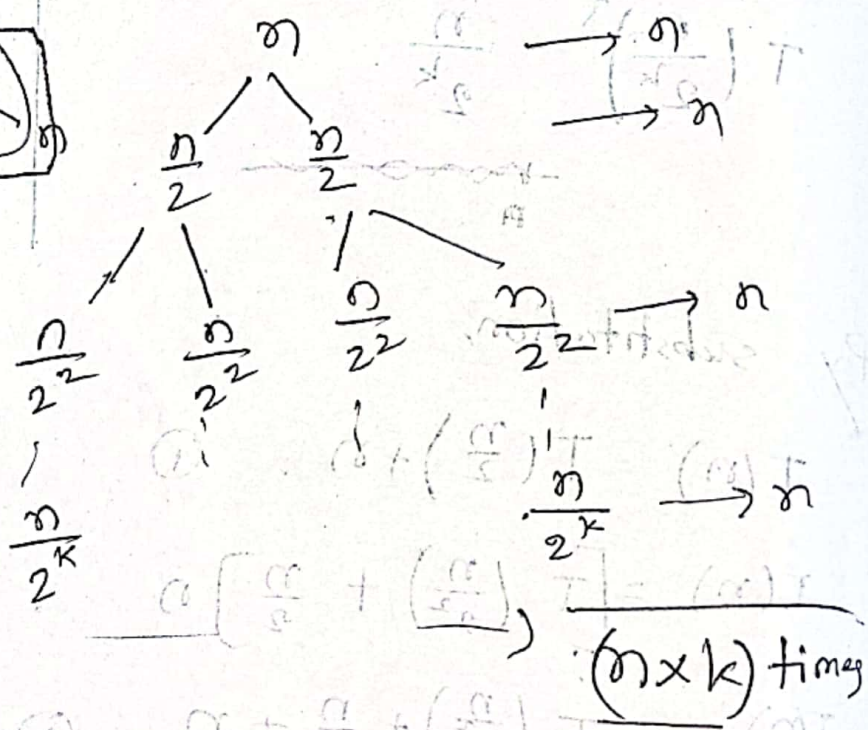
Same as before

$O(n)$

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

```
T(n) → Void Test (int n)
       if (n > 1) {
       for (i=0; i<n; i++) {
n ⟶ Statement }
T(n/2) → Test (n/2);
T(n/2) → Test (n/2); }}
```

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




k steps are there
n time taking

Assume,

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

$$k = \log n$$

$$\therefore O(n \log n)$$

By substitution

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

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

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

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

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

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

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

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

Assumption: $T\left(\frac{n}{2^k}\right) = T(1)$

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

$k = \log n$

$T(n) = 2^k \cdot T(1) + kn$

$= n \times 1 + n \log n$

$= n \log n$

$O(n \log n)$

## Master theorem for dividing fn

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

$a \geq 1$

$b > 1$

$f(n) = \Theta(n^k \log^p n)$

(I) $\log_b a$

(II) $k$

Case-I: if $\log_{ab} a > k$, then $\Theta(n^{\log_b a})$

Case-II: 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-III, if $\log_b a < k$

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

$p < 0$, $\Theta(n^k)$

Ex1 $T(n) = 2T\left(\frac{n}{2}\right) + 1$

$a = 2$

$b = 2$

$f(n) = \Theta(1)$

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

$k = 0, p = 0$

$\log_2 2 = 1 > k = 0$

case I

$\therefore O(n^1)$

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

$\log_2 4 = 2$

$k = 1, p = 0$

case I

$\therefore \Theta(n^2)$

Ex3 $T(n) = 8T\left(\frac{n}{2}\right) + n^1$

$\log_2 8 = 3 > k = 1$

case I

$\therefore O(n^3)$

(4) $T(n) = aT\left(\frac{n}{3}\right) + 1$

$\log_3 a = 2 > k = 0$

$\Theta(n^2)$

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

$\log_2 2 \quad k = 1, \; p = 0$

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

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

$\log_2 4 = 2, \quad k = 2$

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

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

$\log_2 8 = 3, \quad k = 3$

$\Theta(n^3 \log n)$

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

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

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

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

$\log_2 2 = 1, \quad k = 1, \; p = -2$

$\Theta(n)$

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

$\log_2 1 = 0 \quad < k = 2$

$\Theta(n^2)$

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

$\log_2 2 = 1 \quad < k = 2$

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

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

$\log_2 4 = 2, < k = 3$

$\Theta(n^3)$

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

$\log_2 4 = 2 \quad < \quad k = 3$

$\therefore \Theta(n^3)$

## Case I

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

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

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

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

## case III

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

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

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

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

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

## Case II

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

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

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

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

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

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

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

## Root function

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

```
void Test (int n) → T(n)
  if (n > 2) {
    statement → 1
    Test(√n) ; } } → T(√n)
```

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

Solve

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

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

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

$T(n) = T\left(n^{1/2^3}\right) + 3$ -- (III)

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

Assume, $n = 2^m$

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

Assume $T\left(2^{\frac{m}{2^k}}\right) = T(2^1)$

$\therefore \frac{m}{2^k} = 1 \; ; \; m = 2^k \; ; \; k = \log_2 m$

$\because n = 2^m, \; m = \log_2 n$

$k = \log\log_2 n$

$\Theta(\log\log_2 n)$

# Binary search (iterative)

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

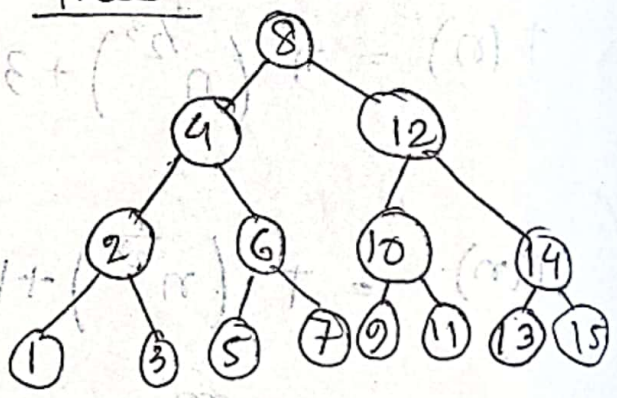
l, h, mid

key = 42

| l | h | mid = $\lfloor \frac{l+h}{2} \rfloor$ |
|---|---|---|
| 1 | 15 | $\frac{1+15}{2} = 8$ |
| 9 | 15 | $\frac{9+15}{2} = 12$ |
| 9 | 11 | $\frac{9+11}{2} = 10$ |
| 11 | 11 | $\frac{11+11}{2} = 11$ |

```
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;
}
```

Tree




O(log n)

minimum → O(1)

maximum → O(log n)

# Recursion

```
Algorithm RBinSearch(l, h, key)
{ if(l == h) {
    if (A[l] == key)
      return l;
    else
      return 0;
  }
  else {
    mid = (l+h)/2;
    if (key == A[mid]
      return mid;
    if (key < A[mid])
      return RBin-(l, mid-1, key)
    else
      return RBin--(mid+1, h, key)
  }
```

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

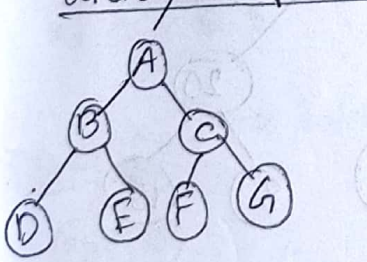
## Recurrence relation

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

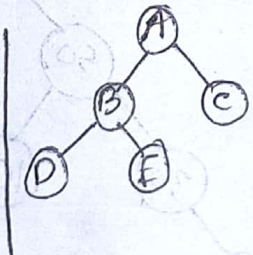
$\theta(\log n)$

# Heap

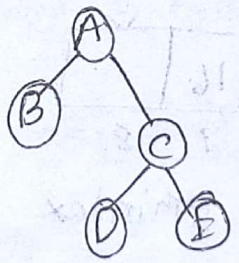
## array representation




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




| A | B | C | D | E |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |



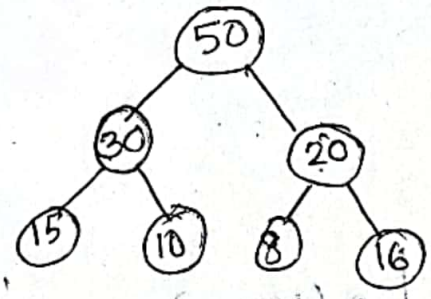

| A | B | C | - | - | D | E |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

if a node at index - i
its left child is at - 2*i
" right " " " - 2*i+1
" parent " " $\lfloor \frac{i}{2} \rfloor$

full binary tree → A binary tree with maximum no. of nodes

complete binary tree → There is no empty location in a tree's array representation (height = log n)

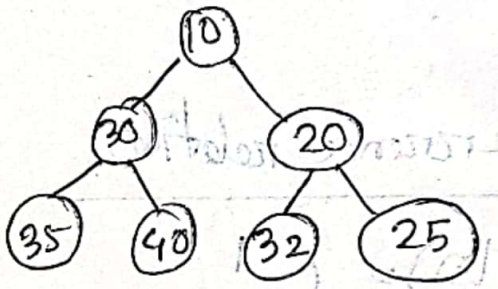
## max heap




H

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

## min heap




H

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

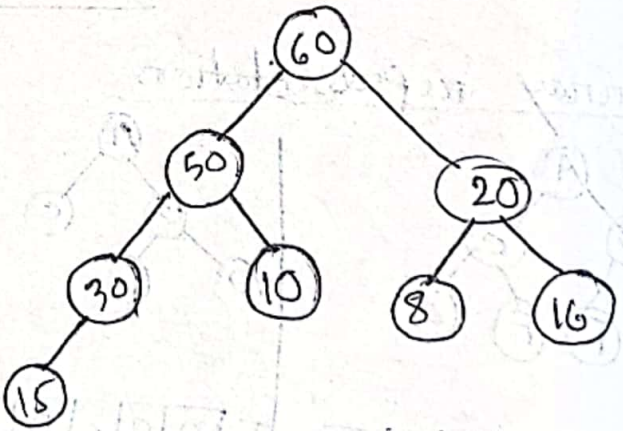
## Insert in max heap

Add 60 ← — — —

H

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

First insert it at 8th index




H

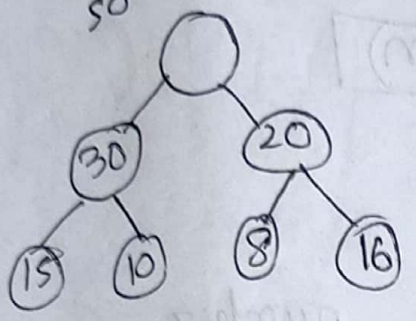
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 60 | 50 | 20 | 30 | 10 | 8 | 16 | 15 |

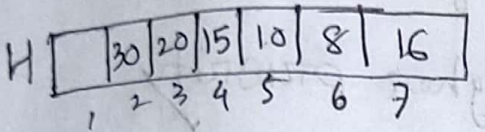
~~max~~ minimum → O(1)

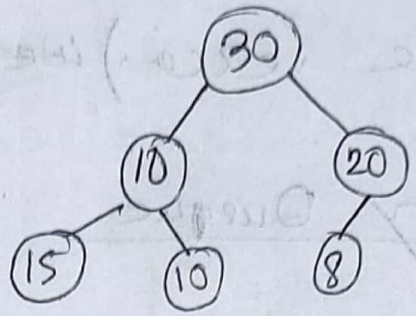
maximum → O(log n)

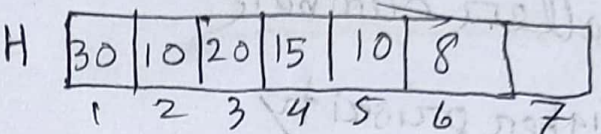
## Delete: (only root element can be deleted)

so




~~minimum time → O(1)~~

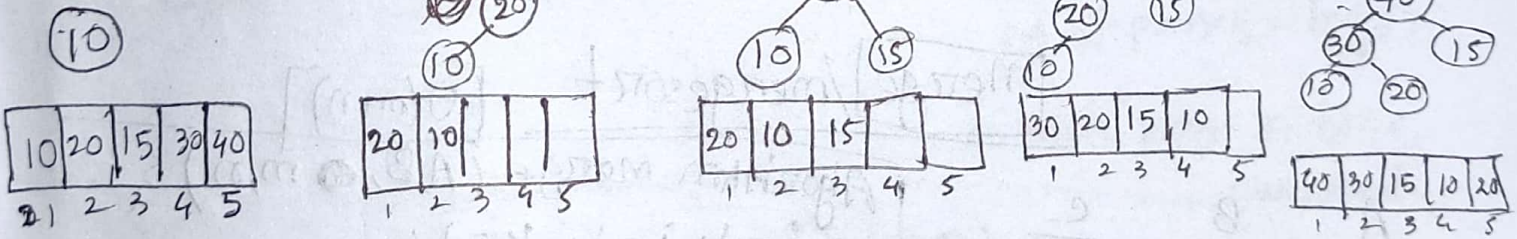
maximum time → O(log n)

