introduction to Asymptotic notations.

problem → Program inc

↓

P1

↓ A1 ↓ A2 ↓ A3 ↓ A4

in $\theta(g(n))$

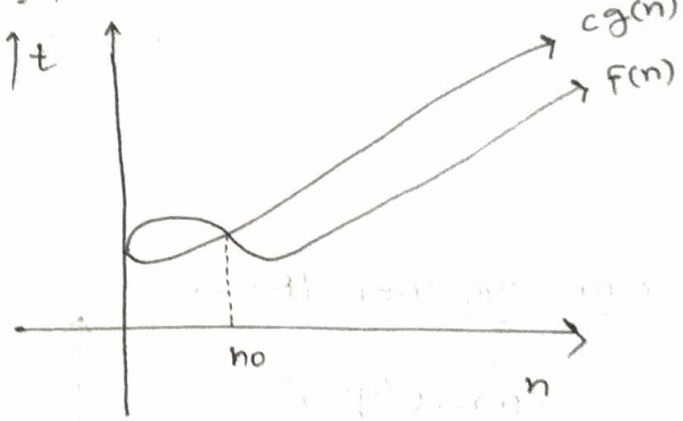
$$\boxed{0 \le c_1 g(n) \le f(n) \le c_2 g(n)}$$

Big(omega) Big(oh)

→ For solving any problem there are large no. of Algorithm we select that algorithm which is having less no. of time and less consuming memory.

Asymptotic Notation

(i) Big (oh) O → tight higher value



v. imp. $\boxed{f(n) \le cg(n)}$ ✓ $n \ge n_0$

$c > 0, n_0 \ge 1$

$$\boxed{f(n) = O(g(n))}$$

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

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

$f(n) \le cg(n)$

$3n+2 \le 4n$

$\boxed{n \ge 2}$

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

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

$g(n) = n^2$

n^3

n^n

Here always $f(n) \le cg(n)$

always f(n) is always greater but always less.

(ii) Big omega (Ω)

(tightly bounded)

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

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

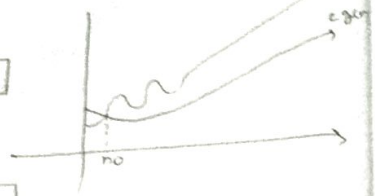
V. imp $\leftarrow$ $\boxed{f(n) \geq c\,g(n)}$

$3n+2 \geq cn$

$c = 1$

$n_0 \geq 1$

$\boxed{3n+2 = \Omega(n)}$



$f(n) = 3n+2 \quad g(n) = n^2$

$3n+2 \overset{?}{=} \Omega(n^2)$

$3n+2 \geq c(n^2)$, no

$f(n) = \Omega(n)$

$\downarrow$

$\log n$

$\downarrow$

$\log \log n$

(iii) Big theta (θ) $\Rightarrow$

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

$\boxed{c_1 g(n) \leq f(n) \leq c_2 g(n)}$

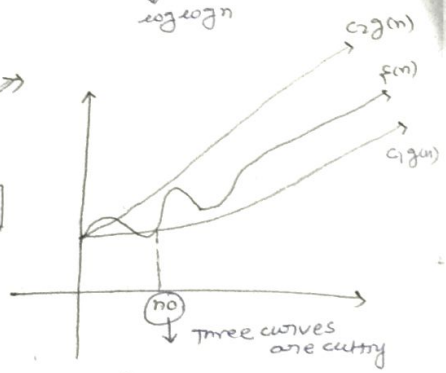
$c_1, c_2 > 0$

$n \geq n_0$

$\boxed{n_0 \geq 1}$

$\downarrow$

shows minimum one input



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

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

$3n+2 \leq 4n$, $n_0 \geq 1$

For any +ve con $c > 0$, $f(n) \geq c_1 g(n)$

$3n+2 \geq n$, $n_0 \geq 1$

(v) in $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

(iv) smaller (o)

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

$\downarrow$ not Asymtotically tight.

$o g(n) = \{ f(n) :$ there exist a constant $n > 0$ such that $0 \leq f(n) \leq c g(n)$ for all $n \geq n_0 \}$

#. physical significance of the Asymptotic Notation $\rightarrow$

O (worst case) $\quad \Omega$ (best) $\quad \theta$ (average case)

$\rightarrow$ we always interested into find out the worst case always

$\rightarrow$ when the worst case and best case are same we are interested into find out the average case

ex $\rightarrow$

5	7	1	2	4	10	12	9

searching for the $\boxed{5}$

in worst case time complexity is $O(n)$ $\leftarrow$

$\boxed{\begin{array}{l} \Omega(1) \\ O(n) \\ \theta(n/2) = \theta(n) \end{array}}$

$\Rightarrow$ Time complexity Analysis of Iterative programs

(Algo)

Iterative

```
A() {
  for i=n to 2n
    max(a,b)
}
```

[for this we check the no of time loop executed.]

Recursive

```
A(n){
  if ()
  A(n/2)
}
```

(same power Analysis different)

use Recursive equation for A(n) using A(n/2) then find time complexity

Note → if there is no iteration and Recursion (means program is not iterative and not Recursive) the time complexity is of $\boxed{O(1)}$. (constant)

Q → Find the time complexity of the following programs.

① A() {
```
int i, j;
for (i = 1 to n)
    for (j = 1 to n)
        printf(" x");
}
```
$\boxed{O(n^2)}$

Note – very imp

```
for (i=1, i≤n; i++) {pf(" ")}
for (i=1, i≤n; i++) {}
for (j=1; j≤n; j++) {
    pf(" ")}
}
```
time complex

$c_0 + c_1 n + c_2 n^2$

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

② A() {
```
i=1; s=1;
while (s<=n) {
    i++;
    s=s+i;
printf("ravi"); }}
```

s	1	3	6	10	15	21	- - -		n
i	1	2	3	4	5	6	- - -		K

let the loop is executed K times.

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

$$\frac{K^2}{2} + \frac{K}{2} = n$$

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

Hence time complexity $\boxed{O(\sqrt{n})}$

③ A() {
```
i=1;
for (i=1; i²≤n; i++)
    pf("ravi");
}
```
time complexity

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

④ A() {
```
int i, j, k, n;
for (i=1; i≤=n; i++) {
    for (j=1; j≤i; j++) {
        for (k=1; k<=100; k++)
        { pf("ravi"); }}}
```

$i=1$	$i=2$	$i=3$
$j=1$ times	$j=2$ times	$j=3$ times
k 100 times	$k=2*100$	$k=3*100$

$i=4$	$i=n$
$j=4$ times	$j=n$ times
k $4*100$ times	$k=n*100$ times

Hence total no of ravi printed

$100 + 2*100 + 3*100 + \cdots n*100$

$= 100(1+2+\cdots n)$

$= 100\left(\frac{n(n+1)}{2}\right)$

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

⑤ A() {
```
int i, j, k, n;
for (i=1; i<=n; i++) {
    for (j=1; j≤i²; j++) {
        for (k=1; k≤n/2; k++) {
            pf("ravi"); }}}
```

$i=1$	$i=2$	$i=3$	$i=n$
$j=1$ times	$j=4$ times	$j=9$ times	$j=n^2$
$k=\frac{n}{2}*1$	$k=4\times\frac{n}{2}$ times	$k=9\times\frac{n}{2}$ times	$k=n^2\times\frac{n}{2}$

total no of Ravi printed

$$\frac{n}{2} + 4\times\frac{n}{2} + 9\times\frac{n}{2} + \cdots n^2\times\frac{n}{2}$$

$$= \frac{n}{2}\left[1+4+\cdots n^2\right]$$

$$= \frac{n}{2}\left[\frac{n(n+1)(2n+1)}{6}\right] = \boxed{O(n^3)}$$

Q → A() {

for (i=1; i<n; $\boxed{i=i*2}$) → Here i is increment by 2 times

printf("ravi"); }

Hence → $\boxed{\log_2 n}$

i = 1, 2, - - - - n

$2^0, 2^1, 2^2 - - - - 2^k$

$2^k = n$

$\Rightarrow \boxed{k = \log n}$

Time complexity $\boxed{O(\log_2 n)}$ ✓

Q → A() {

int i, j, k;

n/2 ← for (i=n/2; i<=n; i++) {

n/2 ← for (j=1; j ≤ n/2; j++) {

$\log_2 n$ · for (k=1; k ≤ n; k=k+2)

printf("ravi"); }}

Here i loop is executed independently means not dependent on j, k

similiary j and k also do this.