## Heap sort

1. Create heap with given numbers
2. Delete from heap

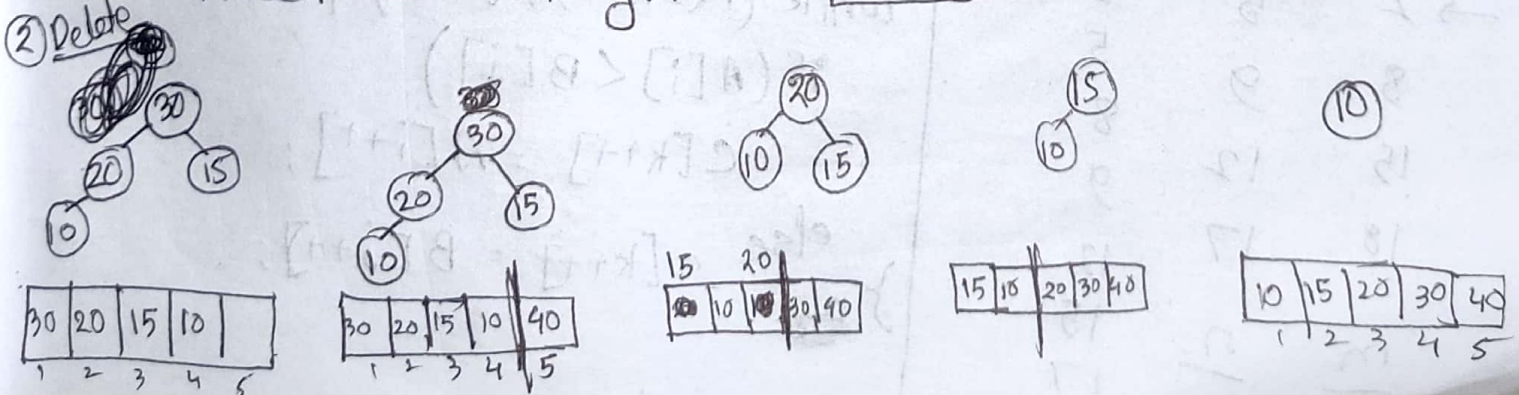
Ex: sort (10, 20, 15, 30, 40)

Sol: (1) Create heap




Time taken = n log n  O(n log n)

(2) Delete



**Heapify:** (it make min heap to max heap or vice versa.) ~~We take~~ $O(n)$

## Priority Queue

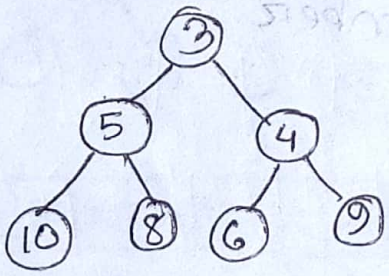
smaller number higher priority

| A | 8 | 6 | 3 | 10 | 5 | 4 | 9 |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

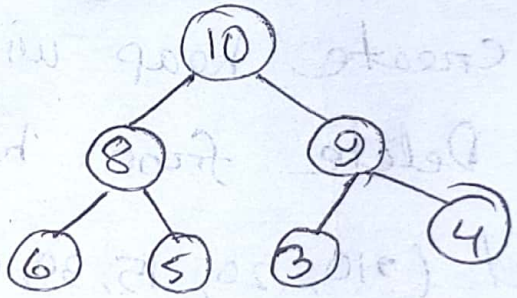
Large number higher priority

| B | 8 | 6 | 3 | 10 | 5 | 4 | 9 |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

min heap




max heap

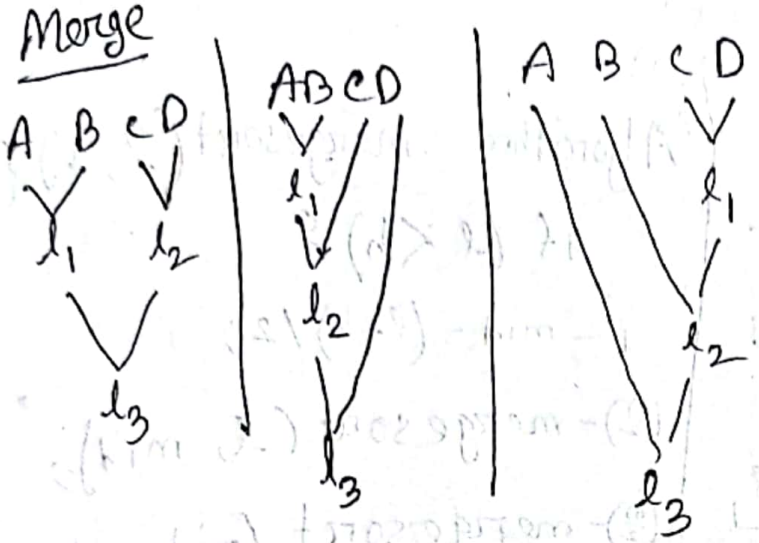



## Merge/mergesort [$\theta(m+n)$]

| A | B | C |
|---|---|---|
| i → 2 | 5 | 2 |
| 8 | 9 | 5 |
| 15 | 12 | 8 |
| 18 | 17 | 9 |
| m | n | 12 |
| | | 15 |
| | | 17 |

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

## Merge




```
for( ; i<=m; j++)
    c[k++] = A[j]
for (; j<=n; j++)
    c[k++] = B[j]
```

## 2way merge sort

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

1st pass: 3 9 | 5 7 | 4 6 | 2 8 (keep it in a separate array)

2nd pass: 3 5 7 9 | 2 4 6 8

3rd pass: 2 3 4 5 6 7 8 9

$\rightarrow n \quad \frac{8}{2} \quad \frac{\frac{2}{2}}{2} = 1$

$\rightarrow n \quad \frac{8}{2^3} = 1$

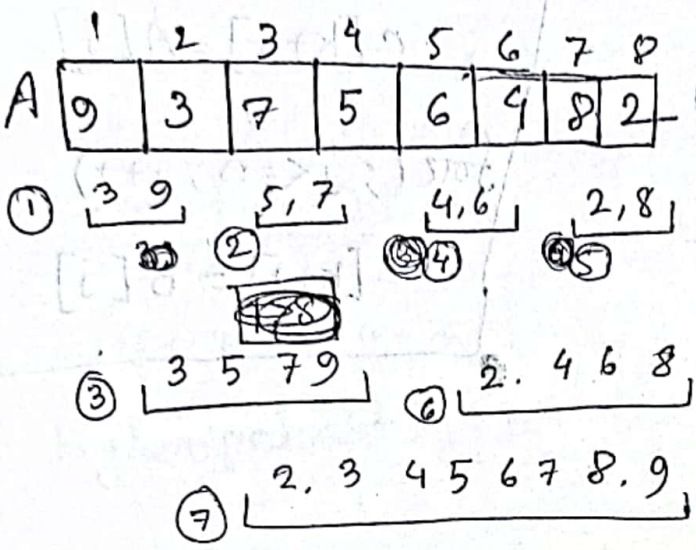
$\rightarrow n \quad \log_2 8 = 3$

no. of passes $= \log n$

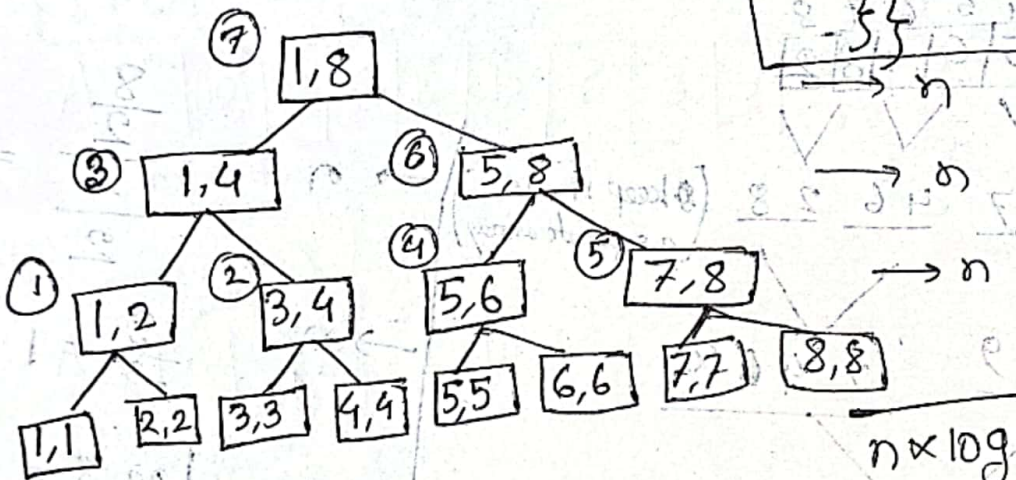
in each pass $n$ time required

$\therefore$ **Time taken $= O(n \log n) / \Theta(n \log n)$**

# merge sort

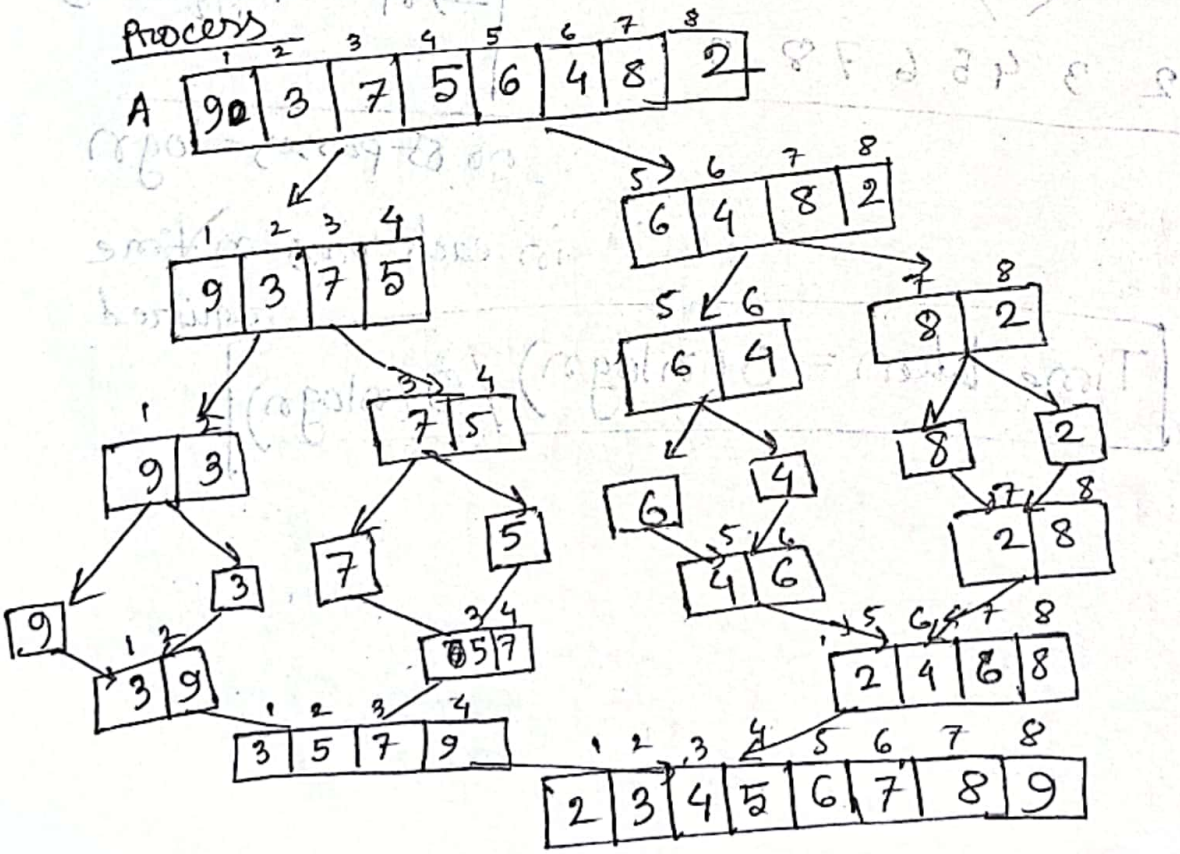



```
T(n)
Algorithm mergesort(l, h){
    if (l < h){
        mid = (l+h)/2;
T(n/2) - mergesort(l, mid);
T(n/2) - mergesort(mid+1, h);
n - merge(l, mid, h)
}}
```




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

Process



## Recurrence relation

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

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

here, $a = 2, b = 2, f(n) = n$

$n^k = n^1$

$\log_2 2 = k$

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

## Pros and cons of merge sort

**Pros:**

1. Large size List can be sorted.
2. Linked List (suitable for)
3. supports external sorting
4. stable

**Cons**

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

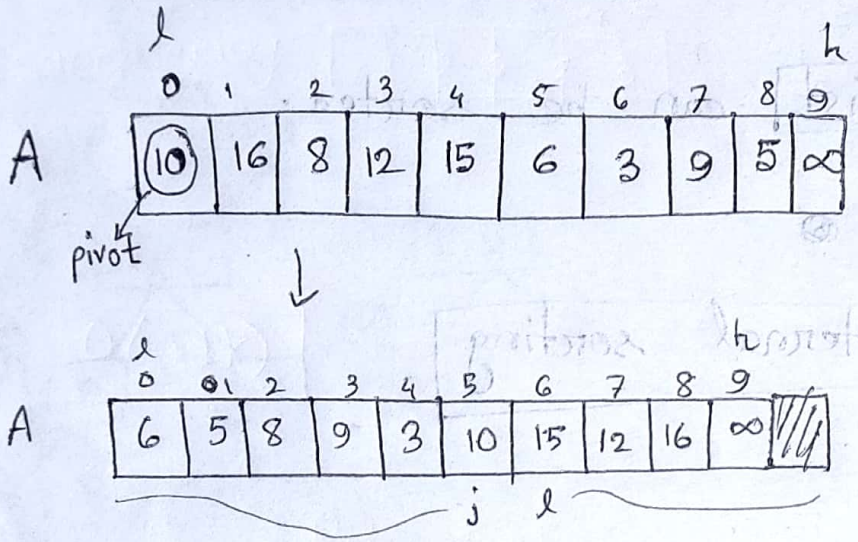
[ insertion sort → $O(n^2)$,
merge sort → $O(n \log n)$

*for small numbers ($n \leq 15$) merge sort is slower because of recursion. After that it becomes faster than insertion sort ]

3. Recursive (use stack). [takes $O(n + \log n)$ space $\Theta(n)$ ]

# Quick Sort

Idea: A number is in the sorted position if all its left elements are smaller and the right elements are greater than the number.




```
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){
n → j = partition(l, h);
    Quicksort(l, j);
    Quicksort(j+1, h);
  }
}
```

~~T.C = O(nlogn)~~

Best case → $O(n \log n)$ [Partition is always in the middle]

worst case → $O(n^2)$ [If the list is already sorted]

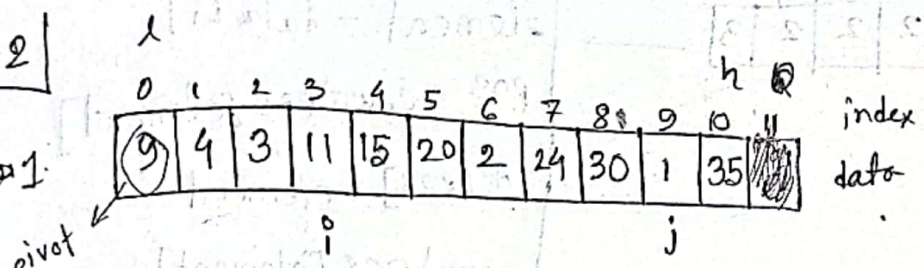
* Removing worst case of quick sort.

(i) select middle element as pivot

(ii) select random element as pivot

Ex-2

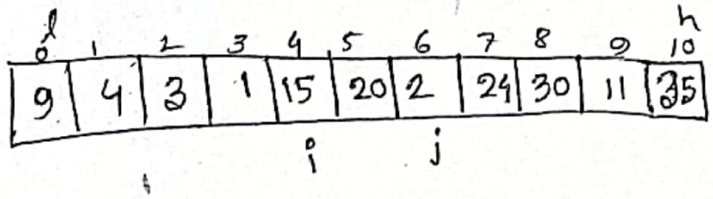
1.




if a[i] <= pivot {i++};

i = 3

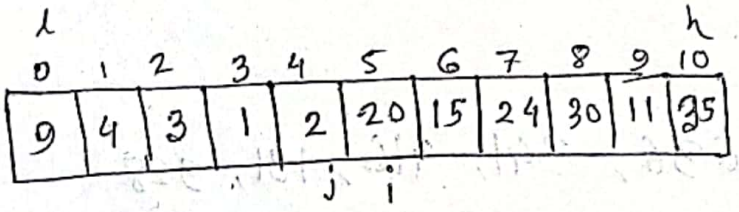
2.




if a[j] > pivot {j--}; i=4, j=6

swap a[j], a[i]

3.

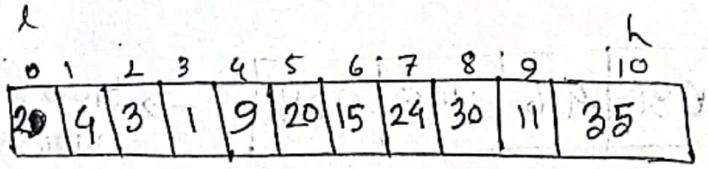



i<j  i=5, j=4

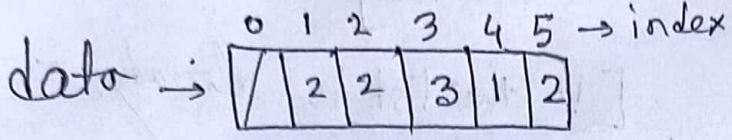
swap a[i] & a[j]

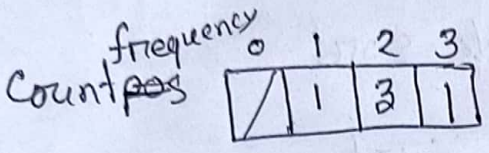
swap a[i] & pivot

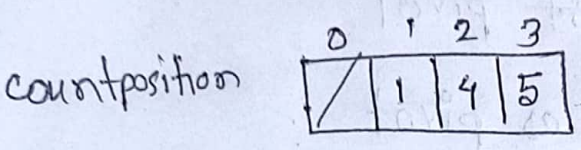
4.

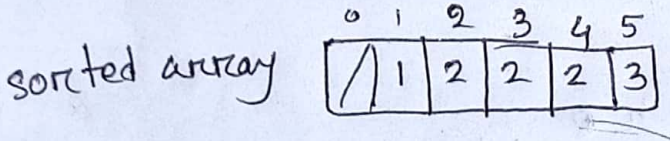


# Counting sort




```
countpos[data[i]]++

O(n)
countpos [data[i]] ++

O(k)
countpos[i] = countpos[i-1]
              + countpos[i]

O(n)
element = data[i]
pos = countpos[element]
SA[pos] = element
countpos[element] --;
```

T.C = O(n+k)

# Radix sort

Input list: 126 (d digit), 328, 636, 341, 416, 131, 328

pass - 1 :

| 341 | 131 | 126 | 636 | 416 | 328 | 328 |
|---|---|---|---|---|---|---|

→ প্রথম ডিজিট সাপেক্ষে

pass - 2 :

| 416 | 126 | 328 | 328 | 131 | 636 | 341 |
|---|---|---|---|---|---|---|

→ দ্বিতীয় ডিজিট সাপেক্ষে

pass - 3 :

| 126 | 131 | 328 | 328 | 341 | 416 | 636 |
|---|---|---|---|---|---|---|

→ তৃতীয় ডিজিট সাপেক্ষে

If order is fixed, it is called stable sorting algorithm

no. of element = n
base = k
digits = d

algorithm:

```
maxelement = max(Arr);
for (pos = 1; maxelement/pos != 0; pos = pos * 10)
    stablesort(Arr, Pos) ⟶ [counting sort (Arr, pos)]
```

$$\therefore \boxed{\begin{aligned} T.C &= O(d(n+k)) \\ &= O(n+k) \end{aligned}}$$

# Greedy

For solving optimization problem :

1. Greedy 2. Dynamic Programming 3. Branch & Bound

## General method :

```
Algorithm Greedy (a, n)
{
    for i = 1 to n do
    {
        x = Select (a);
        if Feasible (x) Then
            solution = solution + x;
    }
}
```

n = 5

| $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |

## knapsack problem

| objects : | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| profits : | 10 | 5 | | | | | | |

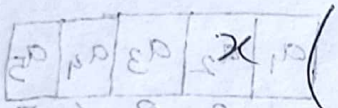
# knapsack

$n = 7$ objects : 0 : 1 2 3 4 5 6 7 $\quad 0 \le x \le 1$

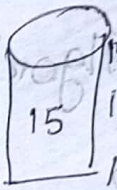
$m = 15$ profits : P : 10 5 15 7 6 18 3

weights : w : 2 3 5 7 1 4 1

| $\frac{P}{w}$ : | 5 | 1·3 | 3 | 1 | 6 | 4·5 | 3 |
|---|---|---|---|---|---|---|---|



$$\begin{pmatrix} 1 & 2/3 & 1 & 0 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{pmatrix}$$




15 − 1 = 14

14 − 2 = 12

12 − 4 = 8

8 − 5 = 3

3 − 1 = 2

2 − 2 = 0

$$\sum x_i w_i = 1\times 2 + 2/3 \times 3 + 1\times 5 + 0 \times 7 + 1\times 1 + 1\times 4 + 1\times 1$$

$$= 2 + 2 + 5 + 1 + 4 + 1 = 15$$

$$\sum x_i p_i = 1\times 10 + \frac{2}{3}\times 5 + 1\times 15 + 1\times 6 + 1\times 18 + 1\times 3$$

$$= 10 + 2\times 1{\cdot}3 + 15 + 6 + 18 + 3 = 54{\cdot}6$$

**Constraint**

$$\sum x_i w_i \le m$$

**objective**

$$\max \sum x_i p_i$$

# Job sequencing with Deadlines

Ex-1

$n = 5$

| Jobs: | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ |
|---|---|---|---|---|---|
| profits: | 20 | 15 | 10 | 5 | 1 |
| deadines: | 2 | 2 | 1 | 3 | 3 |

Rough

0 $\underline{J_2}$ 1 $\underline{J_1}$ 2 $\underline{J_4}$ 3

$\{J_1, J_2, J_4\}$

$J_1 \rightarrow J_2 \rightarrow J_4$ / $J_2 \rightarrow J_1 \rightarrow J_4$

$\phi$

Profit = 20+15+5 = 40

0-1-2-3

| Job consider | slot assign | solution | Profit |
|---|---|---|---|
| – | – | $\phi$ | 0 |
| $J_1$ | [0,1] | $J_1$ | 20 |
| $J_2$ | [0,1] [1,2] | $J_1, J_2$ | 20+15 |
| $J_3$ ✗ | [0,1] [1,2] | $J_1, J_2$ | 20+15 |
| $J_4$ | [0,1] [1,2], [2,3] | $J_1, J_2, J_4$ | 20+15+5 |
| $J_5$ ✗ | [0,1] [1,2] [2,3] | $J_1, J_2, J_4$ | 20+15+5 = 40 |

Ex-2

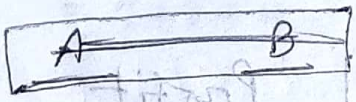
$n = 7$

| Jobs: | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ | $J_6$ | $J_7$ |
|---|---|---|---|---|---|---|---|
| Profits: | 35 | 30 | 25 | 20 | 15 | 12 | 5 |
| deadlines: | 3 | 4 | 4 | 2 | 3 | 1 | 2 |

0 —$J_6$— 1 —$J_3$— 2 —$J_1$— 3 —$J_2$— 4

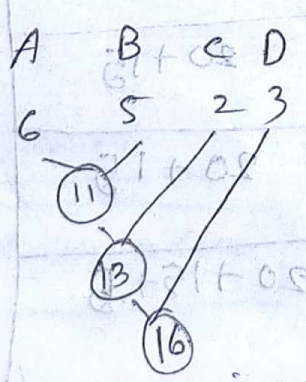
$20 + 15 + 35 + 30 = 110$

## Optimal merge pattern

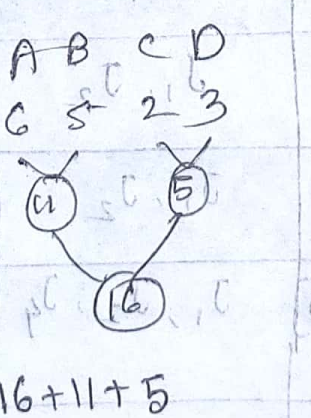



List → A B C D

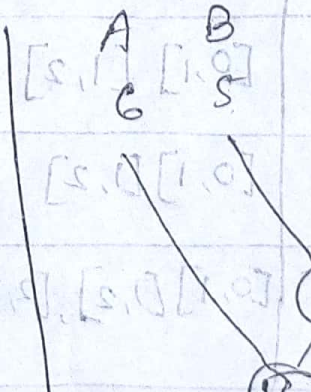
sizes → 6 5 2 3




$11 + 13 + 16 = 40$ sizes



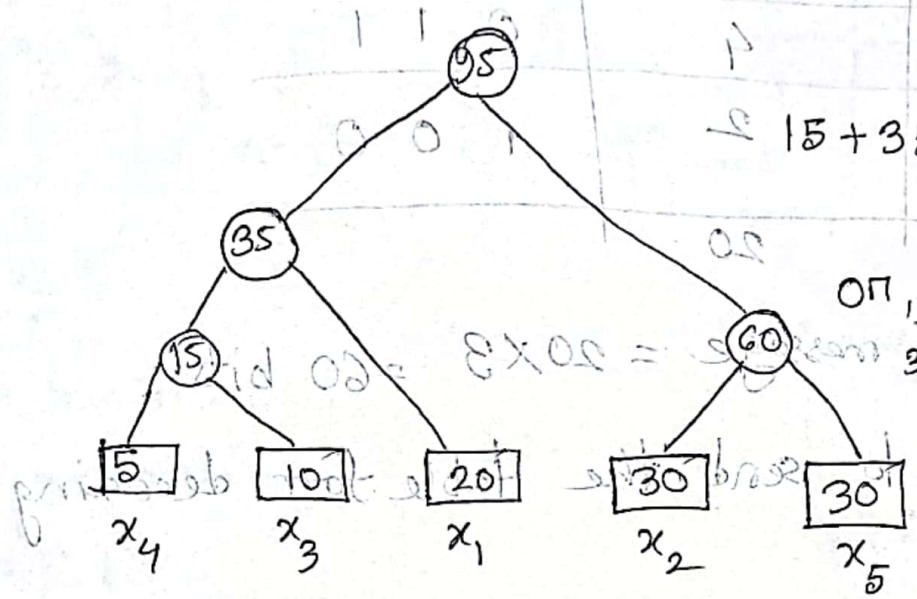

$16 + 11 + 5 = 32$ sizes




$16 + 10 + 5 = 31$ sizes

| Lists → | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
|---|---|---|---|---|---|
| sizes → | 20 | 30 | 10 | 5 | 30 |