Hence we calculate time complexity directly

$= \frac{n}{2} \times \frac{n}{2} \times \log_2 n = O(n^2 \log_2 n)$ ✓

Q → Assume $n \geq 2$

A() { while (n>1) {

n = n/2; }}

$\boxed{k = \log_2 n}$

$\boxed{O(\log_2 n)}$ ✓

Q → A() {

for (i=1; i ≤ n; i++) {

for (j=1; j ≤ n; j=j+i) {

pf("ravi");

}

i=1	i=2	i=3		i=n
j=1 to n	j=1 to n	j=1 to n	. . .	j=1 to n
n	n/2 times	n/3 time		n/n times

$n + \frac{n}{2} + \frac{n}{3} + - - - - 1$

$= n\left(1 + \frac{1}{2} + \frac{1}{3} + - - - \frac{1}{n}\right)$

$= n \log(n)$

$\boxed{\text{time complexity} = O(n \log n)}$

Q → function (int n) {

if (n==1) return;

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

for (int j=1; j<=n; j++) {

printf(" ");

break; }}}

time complexity $\boxed{O(n)}$ → due to break

Q → function (int n) {

for (int i=1; i<=n; i++) // n times

for (int j=1; j ≤ n; j=j*2)

printf("*"); }

this loop executes $\boxed{\log n}$ time from out algo

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

For watt governor

$$N^2 = \frac{895}{h}$$

For Porter governor

$$N^2 = \frac{895}{h}\left[1 + \frac{(M_g + M)}{2m_g}(1+K)\right]$$

For Proell governor

$$N^2 = \frac{895}{h}\,\frac{a}{e}\left[1 + \frac{(M_g + M)}{2m_g}(1+K)\right]$$

Time complexity for Recursive function

```
A(n) {
    if (n>1)
        return (A(n-1)) }
```

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

Solving of Recursive equation

(1) Back substitution method —

$T(n) = 1 + T(n-1)$ — (i)

$T(n-1) = 1 + T(n-2)$ — (ii)

$T(n-2) = 1 + T(n-3)$ — (iii)

putting (iii) in (ii) —

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

$= 2 + T(n-3)$ — (iv)

putting (iv) into (i) —

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

⋮

$$\boxed{T(n) = K + T(n-K)}$$

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

Base condition $n = 1$, $T(n) = 1$

Now we know that $T(1) = 1$

$n - K = 1$

$$\boxed{K = n-1}$$

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

$= n + 1$

Hence $\boxed{O(n)}$

Q→ $T(n) = n + T(n-1)$

Base condition $n > 1$

$\boxed{01}$, $n = 1$

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

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

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

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

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

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

according to base $T(1) = 1$

Hence $n - (k+1) = 1$

$n - k - 1 = 1$

$$\boxed{K = n-2}$$

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

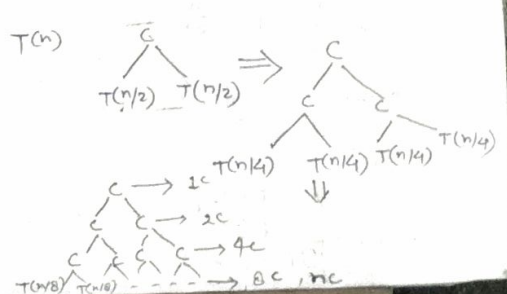
$= \frac{n(n+1)}{2} = O(n^2)$

(2) Recursion tree :→

$T(n) = 2T(n/2) + \boxed{c}$ $n \geq 1$

↓

means if we for 'n' input c can be Reduced Reduce into $T(n/2)$ → two parts



For 100% governor

$$N^\circ = \frac{895}{n}$$

For 80% governor

$$N^\circ = \frac{895}{n}\left[1 + \frac{(mg\,\mu)(1+k)}{2mg}\right]$$

For 120% governor

$$N^\circ = \frac{895}{n}\,\frac{a}{e}\left[1 + \frac{(mg\,\mu)(1+k)}{2mg}\right]$$

Hence

$$c + 2c + 4c + \cdots nc$$

$$= c(1+2+\cdots n) \qquad \text{let } n = 2^k$$

$$= c(1+2+4+\cdots 2^k)$$

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

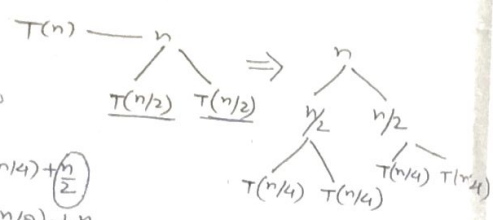
$$= c(2n-1)$$

Hence $\boxed{O(n)}$

(2)

$$T(n) = 2T(n/2) + n \quad , n > 1$$

$$= 1 \quad , n = 1$$



Here $T(n/2)$ new Function –

$$T(n/2) = 2T(n/4) + \frac{n}{2}$$

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

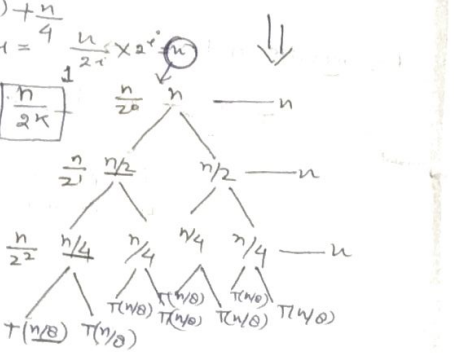
Total cost of each level $= \frac{n}{2^i} \times 2^i = n$

$$\frac{n}{2^0} - \frac{n}{2^1} - \frac{n}{2^2} - \frac{n}{2^3} - \cdots \boxed{\frac{n}{2^k}}$$

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

$$2^k = n$$

$$\boxed{k = \log_2 n}$$



total no. of levels $= (k+1)$

$$= (\log_2 n + 1)$$

$\boxed{\frac{n}{2^k}}\,T(1)$

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

→ comparision of Asymptotic functions.

(1) $2^n > n^2$

$n \log_2 2 \quad 2\log_2 n$

$n \quad 2\log n$

putting $n = 2^{100}$

$2^{100} > 200$

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

$\boxed{n > \log n}$

(3) $n \quad (\log n)^{100} \Rightarrow O(n)$

$\log n \quad 100 \log(\log n)$

putting $n = 2^{128}$

$128 \quad 100(\log(\log 2^{128}))$

$100 \log(128) = 700$

$128 \quad 700$

for $n = 2^{1024}$

$1024 > 1000$

Hence $n > (\log n)^{100}$

(4) $n^{\log n} > n \log n$

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

$n = 2^{1024}$

$1024 \times 1024 \quad (1024 + 10)$

> 1034

(5) $\sqrt{\log n} > \log\log n$

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

$n = 2^{210}$

$= \frac{1}{2} \times 10$

$\log 10$

$\boxed{= 3.5}$

(6) $n^{\sqrt{n}} > n^{\log n}$

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

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

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

$64 \quad 7 \quad n = 2^{128}$

Q. $f(n) = \begin{cases} n^3 & 0<n<10000 \\ n^2 & n \geq 10000 \end{cases}$

$g(n) = \begin{cases} n & 0<n<100 \\ n^3 & n>100 \end{cases}$

	0-99	100-9999	10,000 ...
$f(n)$	n^3	n^3	n^2
$g(n)$	n	n^3	n^3

Here at $n_0 = 10,000$

$g(n) > f(n)$

Hence $\boxed{f(n) = O(g(n))}$

Q→ $f_1 = 2^n$ $f_2 = n^{3/2}$ $f_3 = n\log n$ $f_4 = (n)^{\log n}$

taking $\log n$

$n\log(2)$ $\quad \frac{3}{2}\log(n)$ $\quad 4n + \log(\log n)$ $\quad \log n \log n$

put $n = 2^{1024}$

$2^{1024}(\log 2)$	$\frac{3}{2} \times 1024$	$1024 + 10$	1024×1024
	1536	1034	1024
$2^{128}\log(2)$	$\frac{3}{2} \times 128$	$128 + 7$	
	$\begin{array}{r} 64 \\ \times 3 \\ \hline 192 \end{array}$	135	128×128
$2^{128}\log(2)$			

$\boxed{f_1 > f_4 > f_2 > f_3}$ ✓

⇒ MASTER'S THEOREM (use to solve Recurrence Relation)