[minimum patterns should be taken]




$15+35+60+95 = 205$

sizes

OR,

$3\times5 + 3\times10 + 2\times20 + 2\times30 + 2\times30 = 205$

$\sum d_i * x_i$

## Huffman coding

Message → BCCABBDDAECC BBAEDDCC (~~Length = 20~~)

Length = 20

ASCII - 8 bit

∴ Total bit to bit sent = $8 \times 20 = 160$ ~~bits~~ bits

## Fixed size codes

| Character | count / frequency | code |
|---|---|---|
| A | 3 | 000 |
| B | 5 | 001 |
| C | 6 | 010 |
| D | 4 | 011 |
| E | 2 | 100 |
| | 20 | |

Now, size of message = 20×3 = 60 bits

Now, we have to send the table for decoding the message

5×8 bit (5 character) = 40 bits

5×3 (3 bit for each character) = 15

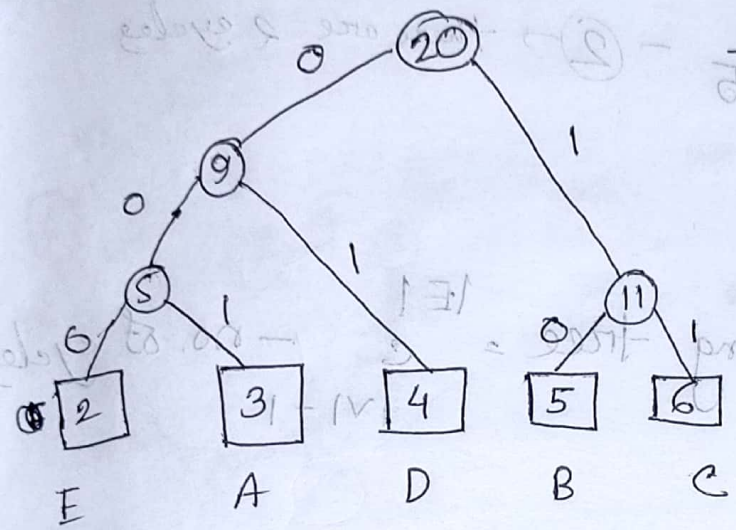
Table bits = 40 + 15 = 55

∴ Total number of bits = 55 + 60 = 115 bits

Huffman coding

BccABBDDAEccBBAEDDcc

10 11 11 00 10 - - -




| char | count | code | |
|---|---|---|---|
| A | 3 | 0 0 1 | 3×3 = 9 |
| B | 5 | 1 0 | 5×2 = 10 |
| C | 6 | 1 1 | 6×2 = 12 |
| D | 4 | 0 1 | 4×2 = 8 |
| E | 2 | 0 0 0 | 2×3 = 06 |
| 5×8 = 40 bits | 20 | 12 bits | 45 bits |

$\sum d_i * f_i = 3\times2 + 3\times3 + 2\times4 + 2\times5 + 2\times6$

$= 45$ bits

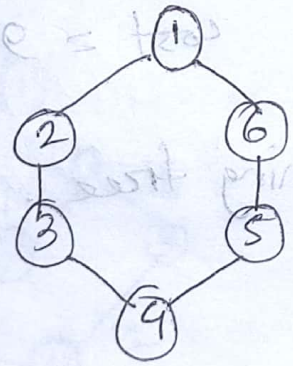
∴ Message size = 45 bits

For table = 40+12 = 52 bits

∴ Total bits = 45+52

= 97 bits

## Minimum cost spanning Tree




G(V,E)

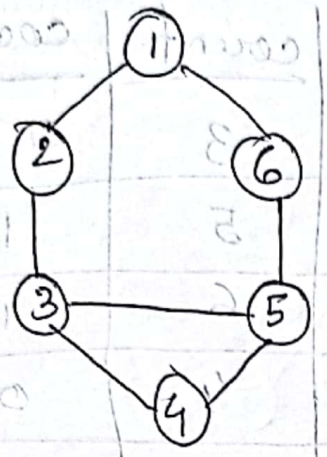
V = {1, 2, 3, 4, 5, 6}

E = {(1,2), (2,3), (3,4) ---}

|V| = 6

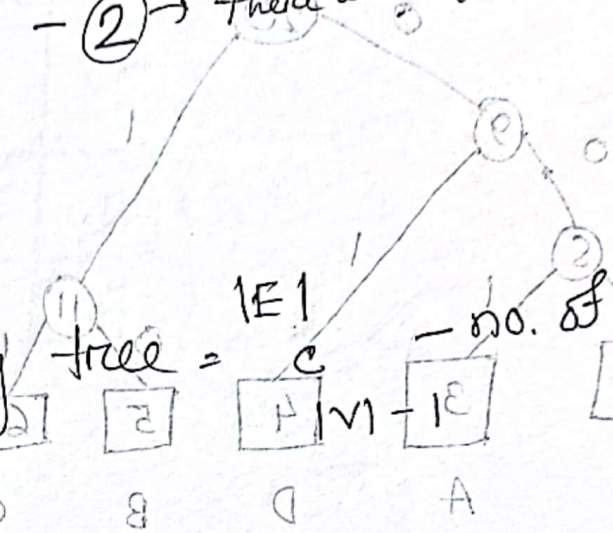
|V|-1 = 5 (spanning Tree)

${}^6C_5$ spanning Trees can be formed

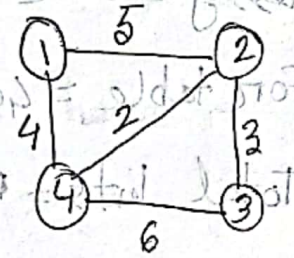


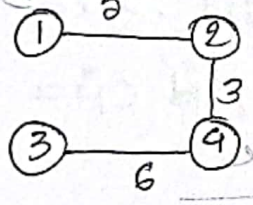

No. of spanning Trees

$= {}^{7}C_{5}$ − (2) → there are 2 cycles

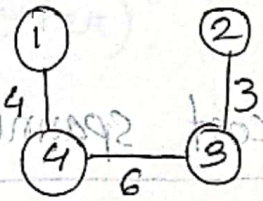



Total no. of spanning tree $= {}^{|E|}C_{|V|-1}$ − no. of cycles

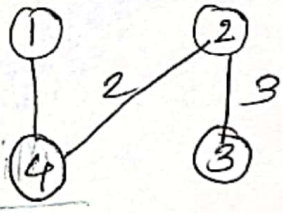



cost = 14




cost = 13



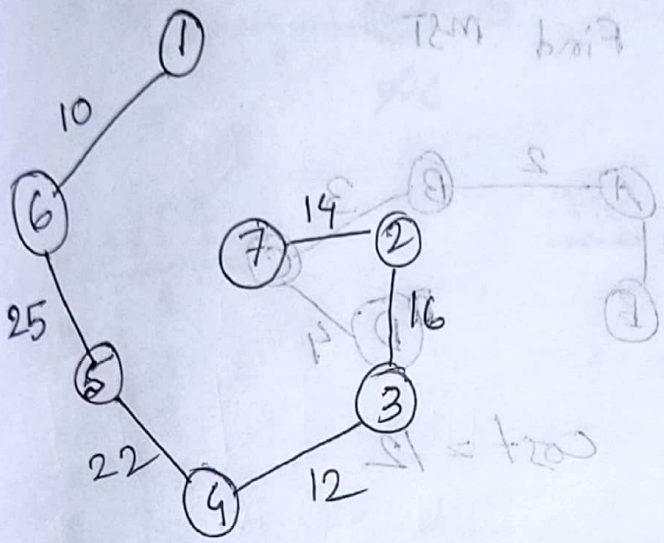

cost = 9

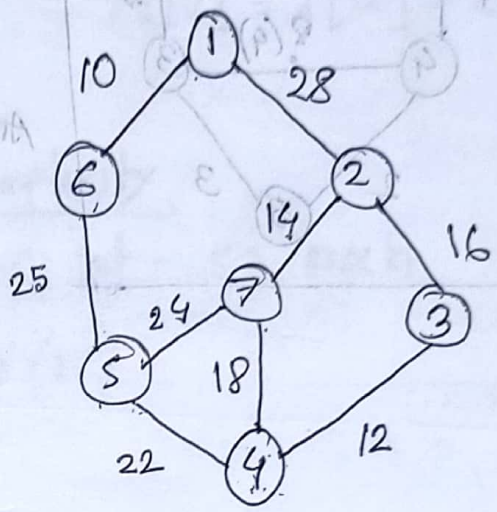
Process to find minimum spanning tree.

1. Prim's
2. Kruskal

# Prim's algorithm

(always select a minimum edge which is connected)



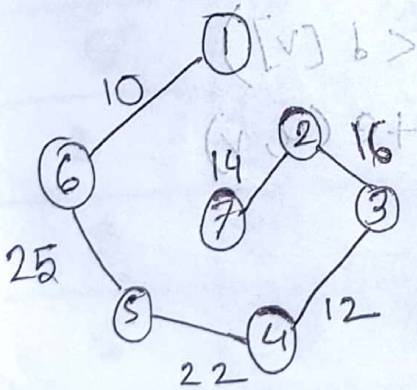




cost = 99

[For non-connected graph, spanning tree can't be found]

# Kruskal's algorithm

[always select minimum cost edge without forming a cycle]




cost = 99

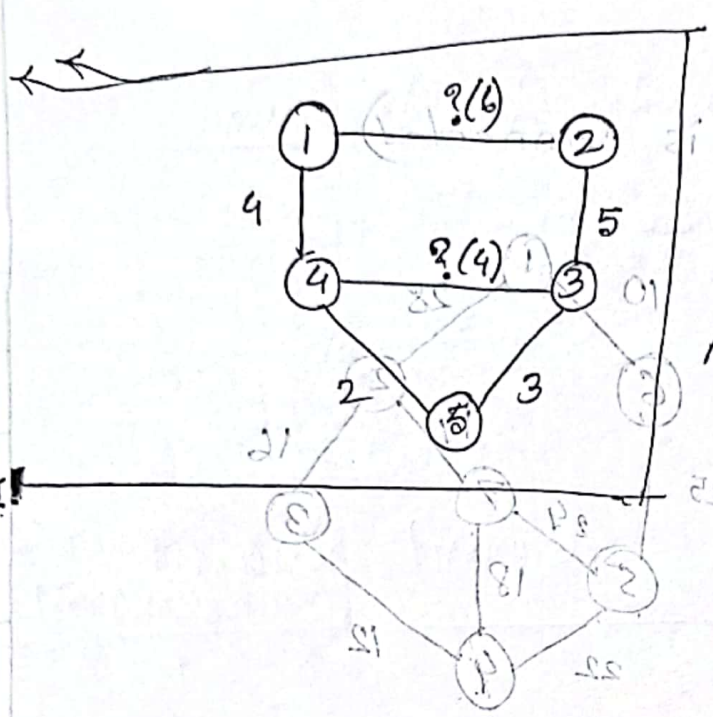
## Complexity (normal)

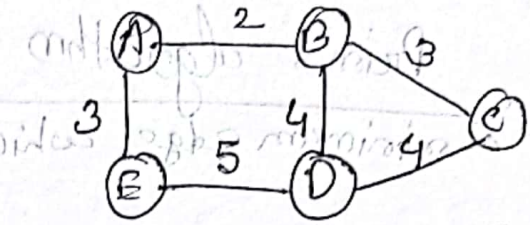
$\Theta(|V|\,|E|)$

$\Theta(n \cdot e) = \Theta(n^2)$

But by using min heap, it will be

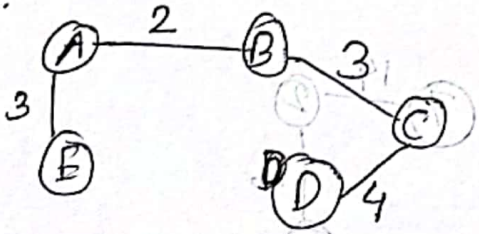
$\Theta(n \log n)$



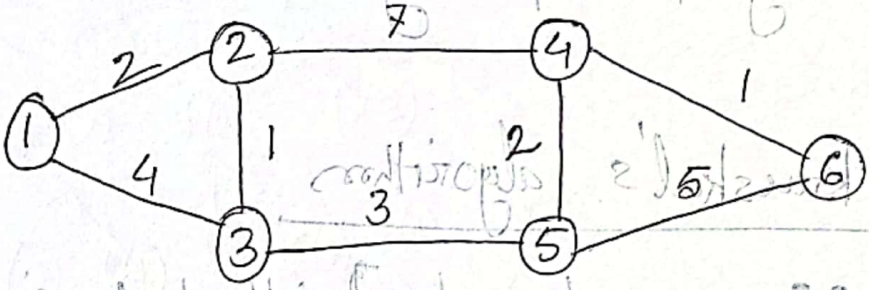

Find MST.

Ans:

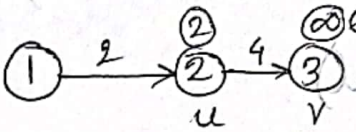



Cost = 12

# Dijkstra Algorithm (shortest path)



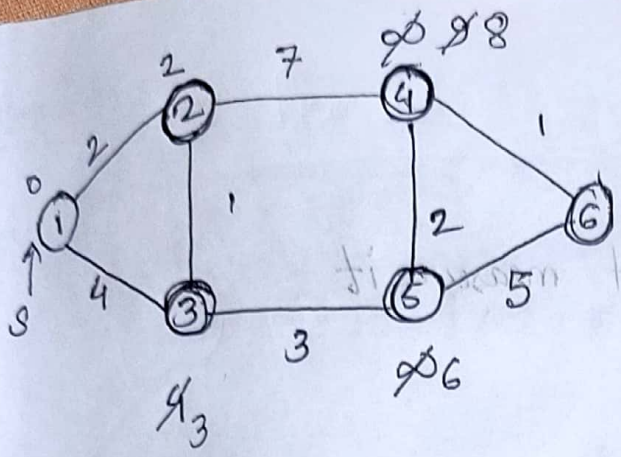

## Relaxation




$2+4<\infty$

if $(d[u] + c(u,v) < d[v])$

$d[v] = d[u] + c(u,v)$




## Relaxation

if $(d[u] + c(u,v) < d[v])$

$d[v] = d[u] + c(u,v)$

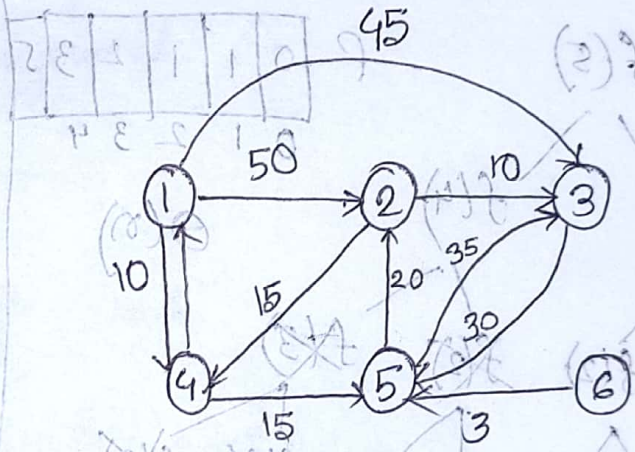
## Time complexity

$n = |V|$ $|V|$ so, $n \times n$

$\therefore \Theta(n^2)$

| v | d[v] |
|---|---|
| 2 | 2 |
| 3 | 3 |
| 4 | 8 |
| 5 | 6 |
| 6 | 9 |

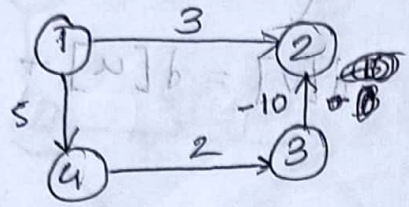
**Ex:**




starting vertex - 1

| Selected vertex | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| 4 | 50 | 45 | (10) | ∞ | ∞ |
| 5 | 50 | 45 | (10) | (25) | ∞ |
| 2 | (45) | 45 | (10) | (25) | ∞ |
| 3 | (45) | (45) | (10) | (25) | ∞ |
| 6 | (45) | (45) | (10) | (25) | ∞ |

## Drawback of dijkstra

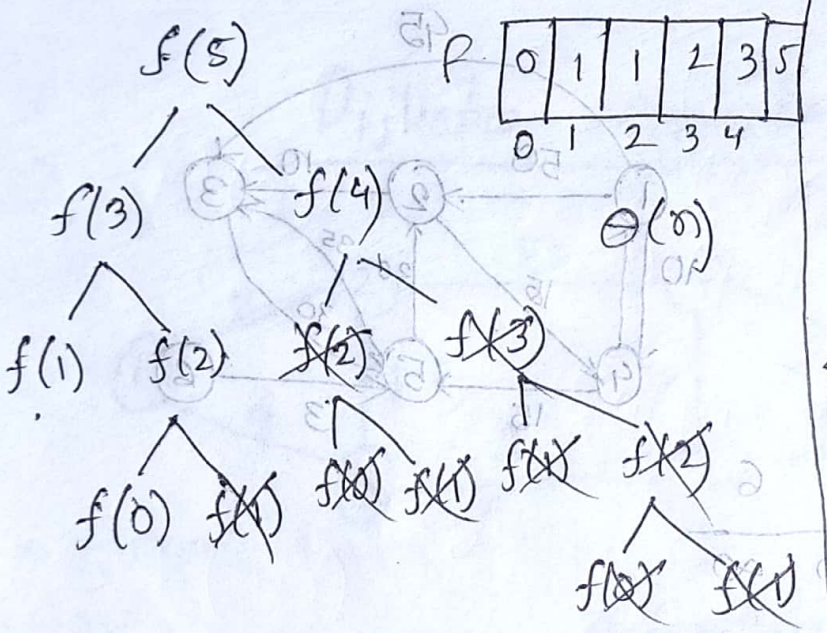



it can't measure it

## Dynamic Programming

**memorization**

0, 1, 1, 2, 3, 5, 8 ---







$\Theta(n)$

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ f(n-2)+f(n-1) & \text{if } n>1 \end{cases}$$

```
int fib(int n)
{
  if (n <= 1)
    return n;
  return f(n-2) + f(n-1);
}
```

~~Top down~~

Top to bottom approach

Tabular

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ f(n-2)+f(n-1) & \text{if } n>1 \end{cases}$$

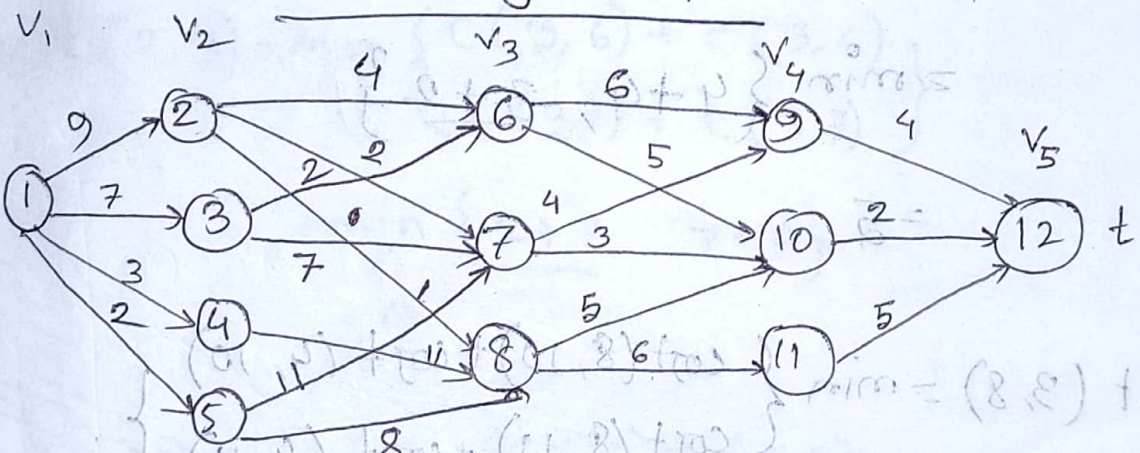
| F | 0 | 1 | 1 | 2 | 3 | 5 |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |

```
int f(int n)
  if (n<=1)
    return n
F[0] = 0, F[1] = 1
```

```
for (int i=2; i<=n; i++)
{
  F[i] = F[i-2] + F[i-1]
}
return F[n];
}
```

Bottom to Top approach

## Multistage Graph




| V | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| cost | 16 | 7 | 9 | 18 | 15 | 7 | 5 | 7 | 4 | 2 | 5 | 0 |
| d | 2/3 | 7 | 6 | 8 | 8 | 10 | 10 | 10 | 12 | 12 | 12 | 12 |

↓ or

stage vertex

(i) $cost(5,12) = 0$

(ii) $cost(4,9) = 4$, $cost(4,10) = 2$, $cost(4,11) = 5$

(iii) $cost(3,6) = \min \begin{Bmatrix} cost(6,9) + cost(4,9) \\ cost(6,10) + cost(4,10) \end{Bmatrix}$

$= \min \{ 6+4, \underline{5+2} \}$

$= 7$

$cost(3,7) = \min \begin{Bmatrix} cost(7,9) + cost(4,9) \\ cost(7,10) + cost(4,10) \end{Bmatrix}$

$= \min \{ 4+4, \underline{3+2} \}$

$= 5$

$cost(3,8) = \min \begin{Bmatrix} cost(8,10) + cost(4,10) \\ cost(8,11) + cost(4,11) \end{Bmatrix}$

$= \{ \underline{5+2}, 6+5 \}$

$= 7$

(iv) $cost(\overset{2,2}{\bigcirc}) = \min \left\{ \begin{matrix} c(2,6)+c(3,6) \\ c(2,7)+c(3,7) \\ c(2,8)+c(3,8) \end{matrix} \right\}$

$= \{ 4+7, \underline{2+5}, 1+7 \}$

$= 7$

$cost(2,4) = \min \{ c(4,8) + c(3,8) \}$

$= \min \{ 11+7 \}$

$= 18$

$cost(2,3) = \min \left\{ \begin{matrix} c(3,6)+c(3,6) \\ c(3,7)+c(3,7) \end{matrix} \right\}$

$= \min \{ \underline{2+7}, 7+5 \}$

$= 9$

$cost(2,5) = \min \left\{ \begin{matrix} c(5,7)+c(3,7) \\ c(5,8)+c(3,8) \end{matrix} \right\}$

$= \min \{ 11+5, \underline{8+7} \}$

$= 15$

(v) $cost(1,1) = \min \begin{Bmatrix} c(1,2)+c(2,2), \\ c(1,3)+c(2,3), \\ c(1,4)+c(2,4), \\ c(1,5)+c(2,5) \end{Bmatrix}$

$= \min \{9+7, 7+9, 3+18, 2+15\}$

$= 16$

**Formula:**

stage, vertex no.

$$cost(i,j) = \min \{ c(j,l) + c(i+1,l) \}$$

where $\langle j,l \rangle \in E$, $l \in V_{i+1}$

Now, using dynamic programming,

for 2

stage, vertex

$d(1,1) =$ (2) → next vertex

$d(2,2) = 7$

$d(3,7) = 10$

$d(4,10) = 12$

for 3

$d(1,1) = 3$

$d(2,3) = 6$

$d(3,6) = 10$

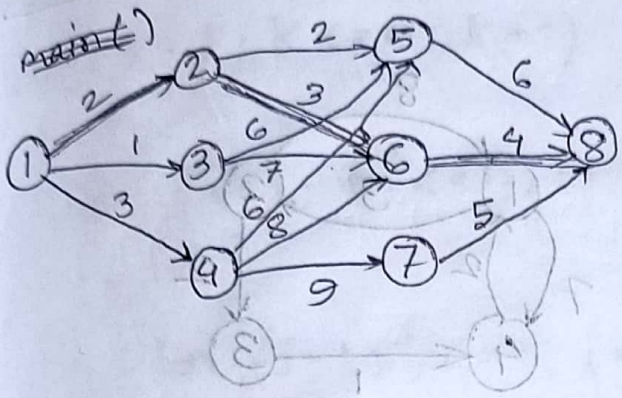
$d(4,10) = 12$

Both results are same

and ~~this~~ these are the shortest path

## Program

```
main(){
    int stages=4, min;
    int n=8
    int cost[9], d[9], path[9]
    int c[9][9] = {{0,0,0,0,0,0,0,0,0},
                   {0,0,2,1,3,0,0,0,0},
                   {0,0,0,0,0,6,7,0,0},
                   {0,0,0,0,0,0,6,8,9,0}
                   {0,0,0,0,0,0,0,2,3,0,0}
    }
    cost[n] = 0
    for(int i=n-1; i>=1; i--){
        min = 32767;
        for(k=i+1; k<=n; k++){
            if(c[i][k]!=0 && c[i][k]
                + c[k]<min)
            {
                min = c[i][k]+c[k];
                d[i] = k;
            }}
        cost[i] = min;
    }
    p[1] = 1; p[stages] = n;
    for(i=2; i<stages; i++){
        p[i] = p[d[i-1]];
    }
```




c =

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

cost

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| | 9 | 7 | 11 | 12 | 6 | 4 | 5 | 0 |

d

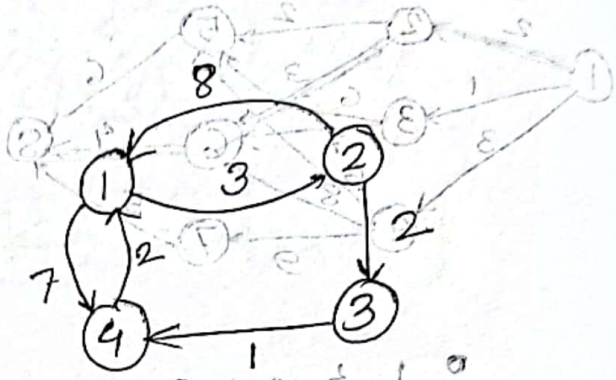
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| | 2 | 6 | 6 | 5 | 8 | 8 | 8 | |

path

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 6 | 8 | | | | |

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

# All pair shortest path (Floyd-Warshall)




$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 8 & \infty & 0 \end{bmatrix} \end{array}$$