$T(n) = aT(n/b) + \theta(n^k \log^p n)$

$a \geq 1, b > 1, k \geq 0$ and p is real number

(i) if $a > b^k$, then $T(n) = \theta(n^{\log_b a})$

(ii) if $a = b^k$

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

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

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

(iii) if $a < b^k$

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

b) if $p < 0$, then $T(n) = O(n^k)$

Note:-

$\boxed{(\log n)^2 \quad (\log^2 n) \quad \log n \log n \quad \log\log n}$ ✓

Q→ $T(n) = 3T(n/2) + n^2$

$a = 3, b = 2, k = 2, p = 0$

$a \quad b^k$

$3 < (2)^2$ Here $a < b^k$

Here $p = 0$ 3(i) → condition

$T(n) = \theta(n^2 \log^0 n)$

$= \theta(n^2)$

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

$a = 4 \quad b = 2, k = 2, p = 0$

$a \quad b^k$ Here $p > -1$, i.e. $0 > -1$

$(4) \quad (2)^2$

$a = b^k$

$T(n) = \theta(n^{\log_2 4} \log n)$

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

Q→ $T(n) = 2T(n/2) + n\log n$

$a=2, b=2 \quad k=1, p=1$

$a \quad b^k$

$(2) \quad (2)$

Here case 2(i) –

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

$= \theta(n^{\log_2 2} \log^2 n)$

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

Q→ $T(n) = 6T(n/3) + n^2 \log n$

$a=6, b=3, k=2 \quad p=1$

$b^k > a$ Here also, $p \geq 0$

$\boxed{\theta(n^2 \log n)}$

Q→ $T(n) = 64T(n/8) - n^2 \log(n)$

↓ due to this not Applicable for M.T

Q→ $T(n) = 2T(\sqrt{n}) + \log n$

Sol→ let $n = 2^m$ $\quad T(n) = T(2^m)$

$\boxed{\log n = m}$

$T(n) = 2T(\sqrt{2^m}) + m$

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

To make it simple we assume,

$S(m) = T(2^m) \Rightarrow S(\frac{m}{2}) = T(2^{m/2}) \Rightarrow S(m) = 2S(\frac{m}{2}) + m$

↓ assume

$S(m) = 2S(m/2) + m$

⇓ Solving this

$S(m) = O(m \log m) = \boxed{O((\log n)\log\log n)}$

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

Analysing the space complexity of iterative and recursive algorithm

space complexity – total number of space required in the terms of the memory input (n) is called space complexity.

v.imp

For calculating the space complexity always look for the "EXTRA SPACE" requires by the program.

```
Algo(A,1,n){
  int i,J=10;
  for (i=1toJ)
      A[i]=0;
}
```

space complexity = $O(1)$

Because we use only extra two variable for the calculation.

```
Algo(A 1,n)
  { int i;
    create B[n];
    for(i=1ton)
    B[i]=A[i];
  }
```

Here $\boxed{n+1}$

$O(n)$

```
Algo(A,1,n){
Create B[n,n]
int i,J
for(i=1ton)
for(J=1ton)
B[i,J]=A[i]
}
```

(n^2+2)

Here space $\boxed{O(n^2)}$

Q→ Find the space and time complexity of the following program

```
A(n){ if(n≥1){
      A(n-1);
      pf(n);
      }}
```

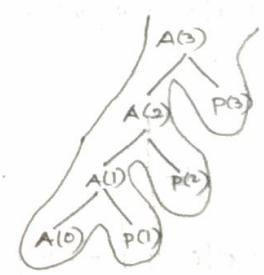
No. of function calls.

A(3) → 4

A(2) → 3

A(1) → 2

A(0) → 1



Hence $\boxed{A(n) \to (n+1)}$

time complexity $\boxed{O(n)}$

sorting Algorithm

(1) insertion sort :→

```
insertion-sort (A) {
    for j=2 to A.length
        key = A[j];
        //insert A[j] into sorted sequence
                A[1 - - - - j-1]
        i = j-1;
        while (i>0 and A[i] > key)
            A[i+1] = A[i]
            i = i-1
        A[i+1] = key }
```

Performance

1. worst case complexity – $\boxed{O(n^2)}$
2. Best case complexity – $O(n)$
3. Avarage case $O(n^2)$
4. worst case space complexity : $O(1)$ total $O(1)$ Auxiliary

9	6	5	0	8	2	7	1

we assume that playing cards in the left hand side of the man the sorted.

take 9 already sorted then 6 take it out if 6 is greater than 9 then place it Right side of

6	9	5	0	8	2	7	1

for 5 it out. compare with 9

5	6	9	0	8	2	7	1

0	5	6	9	8	2	7	1

0	5	6	9	2	7	1	

0	2	5	6	8	9	7	1

0	2	5	6	7	8	9	1

0	2	5	6	7	8	9	

Analysis :-

value of J	no. of comparison + no. of movement
2	1+1
3	2+2
4	3+3
n	$(n-1)+(n-1) = 2(n-1)$

total time complexity :-

$$2(1)+2(2)+2(3)+ - - - - 2(n-1)$$
$$= 2\left[1+2+3+ - - - (n-1)\right]$$
$$= 2\,\frac{n(n-1)}{2} = \boxed{O(n^2)}$$

(worst case)

Best case –

	1	2	3	4	5	6	7	8
no of comparision	1	1	1	1	1	1	1	
movement	0	0	0	0	0	0	0	

$$\boxed{\Omega(n-1) = \Omega(n)}$$

imp

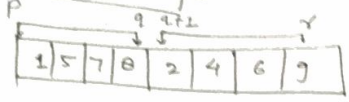
→ this is a type of Inplace sorting

$\boxed{\text{space complexity} = O(1)}$

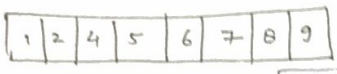
MERGE SORT

→ Better compare to insertion sort and sometimes better than quick sort.

Process of merging -

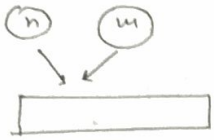


O(n)



O(n)
↓
time complexity

Note :→



(n+m)

O(n+m) time complexity

O(n+m) → space complexity

MERGE

```
for (k=p to r)
    if (L[i] ≤ R[j])
        A[k]=L[i]
        i=i+1
    else
        A[k]= R[j]
        j=j+1;
```

```
merge (A,p,q,r) {
    n1= q-p+1;
    n2=r-q;
    let L[1 - - - n1+1] and R[1 to n2+1] be new --
    for (i to n1)
        L[i]= A[p+i-1]
    for (j=1 to n2)
        R[j]= A[q+j]
    L[n1+1]=∞
    R[n2+1]=∞
    i=1, j=1
```

✓ program for merge sort -

MERGE-SORT

```
merge-sort (A,p,r) {
    if p<r
        q= [(p+r)/2]
    merge-sort (A,p,q)
    merge-sort (A,q+1,r)
    merge (A,p,q,r) }
```

if ms(1,6)

$q = \left[\frac{(1+6)}{2}\right] = 3$

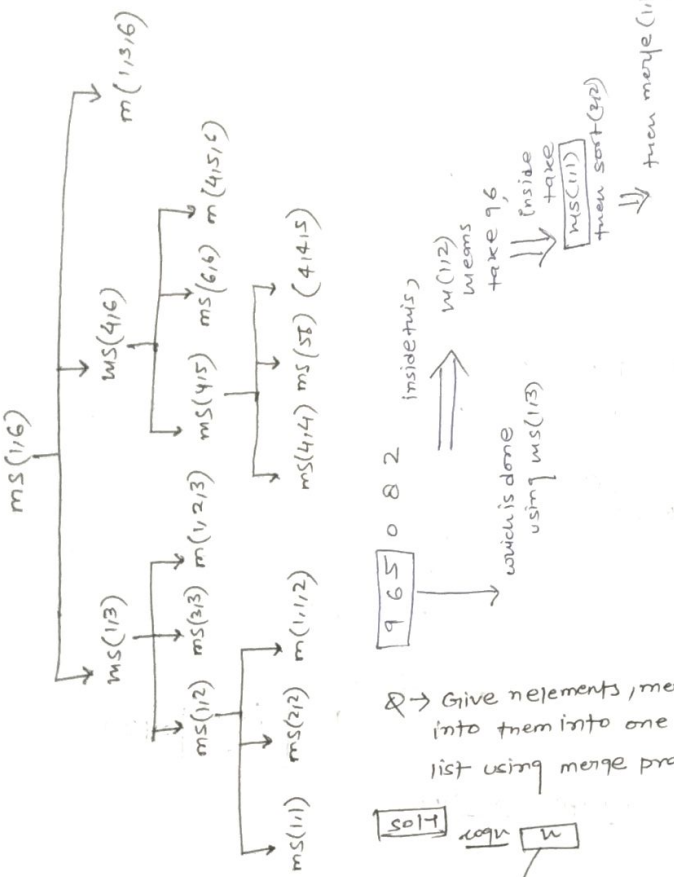
v. imp.