$A^0[2,3]$ : $A^0[2,1] + A^0[1,3]$

$2 < 8 + \infty$

$A^0[2,4]$ $A^0[2,1] + A^0[1,4]$

$\infty > 8 + 7$

$A^0[3,2]$ : $A^0[3,1] + A^0[1,2]$

$\infty > 5 + 3$

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 2 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$A^1[1,3]$ $A^1[1,2] + A^1[2,3]$

$\infty > 3 + 2$

$A^1[1,4]$ $A^1[1,2] + A^1[2,4]$

$7 < 3 + 15$

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 8 & 7 & 0 \end{bmatrix} \end{array}$$

$A^2[1,2]$ $A^2[1,3] + A^2[3,2]$

$3 < 5 + 8$

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 7 & 0 & 2 & 3 \\ 5 & 8 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

$$A^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 7 & 0 \end{bmatrix} \end{array}$$

**Formula:**

$$A^k[i,j] = \min\left\{ A^{k-1}[i,j],\ A^{k-1}[i,k] + A^{k-1}[k,j] \right\}$$

Code:

```
for (k = 1; k <= n; k++)
{
  for (i = 1; i <= n; i++)
  {
    for (j = 1; j <= n; j++)
    {
      A[i,j] = min(A[i,j], A[i,k] + A[k,j])
    }
  }
}
```

$T.C. = O(n^3)$

## Matrix-chain Multiplication

$A_1 \times A_2 \times A_3$

2 3 3 4 4 2

$d_0\ d_1 \quad d_1\ d_2 \quad d_2\ d_3$

$(A_1 \times A_2) \times A_3$

2 3 3 4 4 2

$= 2 \times 3 \times 4$

$= 24$ (multiplication)

4 2

0

Total $= 24 + 0 + (2 \times 4 \times 2) = 40$

$A_1 \times (A_2 \times A_3)$

2 3 3 4 4 2

0 $\quad 3 \times 4 \times 2 = 24$

Total $= 0 + 24 + (2 \times 3 \times 2)$

$= 36$ (multiplication)

## Formula:

$(A_1 \times A_2) \times A_3$

2 3 3 4 4 2

$c[1,2] = 24$ $c[3,3] = 0$

2 4 4 2

$d_0$ $d_2$ $d_2$ $d_3$

$c[1,2] + c[3,3] + d_0 \times d_2 \times d_3 = 40$

$i \; k \quad k+1 \; j$

$d_{i-1} \; k \; d_j$

$A_1 \times (A_2 \times A_3)$

2 3 3 4 4 2

$c[1,1]$ $c[2,3] = 24$

0

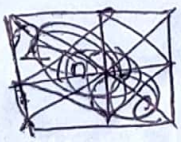
$c[1,1] + c[2,3] + d_0 \times d_1 \times d_3 = 36$

$i \; k \quad k+1 \; j$

$d_{i-1} \; k \; d_j$

$$\therefore c[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

## Formula for matrix chain Multiplication



$$\frac{2(n-1)\, C_{(n-1)}}{n} \qquad \frac{2\times 3}{4} C_3 = 5 \text{ (for 4 matrix)}$$

$A_1 \times A_2 \times A_3 \times A_4$

1. $A_1 (A_2 (A_3 A_4))$
2. $A_1 ((A_2 A_3) A_4)$
3. $(A_1 A_2)(A_3 A_4)$
4. $(A_1 (A_2 A_3)) A_4$
5. $((A_1 A_2) A_3) A_4$

$$c[i,j] = \min_{i \le k < j} \{ c[i,k] + c[k+1,j] + d_{i-1} \times d_k \times d_j \}$$

$A_1 \times A_2 \times A_3 \times A_4$

| $\frac{3}{d_0}$ | $\frac{2}{d_1}$ | $\frac{2}{d_2}$ | $\frac{4}{d_3}$ | $\frac{2}{d_4}$ |
|---|---|---|---|---|

| c | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 24 | 28 | 58 |
| 2 | | 0 | 16 | 36 |
| 3 | | | 0 | 40 |
| 4 | | | | 0 |

$$c[1,2] = \min_{1 \le k < 2}^{i\ j} \{ c[1,1] + c[2,2] + d_0 \times d_1 \times d_2 \}$$

$k=1$

$0 + 0 + 3 \times 2 \times 4$

$= 24$

$$c[2,3] = \min_{2 \le k < 3}^{i\ j} \{ c[2,2] + c[3,3] + d_1 \times d_2 \times d_3 \}$$

$k=2$

$0 + 0 + 2 \times 4 \times 2$

$= 16$

| k | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | 3 |
| 2 | | | 2 | 3 |
| 3 | | | | 3 |
| 4 | | | | |

$$c[3,4] = \min_{3 \le k < 4}^{i\ j} \{ c[3,3] + c[4,4] + d_2 \times d_3 \times d_4 \}$$

$k=3$

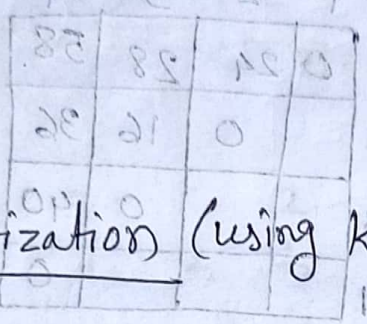
$0 + 0 + 4 \times 2 \times 5$

$= 40$

$$c[1,3] = \min_{1 \le k < 3} \begin{cases} k=1 & c[1,1] + c[2,3] + d_0 \times d_1 \times d_3 \\ & 0 + 16 + 3 \times 2 \times 2 \\ k=2 & c[1,2] + c[3,3] + d_0 \times d_2 \times d_3 \\ & 24 + 0 + 3 \times 4 \times 2 \end{cases} = \begin{cases} 28 \\ 48 \end{cases}$$

$$c[2,4] = \min_{2 \le k < 4} \begin{cases} k=2 & c[2,2] + c[3,4] + d_1 \times d_2 \times d_4 \\ & 0 + 40 + 2 \times 4 \times 5 \\ k=3 & c[2,3] + c[4,4] + d_1 \times d_3 \times d_4 \\ & 16 + 0 + 2 \times 2 \times 5 \end{cases} = \begin{cases} 80 \\ 36 \end{cases}$$

$$c[1,4] = \min_{1 \le k < 4} \begin{cases} k=1: & c[1,1] + c[2,4] + d_0 \times d_1 \times d_4 = 0 + 36 + 3 \times 2 \times 5 = 86 \\ k=2: & c[1,2] + c[3,4] + d_0 \times d_2 \times d_4 = 24 + 40 + 3 \times 4 \times 5 = 124 \\ k=3: & c[1,3] + c[4,4] + d_0 \times d_3 \times d_4 = 28 + 0 + 3 \times 2 \times 5 = 58 \end{cases}$$



Parentization (using K table)

$((A_1)(A_2\ A_3))(A_4)$

$\therefore (A_1 \cdot (A_2 A_3))\ A_4$

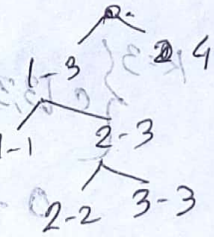
| K | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | 1 | 1 | 3 |
| 2 | | | 2 | 3 |
| 3 | | | | 3 |
| 4 | | | | |

T.C of parentization $= O(n^3)$

Program:

$A_1$ $A_2$ $A_3$ $A_4$

$5 \times 4$ $4 \times 6$ $6 \times 2$ $2 \times 7$




$(A_1(A_2 \cdot A_3))A_4$

| P | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | 5 | 4 | 6 | 2 | 7 |

S

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 3 |
| 2 | 0 | | 0 | 2 | 3 |
| 3 | 0 | | | 0 | 3 |
| | 0 | | | | 0 |

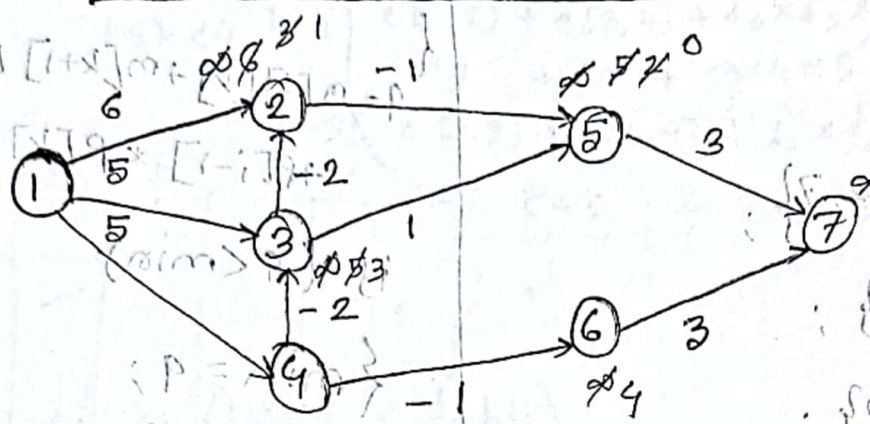
m

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 120 | 88 | 158 |
| 2 | 0 | | 0 | 48 | 104 |
| 3 | 0 | | | 0 | 84 |
| 4 | 0 | | | | 0 |

```
main()
{
int n = 5
int p[] = {5,4,6,2,7};
int m[5][5] = {0};
int s[5][5] = {0};

int j, min, q;
for (int d = 1; d < n-1; d++)
{
  for (int i = 1; i < n - d; i++)
  {
    j = i + d;
    min = 32767;
    for (int k = i; k <= j-1; k++)
    {
      q = m[i][k] + m[k+1][j]
          + p[i-1] * p[k] * p[j];
      if (q < min)
      {
        min = q;
        s[i][j] = k;
      }
    }
    m[i][j] = min;
  }
}
cout << m[1][n-1];
}
```

# Single source shortest path (Bellman - ford)




1 - 0
2 - 1
3 - 3
4 - 5
5 - 0
6 - 4
7 - 3

Relaxation [(n-1) times]

```
if (d[u] + c[u,v] < d[v])
    d[v] = d[u] + c(u,v)
```

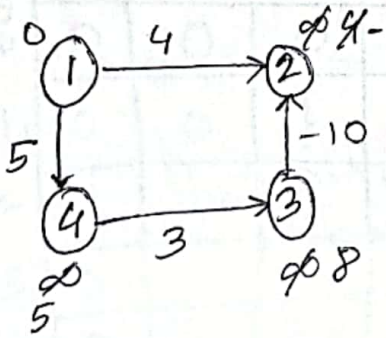
edges list → (1,2), (1,3), (1,4), (2,5), (3,2), (3,5), (4,3), (4,6), (5,7), (6,7)

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

[If it is complete graph [Every edge is connected to everyone), then T.C. will be $O(n^3)$]

---

Ex-2

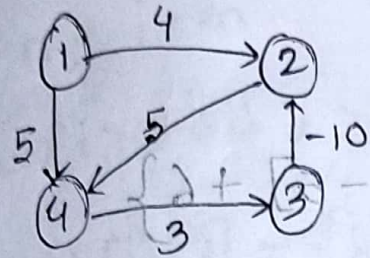


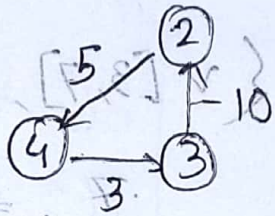

1 - 0
2 - (-2)
3 - 8
4 - 5

edges list: (3,2) (4,3) (1,4) (1,2)

n = 4 - 1 = 3

But, This graph can't be solved with it because there is a cycle in the graph in which total weight is negative.




$5 + 3 - 10 = -2$

if the total weight of the cycle was positive, it could be solved with this algorithm.