Note :→ outspace Algorithm :-

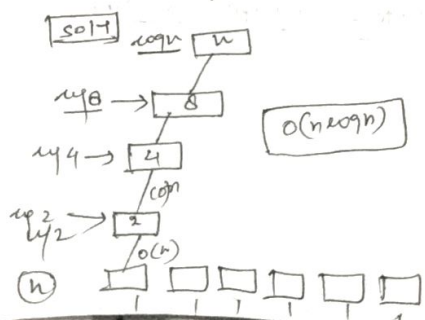
space complexity → merge: O(n)
stack — O(log n)

O(n+log n)

Hence O(n)



Q→ Give n elements, merge into them into one sorted list using merge procedure.



$O(n \log n)$

Time complexity of merge sort →

```
merge-sort (A,p,r) {
    if p<r
    q = ⌊(p+r)/2⌋
    merge-sort (A,p,q) — T(n/2)
    merge-sort (A,q+1,r) → T(n/2)
    merge (A,p,q,r) —— O(n)
}
```

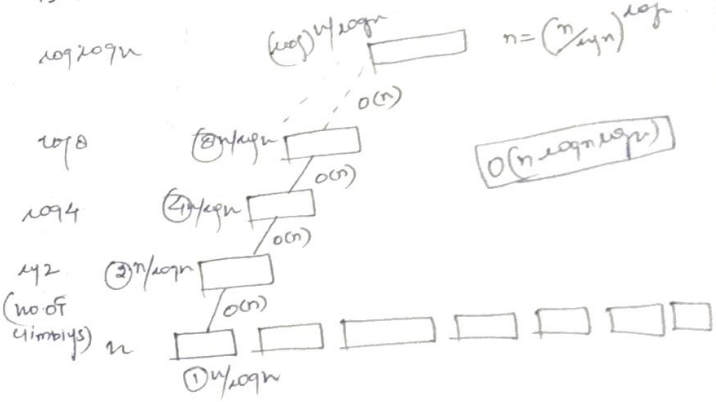
$T(n) = 2T(n/2) + n$

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

$a = b^k$

Hence Time complexity $\boxed{\Theta(n \log n)}$

Q→ Given "$\log n$" sorted lists each of size "$n/\log n$" what is the total time required to merge them into one



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

Note :-

1) merge sort uses Divide and conquer

2) 'm' and 'n'

$O(m+n)$

3) 20 47 15 8 9 4 40 30 12 17

what is order of merging after 2 pass ?

(2 way Merging)

20 47 15 8 9 4 40 30 12 17

(20 47) (8 15) (4 9) (30 40) (12 17)

(8 15 20 47) (4 9 30 40) (12 17)

Mergesort.

Performance :→

1. worst case complexity : $\Theta(n \log n)$
2. Best case complexity : $\Theta(n \log n)$
3. Average case complexity : $\Theta(n \log n)$
4. worst case space complexity : $\Theta(n)$ → Auxilary

—: Quick sort :—

```
PARTITAON(A,P,r) {
        x = A[r];
        i = p-1;
        for (J=p to r-1){
            if(A[J]≤x) {
                i=i+1;
                exchange A[i] with A[J]
            }}
        exchange A[i+1] with A[r]
        return i+1; }
```

select $\boxed{\text{pivot}}$

searching for element where this is less than (1) increment i and swap with J.

if a[J] > x then only J(↑)

Here we suppose A[r] as pivot, i starts with (p-1) and J starts with p.

$\boxed{x=11}$

13	19	9	5	12	8	7	4	21	2	6	11
1	2	3	4	5	6	7	8	9	10	11	12

0 ↑ i (P) ↑ J r↓

Here 13>11 Hence Hence J increment (∵ J is proceeding forward if it found the no. less than 11 place it in left most)

again J points to 19 Here also, 19>11 J incr

Now 11>9 Hence Here [increment i and swap with J]

Now (i+1) points to 13 and J points to 9.

swap 13 and 9.

$\boxed{O(n)}$

9	19	13	5	12	8	7	4	21	2	6	11

↑ i ↑ J

again 11>5 exchange 19 with 5

QUICKSORT (A, p, r) {

if (p<r) {

q = PARTITION (A, p, r) — O(n)

QUICKSORT (A, p, q-1) T(n/2)

QUICKSORT (A, q+1, r) T(n/2) T(n-1)

}}

Best case

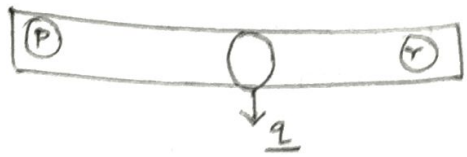
O(n)

when input is

2	2	2	2	2	2

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

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

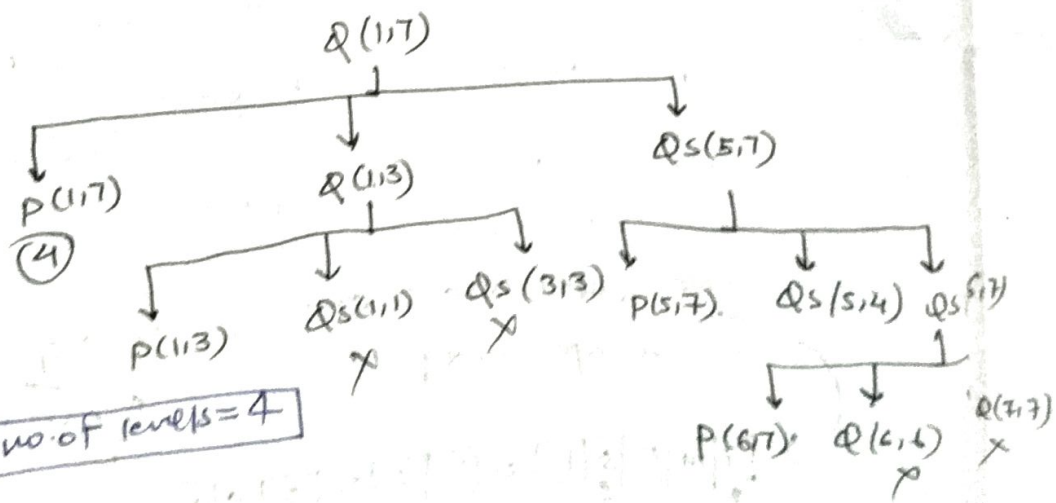


For analysis the space complexity what is extra space is used?

Here only q is used as extra variable.

the space complexity is devided by the size of stack used. the size of stack is devided by the "levels in the Recursive tree".

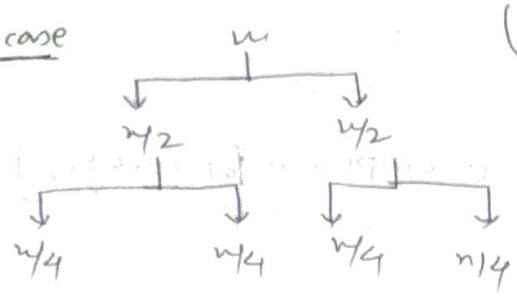
	1	2	3	4	5	6	7
A	5	7	6	1	3	2	4



$\boxed{\text{no. of levels} = 4}$

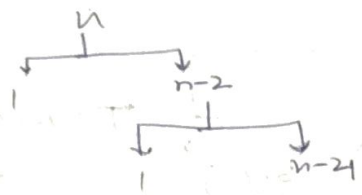
① intu balance (intu Best case)

$\boxed{\Theta(n \log n)}$



space complexity $O(c \log n) = O(\log n)$

② worst case –



performance
worst case complexity: $\boxed{O(n^2)}$
Best case complexity: $\boxed{O(n \log n)}$ avg

space complexity $\boxed{O(n)}$

time complexity

time complexity ⇒

① $T(n) = 2 * T(n/2) + O(n)$

$= \Theta(n \log n) \longrightarrow$ Best case

② worst case –

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

$= T(n-1) + cn$

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

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

$= c + c \cdot 2 + c \cdot 3 + \cdots cn$

$= \boxed{O(n^2)}$ ✓

V.imp

Note:- whenever the input is in Ascending or Desending or all inputs are equal then the time complexity $\boxed{O(n^2)}$

the Best case time complexity $\boxed{\Omega(n\log n)}$

① unsorte

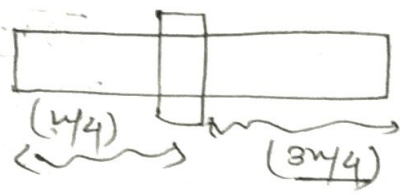
② sorted a

③ linked

min

from

V.imp:- in the quick sort, for sorting 'n' elements the $(n/4)$th smallest element is selected as Pivot using $O(n)$ time algorithm what is the worst case time complexity of quick sort?