## 0/1 knapsack problem

$m = 8$

$P = \{1, 2, 5, 6\}$

$n = 4$

$w = \{2, 3, 4, 5\}$

$x_i = 0/1$

$x = \{1, 0, 0\}$

| $P_i$ | $w_i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 3 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 |
| 5 | 4 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7 | 7 |
| 6 | 5 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7 | 8 |

using formula: $v[i,w] = \max\{v[i-1,w], v[i-1, w-w[i]] + P[i]\}$

$v[4,1] = \max\{v[3,1], v[3, 1-5] + 6\}$

$V[4,5] = \max\{V[3,5], V[3,5-5]+6\}$

$5 < 0+6$

$V[4,6] = \max\{V[3,6], V[3,6-5]+6\}$

$6 = 0+6$

$V[4,7] = \max\{V[3,7], V[3,7-5]+6\}$

$7 = 1+6$

$V[4,8] = \max\{V[3,8], V[3,8-5]+6\}$

$7 < 2+6$

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | $8-6=2$, $2-2=0$ (Remaining profit) |

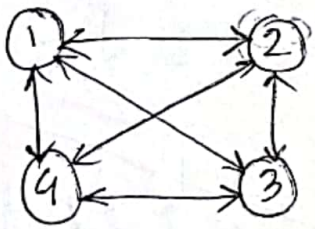
**Program:**

```
main()
{
int P[5] = {0,1,2,5,6}
int wt[5] = {0,2,3,4,5}
int m = 8, n = 4;
int K[5][9];
for (int i=0; i<=n; i++){
for (int w=0; w<=m; w++){
① if (i==0 || w==0){
K[i][w] = 0;
② else if (wt[i] <= w)
K[i][w] = max(P[i] +
K[i-1][w-wt[i]],
K[i-1][w]);
③ else K[i][w] = K[i-1][w];
}}
cout << K[n][w]
}
```

To see which objects are included:

```
i = n; j = m
while (i>0 && j>0)
{ if (k[i][j] == k[i-1][j])
    i--
  else{
    cout << i << " = 1";
    i--;
    j = j - wt[i];
  }
```

## Traveling salesman problem




| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 10 | 15 | 20 |
| 2 | 5 | 0 | 9 | 10 |
| 3 | 6 | 13 | 0 | 12 |
| 4 | 8 | 8 | 9 | 0 |

problem statement: start from a vertex and return to it with minimum cost.

$$g(i,S) = \min_{k \in S} \{ c_{ik} + g(k, S - \{k\}) \}$$

$$g(1,\{2,3,4\}) = \min_{k \in \{2,3,4\}} \{ c_{1k} + g(k, \{2,3,4\} - \{4\}) \}$$

Solve:

$g(2,\phi) = 5$

$g(3,\phi) = 6$

$g(4,\phi) = 8$

$g(2,\{3\}) = 15$

$g(2,\{4\}) = 18$

$g(3,\{2\}) = 18$

$g(3,\{4\}) = 20$

$g(4,\{2\}) = 13$

$g(4,\{3\}) = 15$

$g(2,\{3,4\}) = 25$

$g(3,\{2,4\}) = 25$

$g(4,\{2,3\}) = 23$

$g(1,\{2,3,4\}) = 35$ [minimum]

## Longest common subsequence (LCS)

string 1 : a b c d e f g h i j

string 2 : c d g i

~~string 3 : c d g i~~

c d g i (longest)

1 : a b d a c e

2 : b a b c e

bace

abce

m A

| b | d |
|---|---|
| 1 | 2 |

n B

| a | b | c | d |
|---|---|---|---|
| 1 | 2 | 3 | 4 |

| | | a | b | c | d |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| b 1 | 0 | 0 | 1 | 1 | 1 |
| d 2 | 0 | 0 | 1 | 1 | 2 |
| | | | b | | d |

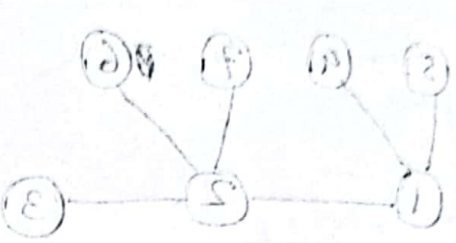
bd

T.C. = $\Theta(m \times n)$

str.1 : stone

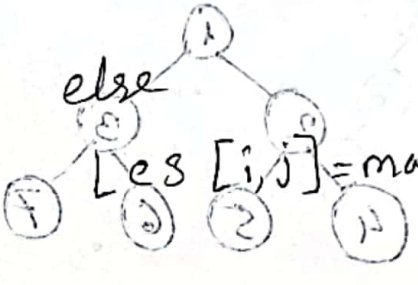
str.2 : longest

(one) (3)



if (A[i] = B[j])

Lcs[i, j] = 1 + Lcs [i-1, j-1]



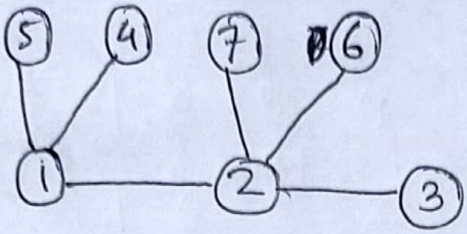
else

Lcs [i, j] = max (LCS[i-1, j], Lcs [i, j-1])

| | | l | o | n | g | e | s | t |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| s | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| t 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| o 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 2 |
| n 4 | 0 | 0 | 1 | 2 | 2 | 2 | 2 | 2 |
| e 5 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 |
| | | 0 | | n | | e | | |

# BFS & DFS

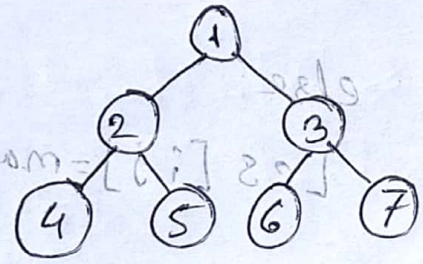
(1)




BFS : 1, 2, 4, 5, 7, 3, 6

DFS : 1, 2, 3, 6, 7, 4, 5

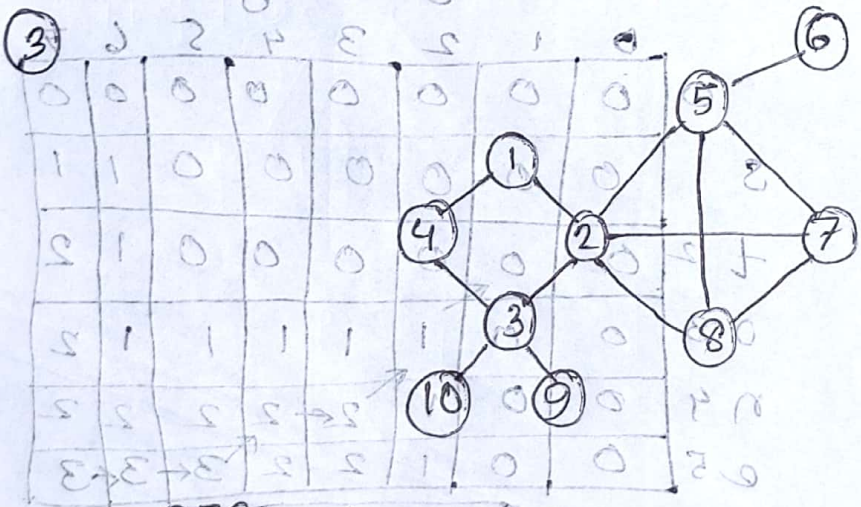
(2)




(level-order) BFS : 1, 2, 3, 4, 5, 6, 7

(Pre-order) DFS : 1, 2, 4, 5, 3, 6, 7

(3)

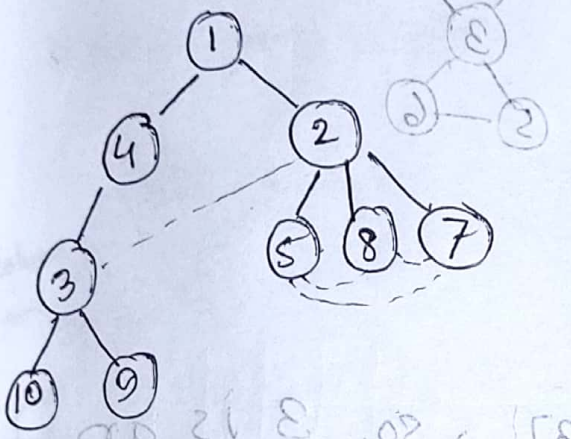


BFS : 1, 4, 2, 3, 5, 8, 7, 10, 9, 6

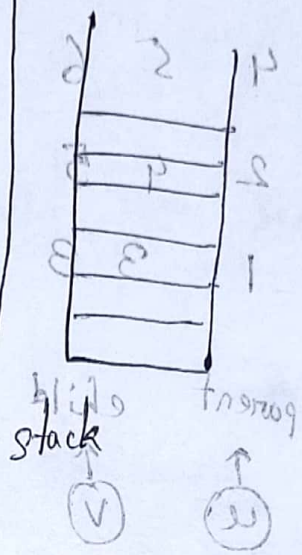
(Queue) Q

| 1 | 4 | 2 | 3 | 5 | 8 | 7 | 10 | 9 | 6 |
|---|---|---|---|---|---|---|---|---|---|

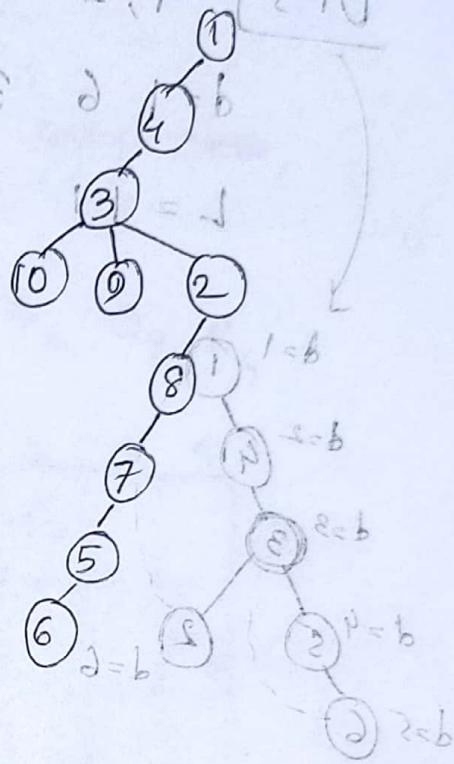
BFS spanning Tree



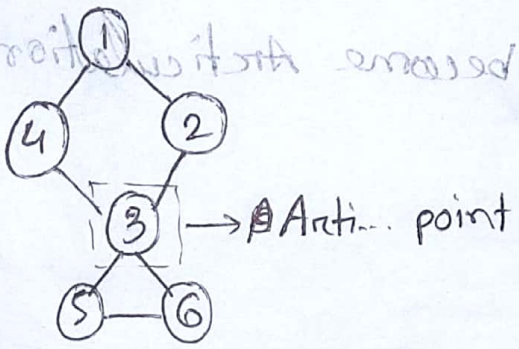
DFS : 1, 4, 3, 10, 9, 2, 8, 7, 5, 6

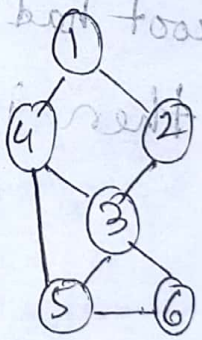



DFS spanning Tree



## Articulation point



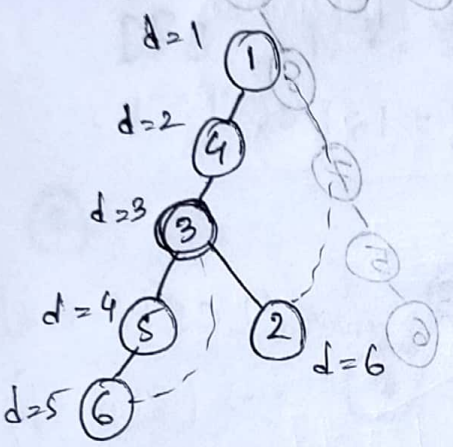
Now there is no A.P as 4, 5 are connected

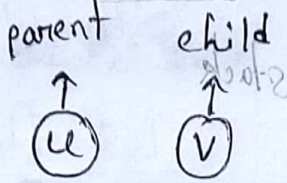
# To find Articulation point:

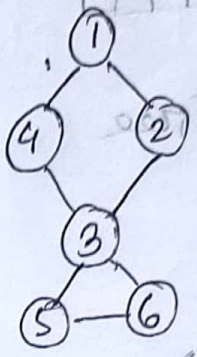
d = discovery point
L = Lowest number to it can go

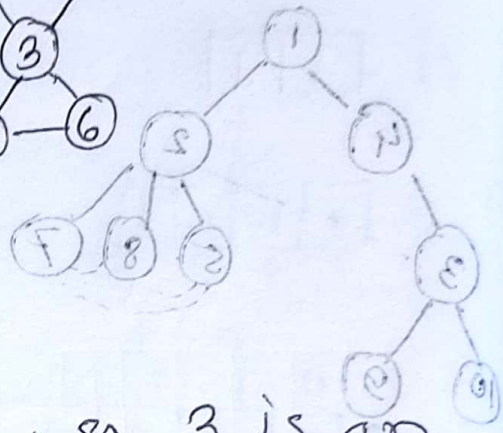
DFS

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| d = | 1 | 6 | 3 | 2 | 4 | 5 |
| L = | 1 | 1 | 1 | 1 | 3 | 3 |



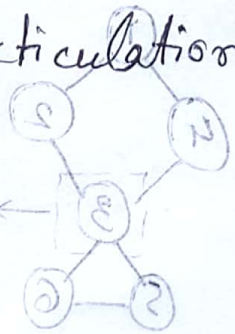


$L[v] \geq d[u]$

$L[5] \geq d[3]$ → So, 3 is an articulation point

$3 \geq 3$

[If root had one more vertices attached to it, then it will become Articulation point]