$T(n) = O(n) + 1 + O(n) + T(n/4) + T(3n/4)$

if Any problem is break into 1:3 or 1:9 or 1:99 or 1:999 then always

Time complexity $\boxed{\Theta(n\log n)}$

Q→ time for sort $1, 2, 3 \cdots n \longrightarrow T_1$

$n, n-1, n-2, \cdots 1 \longrightarrow T_2$

what is Relation b/w T_1 and T_2.

$\boxed{T_1 = T_2}$ Ans.

Q→ $O(n)$ $\quad \frac{1}{5}(n) \mid \frac{4}{5}(n)$

$T(n) \leq O(n) + T(4n/5)$.

— : Heap sorts : —

	inserted	search	Find min	Delete min
① unsorted Array	$O(1)$	$O(n)$	$O(n)$	$O(n)$
② sorted array	$O(n)$	$O(\log n)$	$O(1)$	$O(n)$
③ linked list	$O(1)$	$O(n)$	$O(n)$	$O(n+1) = O(n)$
min Heap	$O(\log n)$		$(O(1))$	$O(\log n)$

from ①, ② and ③ the time complexity is of order of (i.e atleast) one is $O(n)$. in the case of insertion Find min, Delete min.

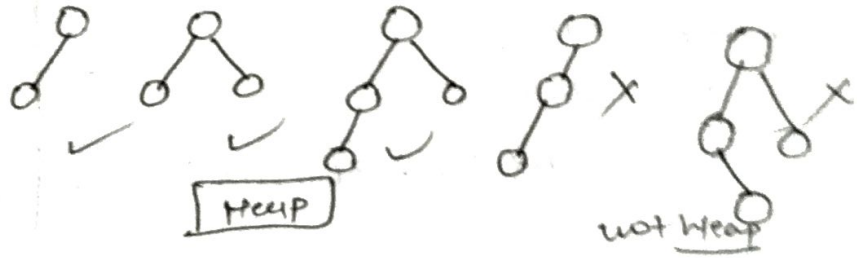
so decrese the search operation (like insertion, Find min, Delete min). we use the

$\boxed{\text{Heap}}$

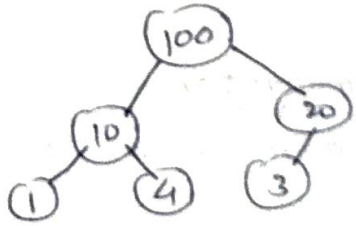
↓

is a binary tree or 3-ary.

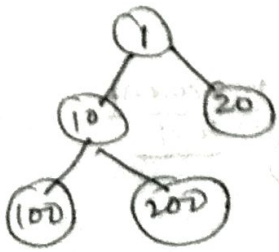
Heap is an almost complete



Max Heap →



min Heap



e axis, the bell-crank levers move on the
ases, the sleeve moves downwards. The
e. The spring force can be adjusted with

wo positions (assuming that the sleeve

(16.6)

(16.6a)

quity effects of the arms of the bell-

(i)

(ii)

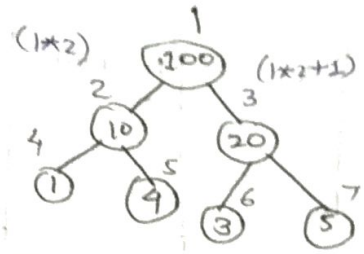
$-F_1)$

$F_1)$

(16.7)

s are 40 mm and 60 mm,
nass of each ball is 3 kg.
ssion and the constant of

max Heap



1	2	3	4	5	6	7
100	10	20	1	4	3	5

V.imp

if Root = i
index of left subtree = 2×i
index of Right subtree = (2i+1)

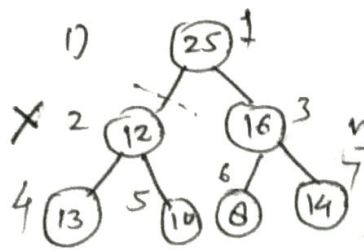
$\left\lceil i/2 \right\rceil = \left\lceil 5/2 \right\rceil = 2$ ✓

left

	A.length	A.heap
1) 25,12,16,13,10,8,14	7	1
2) 25,14,16,13,10,8,12	7	7
3) 25,14,13,16,10,8,12	7	1
4) 25,14,12,13,10,8,16	7	2
5) 14,13,12,10,8	5	5
6) 14,13,8,12,10	5	5
7) 14,13,12,8,10	5	5
8) 89,19,40,17,12,10,2,5,7,11,6,9,70	13	$\boxed{2}$ ✓

A.heapsize
↓
how many of them
following heap property?

1)



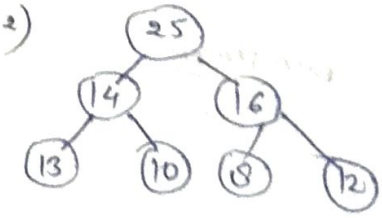
not count that

1

✓ Here 1 2 3
↓
(not satisfied)

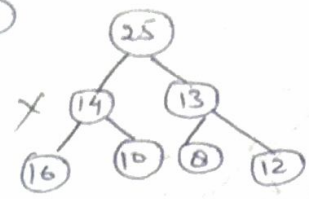
Here only $\boxed{1}$

2)



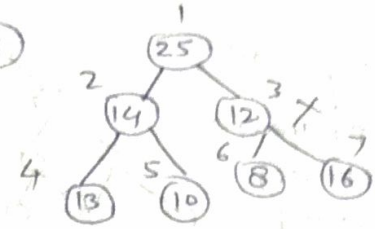
7

(3)



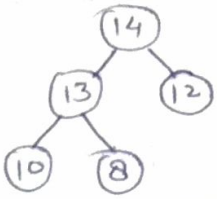
(1)

(4)



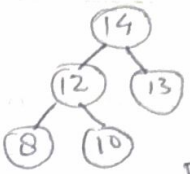
$\boxed{12}$ (2) nodes

(5)



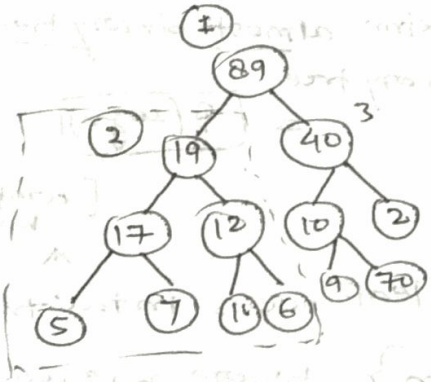
$\boxed{5}$

(6)



$\boxed{5}$

(7)



V. imp.

Note :→ if Array is in the desending order then turs is in

$\xrightarrow{14,13,12,10,8}$ (max Heap)

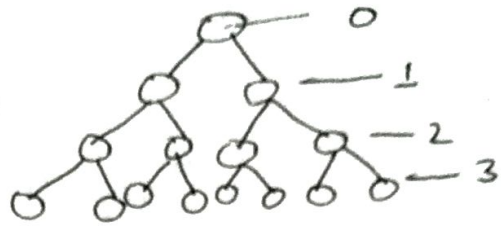
$\xrightarrow{8,10,12,13,14}$ (min Heap)

V. imp :- The time complexity of heapify is $\boxed{O(\log n)}$

Time complexity of create and Buildheap() is $\boxed{O(n)}$ and overall time complexity of heap sort is $\boxed{O(n \log n)}$

v.imp.

[Height of any node = no. of edges in the longest path]



$= (2^4 - 1) =$ [illegible]

$\boxed{\text{Height of Tree} = H(\text{ROOT})}$

Note :→ maximum no. of nodes in the complete binary tree $= \left[(2)^{h+1} - 1\right]$ $h \rightarrow$ height

The height of maxim. almost binary tree and complete binary tree $= \boxed{\Theta(\log n)}$

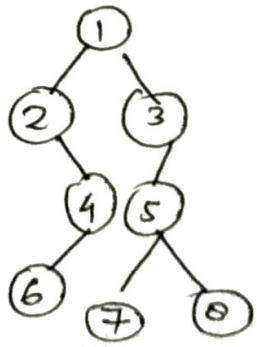
v.v.imp :

[complete binary]

⟹ The position of the leaf nodes in the tree is $\left\{ \left\lceil \frac{n}{2} \right\rceil + 1 \text{ to } n \right\}$ where n is no. of nodes.

$\left\lceil \frac{n}{2} \right\rceil + 1$ to

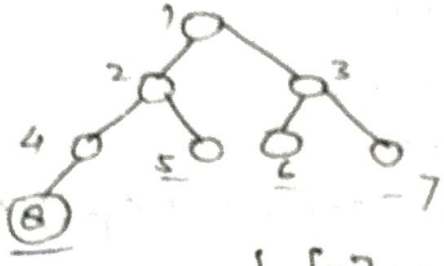
(location of the leaf nodes in the Heap tree)



$\{4+1 \text{ to } 8]$

(5 to 8)

∴ this is not complete binary tree Hence Not Follow



$\left\{ \left\lceil \frac{n}{2} \right\rceil + 1 \text{ to } n \right\}$

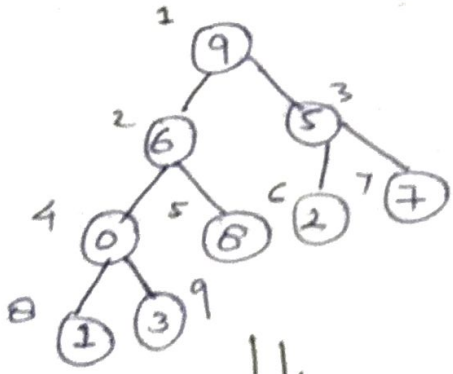
$\{4+1 \text{ to } 8\} = \{5 \text{ to } 8\}$

4 leaf nodes
5, 6, 7, 8

#.

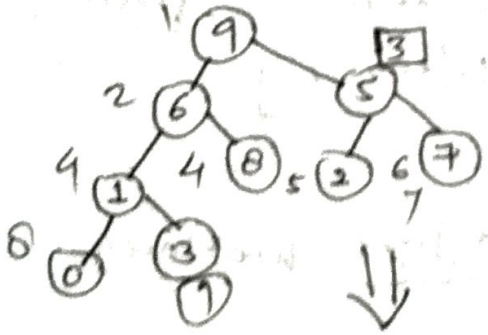
construction of the max Heap →

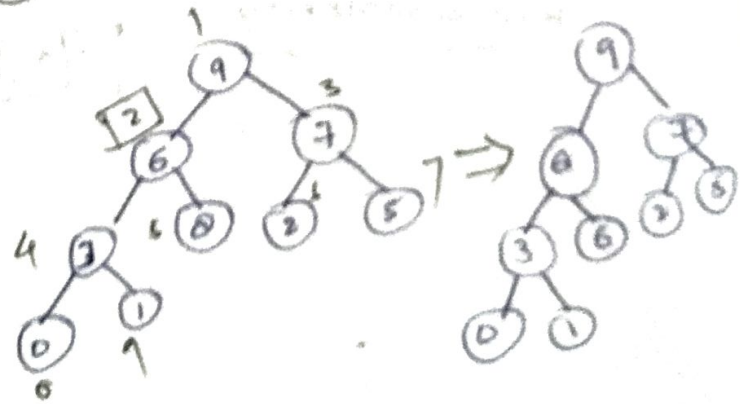
9 6 5 0 8 2 7 1



for the construction of max Heap.

① select max indexed node. ex → Here 4 (node) then 3 2 1





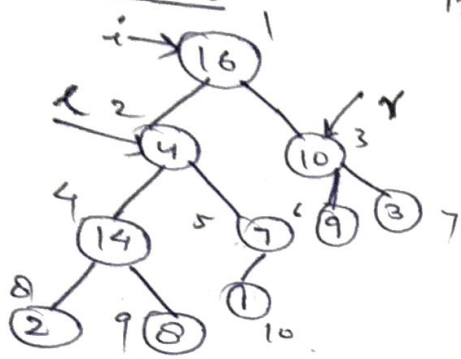
(पढ़ना है exam के लिए)

MAX-HEAPIFY (A, i) {

$l = 2i$;

$r = 2i+1$;

example



if ($l \le$ A.heap-size and A[l] > A[i])

presence of left

largest = l;

else

largest = i;

if ($r \le$ A.heap-size and A[r] > A[largest])

presence of Right

largest = r;

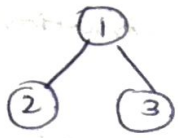
if (largest ≠ i)

exchange A[i] with A[largest]

MAX-HEAPIFY (A, largest)

}

note v.v.imp



→ From above Algorithm we compare 1 → 2 then 2 → 3 which one is larger place at top (not of both)

Time complexity → $\theta(2 \times \log n)$ $\boxed{O(\log n)}$ $\boxed{O(n)}$ not correct

space complexity — $\boxed{O(\log n)}$ no. of levels.

Build a MAX Heap AND Analysis

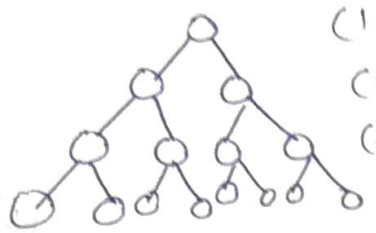
पढ़ना है।

```
BUILD-MAX-HEAP(A) {
    A.heap-size = A.length
    for (i = ⌈A.length/2⌉ down to 1)
        MAX-HEAPIFY(A, i) }
```

note

v.v.imp

in the complete binary tree →



(1,3)

(2,2)

(4,2)

(8,0) → height

↓

no. of nodes

complete binary tree.

$$\boxed{\text{No. of nodes at the height "h" in the CBT} = \left\lceil \frac{n}{2^{h+1}} \right\rceil}$$

max

Here example -

$$\left\lceil \frac{15}{2^{0+1}} \right\rceil = \left\lceil \frac{15}{2} \right\rceil = 8$$

$$\left\lceil \frac{15}{2^{1+1}} \right\rceil = \left\lceil \frac{15}{4} \right\rceil = 4 \checkmark$$

→ The time-complexity to build max Heap ⇒

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

(no. of nodes) at height "h"

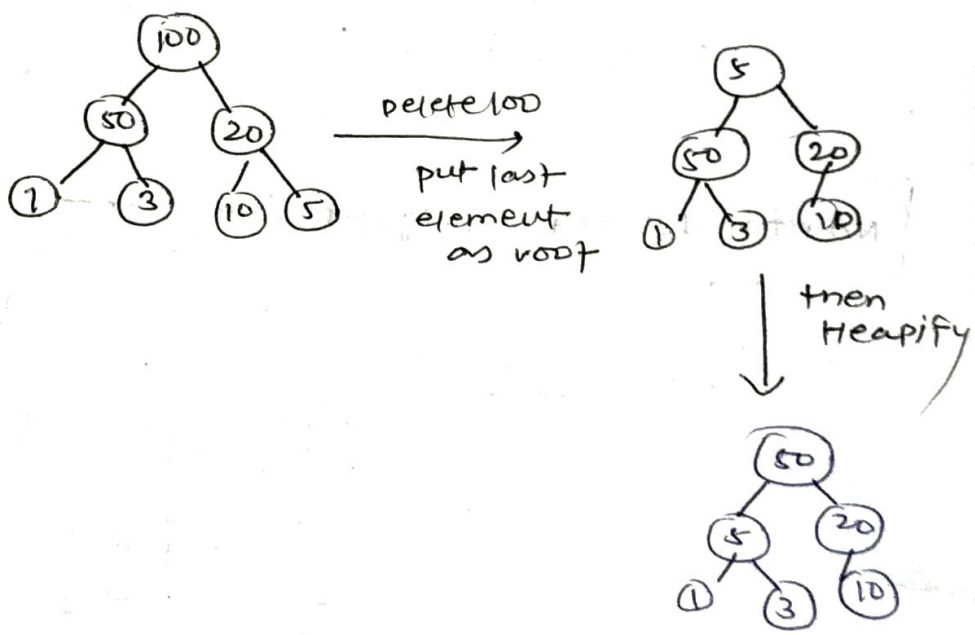
$$\frac{cn}{2} \sum_{h=0}^{\log n} \left(\frac{h}{2^h}\right) = O\left(\frac{cn}{2} \sum_{h=0}^{\log n} \frac{h}{2^h}\right)$$

$$= \boxed{O(n)} \checkmark$$

Extract max, increase key and insert key into Heap

```
Heap-Extract-max(A){
    if A.heap-size < 1
        error "heap underflow".
        max = A[1]
        A[1] = A[A.heap-size]
        A.heapsize = A.heap-size - 1
        MAX-HEAPIFY (A,1)
        return max }
```