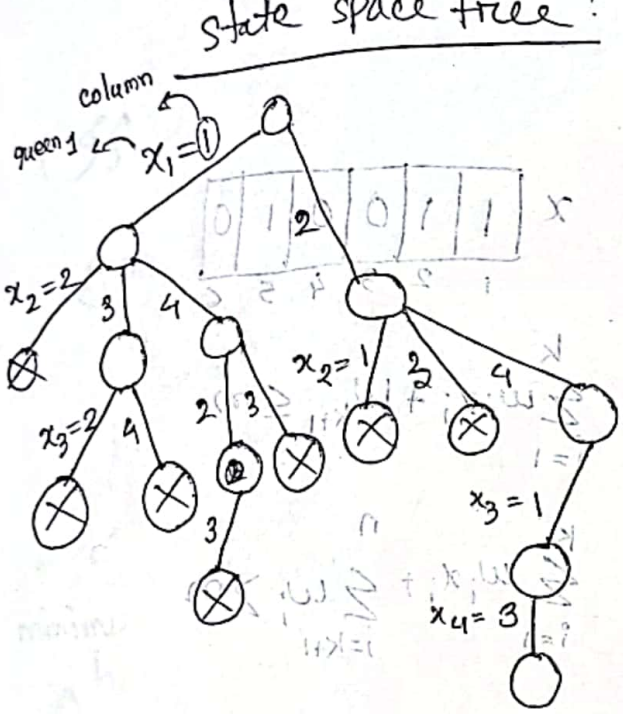
# Backtracking

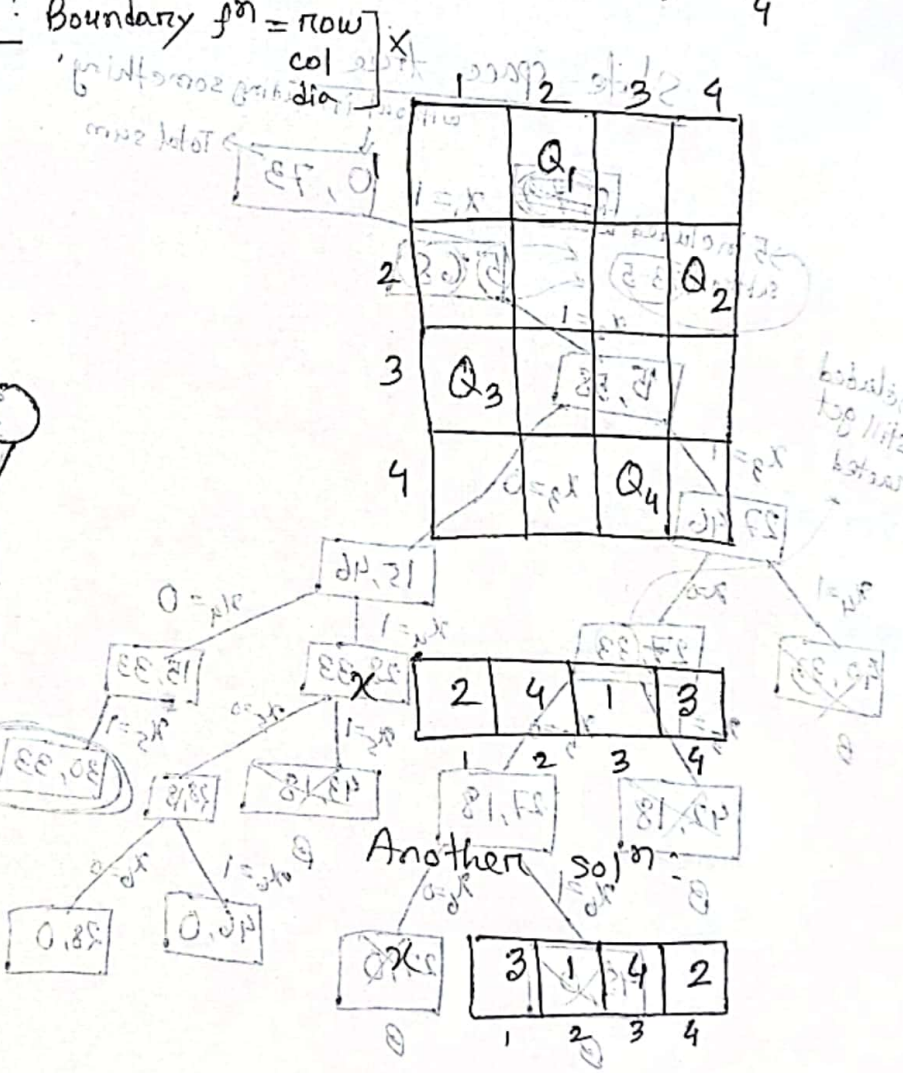
## N queens problem :

No queens can be placed in the same row, column and diagonal.

$Q_1$ $Q_2$ $Q_3$ $Q_4$

**State space tree :** Boundary fn = row, col, dia



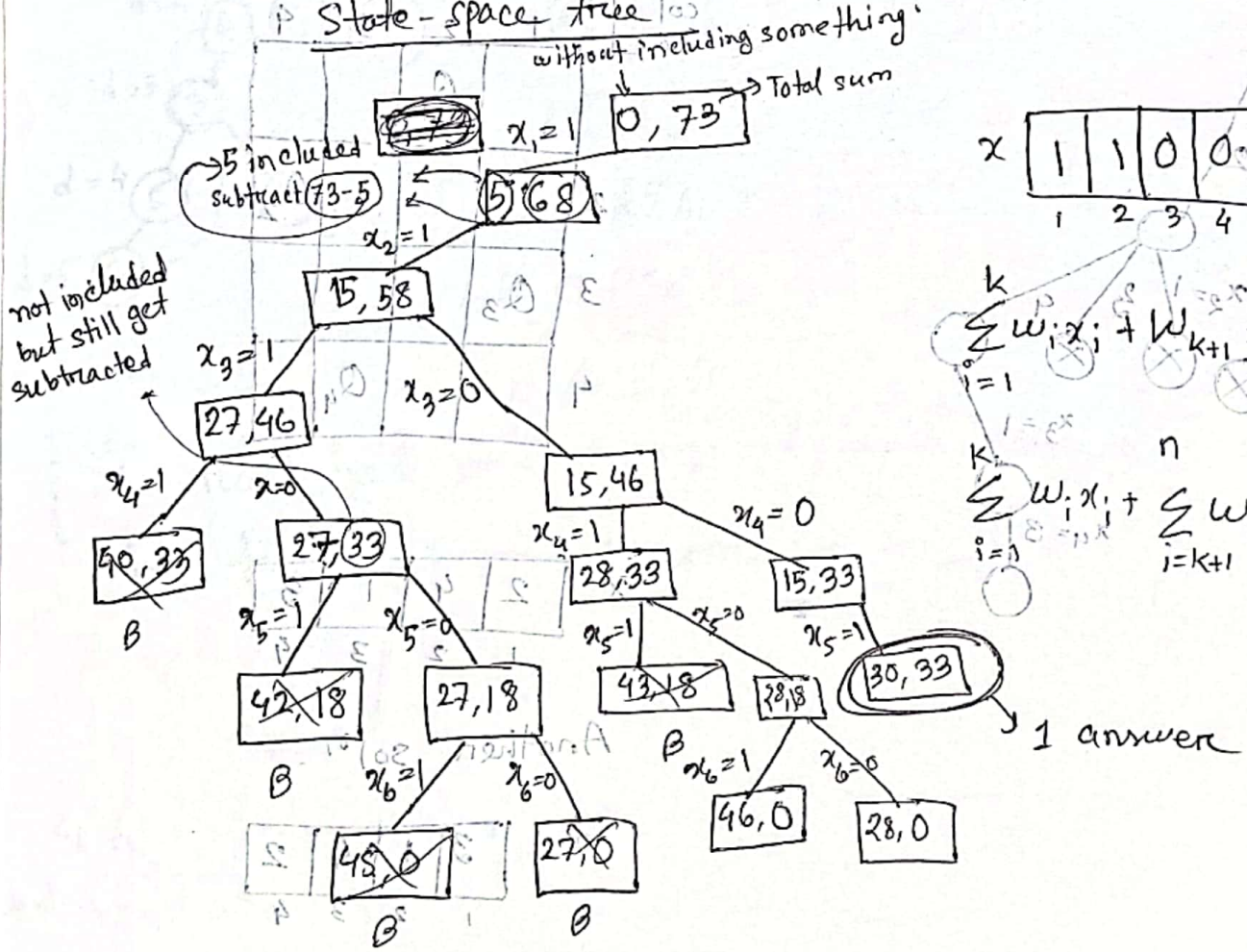
# Sum of subsets

Given set $w[1:6] = \{5, 10, 12, 13, 15, 18\}$ (1 2 3 4 5 6)

total numbers, $n = 6$

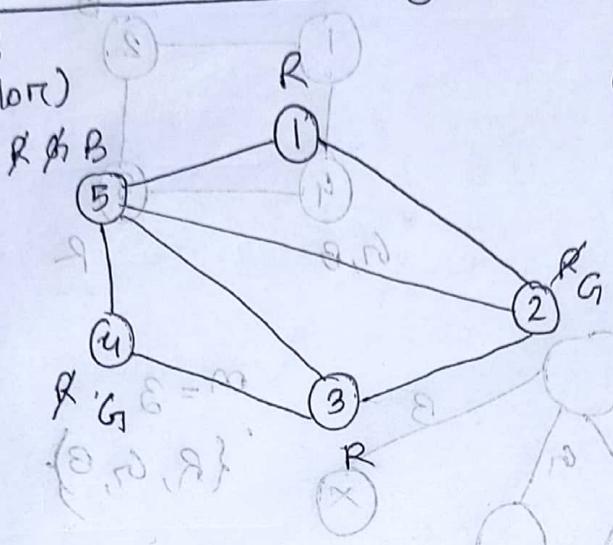
required sum, $m = 30$

State-space tree



# Graph colouring problem

(No two vertices can have same color)




Given color = R, G, B

1 Solution = 1 2 3 4 5

R G R G B

~~Given~~

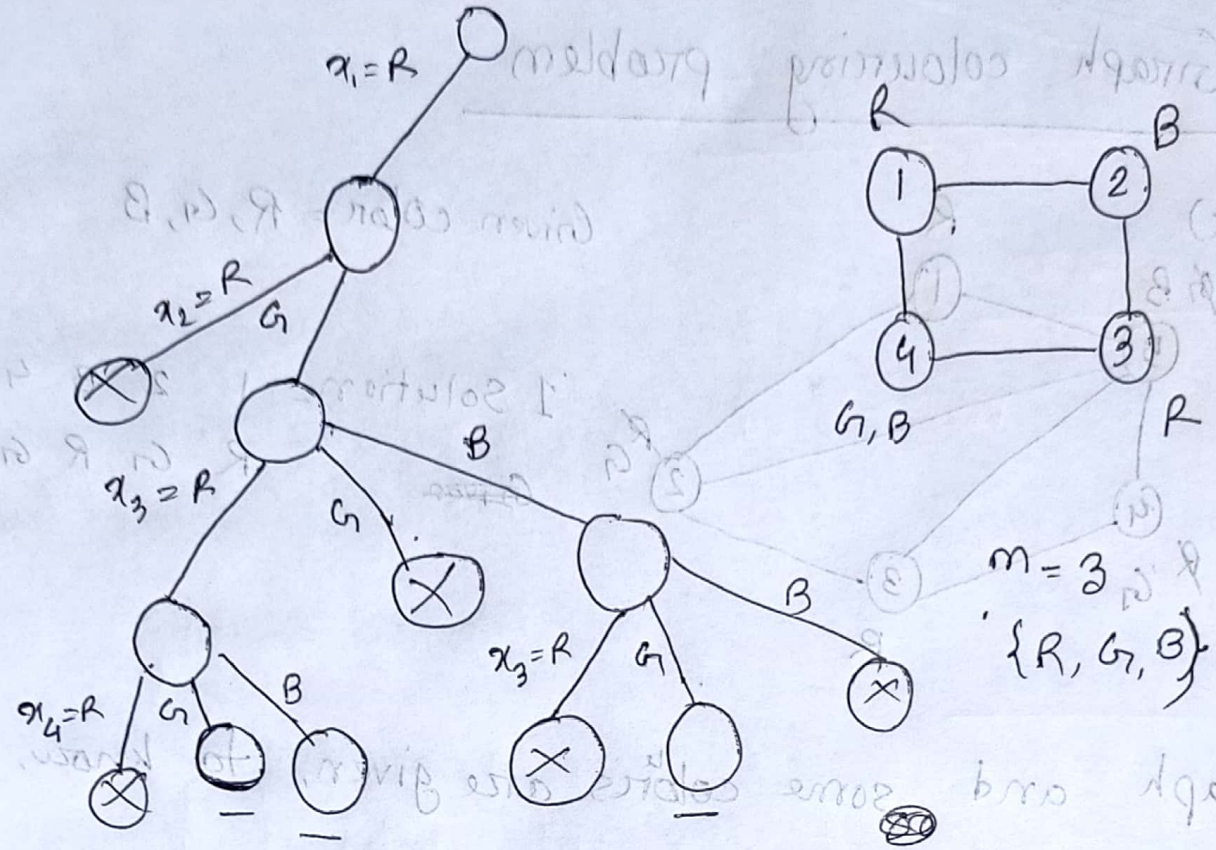
* If a graph and some colors are given, to know.

~~are~~ the graph can be ~~color~~ coloured or not.

**m-colouring decision problem**

* If graph and some colours are given, to know minimum how many colours are needed to colour the graph

**m colouring optimization problem**




Solutions:

i) R, G, R, G

ii) R, G, R, B

iii) R, G, B, G