100 50 20 1 3 10 5



Note :-

max heap

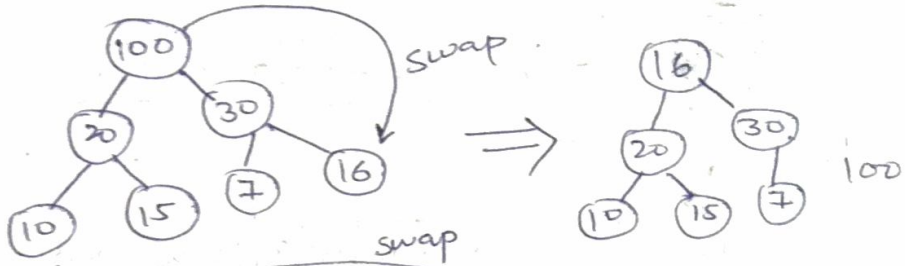
Find max	Delete max	insert	increase	decr key
$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

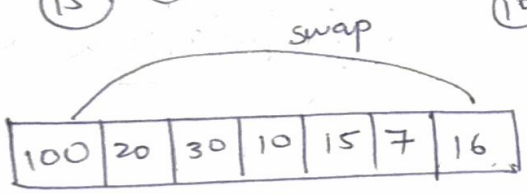
Find min	Delete	Search
$O(n)$	$O(n)$	$O(n)$

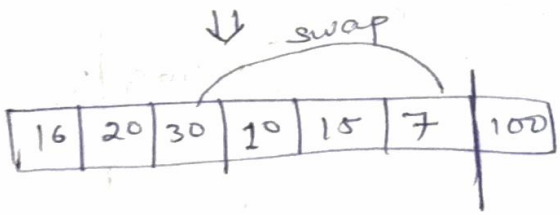
पहला है exam के लिए

Heapsort and Analysis

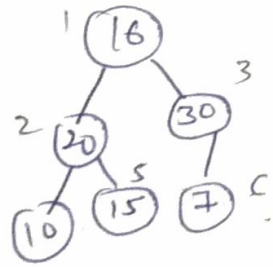
```
Heapsort (A){
    BUILD-MAX-HEAP (A)
        for (i = A.length downto 2)
            exchange A[1] with A[i]
            A.heap-size = A.heap-size - 1
            MAX-HEAPYFY (A,1) }
```



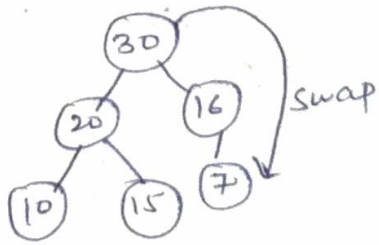


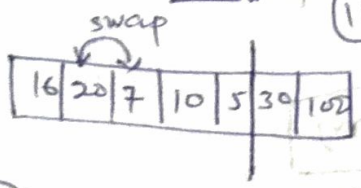


Now Heapify

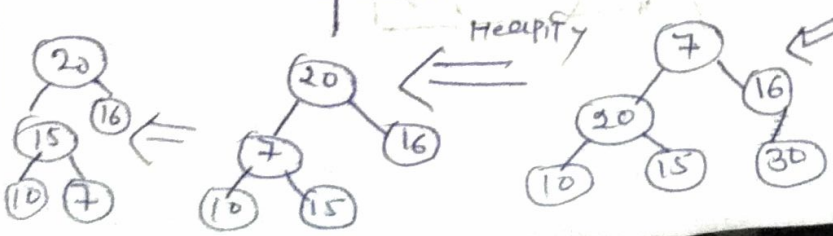


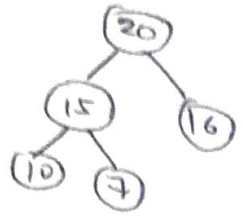
(last element in Heap)





Now Remove [30]

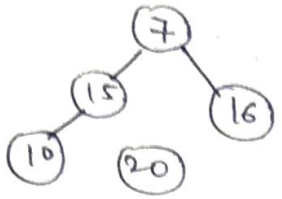




swap

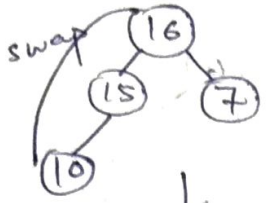
20	15	16	10	7	30	100

exchange (20) with (7)



7	15	16	10	20	30	100

Heapify

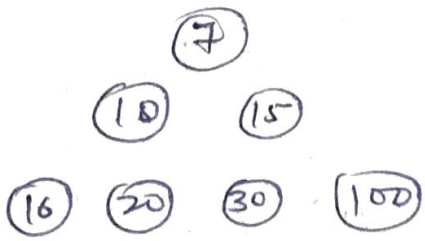


16	15	7	10	20	30	100

10	15	7	16	20	30	100

15	10	7	16	20	30	100

7	10	15	16	20	30	100



Note :→ Time complexity $\boxed{O(n \log n)}$

$x + mg(y_2 - y_1)$

$(y_2 - y_1)$

Q → in a heap with 'n' elements with the smallest element at the root, the 7th smallest element can be found in time

a) $O(n \log n)$ b) $O(n)$ c) $O(\log n)$ d) $\Theta(1)$

Sol: this is min Heap.

delete I min → $O(\log n)$

⋮

6th min → $O(\log n)$

Find 7th min → $O(1)$

insert → $6 * O(\log n)$

$= O(\log n)$

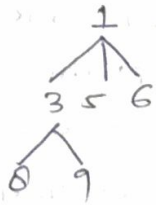
Q → which is of following the ternary max heap?

a) 1, 3, 5, 6, 8, 9

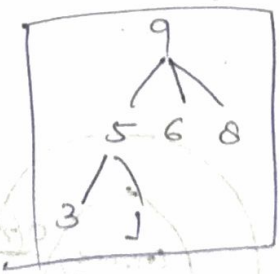
b) 9, 6, 3, 1, 8, 5

c) 9, 3, 6, 8, 5, 1

✓ d) 9, 5, 6, 8, 3, 1



Not max heap



max Heap

$y + mg\,(x_2 - x_1)$

$(x_2 - x_1)$

Q → in a heap with 'n elements" with the smallest element at the root, the 7th smallest element can be found in time

a) $O(n \log n)$ b) $O(n)$ c) $O(\log n)$ d) $\Theta(1)$

Sol: this is min Heap.

delete I min → $O(\log n)$

⋮

6th min → $O(\log n)$

Find 7th min → $O(1)$

insert → $6 * O(\log n)$

$= O(\log n)$

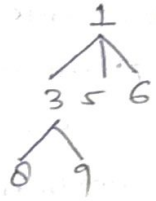
Q → which is of following the ternary max heap?

a) 1, 3, 5, 6, 8, 9

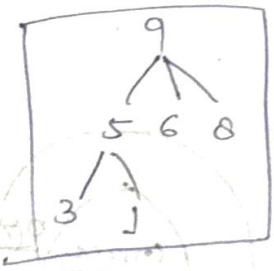
b) 9, 6, 3, 1, 8, 5

c) 9, 3, 6, 8, 5, 1

✓ d) 9, 5, 6, 8, 3, 1



Not max heap



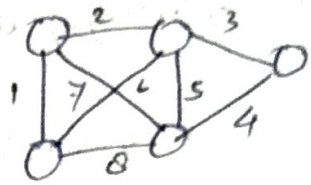
max Heap

(G.A is work is the stage. in each stage, a decision is made that is good at that point without somering about future)

Introduction to Greedy Algorithm

optimization

- min cost
- max profit
- max reliability
- min risk



Here to reach one vertext to another $O(2^n)$.

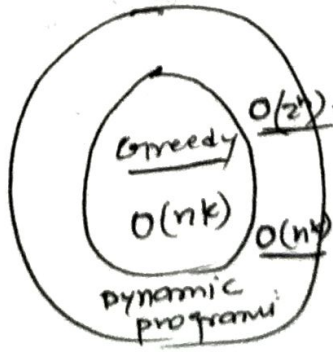
to solve the $O(2^n)$ problem [exshastive search]

Two Algoritms used ?

(1) Greedy Algoritm :- using Greedy we will not able save all optimization problem. there are some subset of optimization problem which is solve by Greedy

(2) Dynamic programming :-

Dynamic programming consolve all the optimization problem by DP. but which can use only those which having time complexity $\boxed{O(n^k)}$. we discard all which having time complexity (2^n).



examples of Greedy Algorithm :-

① Knapsack problem :-

(Bag)

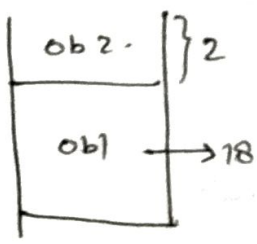
$n=3,\ m=20$

	ob1	ob2	ob3
profit	25	24	15
weight	18	15	10
P/W	$\frac{25}{18}=1.4$	$\frac{24}{15}=1.6$	$\frac{15}{10}=1.5$

Find the max profit which can be obtained in the maximum of 20 weight.

① Greedy about profit :-

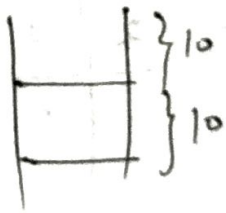
	W	P
ob1:	18	25
ob2:	2	$24\times\left(\frac{2}{15}\right)$
	20	28.2



20 units.

② Greedy about weights :-

ob3:	10	15
ob2:	10	$24\left(\frac{10}{15}\right)$
	20	$\boxed{31}$



③ Greedy about P/W

	W	P
ob-2	15	24
ob-3	5	$(15)\times\frac{5}{10}$
	20	$\boxed{31.5}$

```
Greedy Knapsack {
    for i=1 to n;
        compute pi/wi;
    Sort object in non increasing order of P/w //Dec
    for i=1 to n from sorted list
        if(m>0 and mi ≤ m)
            m = m - wi;
            P = P + Pi;
        else
            break;
    if (m>0)
        P = P + Pi (m/wi);  }
```

time complexity

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

$m=15, n=7$

objects	1	2	3	4	5	6	7
profit	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1
	5	1.6	3	1	6	4.5	3.

sort (5, 1, 6, 3, 7, 2, 4)

By (object)

w ~~15~~ ~~14~~ ~~12~~ ~~8~~ ~~3~~ 2 0 → not fill by 2 Hence take fraction

$P = 6 + 10 + 18 + 15 + 3 + 5(2/3)$

$= \boxed{55.3}$

fraction

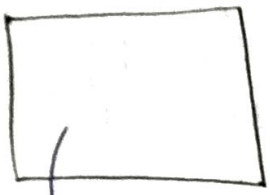
2
7
3
6
1
5

object Added

② HUFFMAN CODING

→ used to compression of the file.
→ compress the data.

abcd



file containing only abcd

+ Here are 4 charater abcd.
Hence 4 bit requies.

a – 00
b – 01
c – 10
d – 11

if 2 of for one either a/b/c/d

for 200 charcters (bit requires)

$= 200 \times 2 = 400$ bits.

$\boxed{\text{uniform encoding}}$

-80 bits compre

Non-uniform coding –

if a ocures 100 b → 80 c → 10 d → 10

bits representation

a	0	100
b	10	80
c	110	10
d	111	10

$\Rightarrow 100 \times (1) + 80 \times (2) + 10 \times (3) + 10 \times 3$

$= \boxed{320}$

How to identify

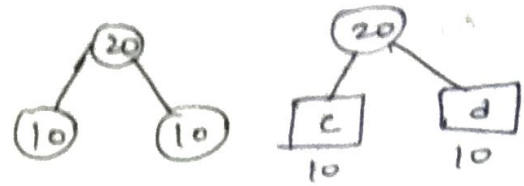
abcda

0 10110111 0

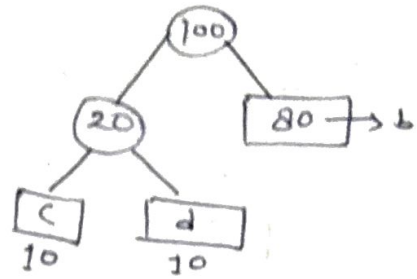
$\boxed{a \quad b \quad c \quad d \quad a}$

100 a, 80 b, 20 (10 c, 10 d)

1. take least 2 and merge them.

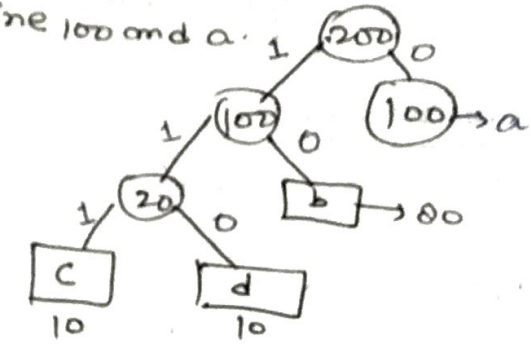


2. Now combined 20 and 80.



left (always)
↑
in 20, 80
↓
Right (al

3. combine 100 and a.



take either left side 0 or right side.

Hence to Reach

Coding

a →	0
b →	10
c →	111
d →	110

ex.

$100\times1 + 80\times2 + 80\times3 + 10\times3 + 10\times3$

$= 160 + 160$

$= 320$

$\frac{320}{400} = 1.6$ bit / Character

weighted external path length

$100\times1 + 80\times2 + 10\times3 + 10\times3$

```
HUFFMAN (c){
    n=|c|;
    make a min heap 'Q' with c

        for i=1 to n-1
        z.left = x = Extract-min (Q)
        z.right = y = Extract-min (Q)
        z.freq = x.freq + y.freq

            insert (Q,z)
            return (Extract-min (Q))
                                    }
```

Time complexity :-

$O(n \log n)$ where n is the number of unique character

if there are n nodes extramin() is called $2^{(n-1)}$ times.

extramin() takes $O(\log n)$ time as minHeapify().

if data is sorted

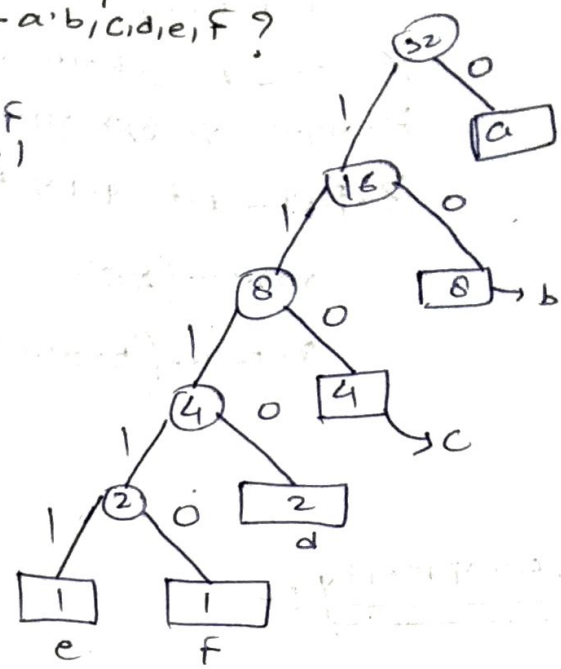
$\boxed{O(n)}$ time complexity

Q. suppose the letters a, b, c, d, e, f have probabilities 1/2, 1/4, 1/8, 1/16, 1/32, 1/32

1) which of following is the Huffman code for the letter a, b, c, d, e, f ?

a b c d e f
16, 8, 4, 2, 1, 1

a → 0
b → 10
c → 110
d → 1110
e → 11111
f → 11110



(2) av. length = $\frac{1+2+3+4+10}{6} = \frac{20}{6} = x$

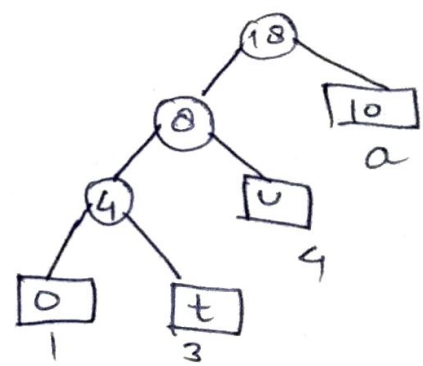
$\Downarrow$

$$\frac{16\times1+8\times2+4\times3+2\times4+1\times5+1\times5}{16+8+4+2+1}$$

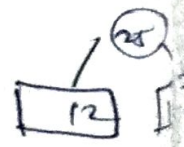
$= 1.9375$

v. imp

Q → a e i o u s t
10 15 12 3 4 13 1
4



not take 18 and 12 or 13 ✗



take always min 12, 13 ✓

③ Job sequencing with deadlines :→

time complexity (sort time) ⇒ $O(n \log n)$

worstcase ⇓ $\boxed{O(n^2)}$

Job	J1	J2	J3	J4
deadline :	2	1	1	2
profit :	6	8	5	10

max profit → 10 ↓ 8 ↓ 6

①

0	1	2
J4	J1 6	
10		

Here we can't place J2 because of deadline is 1

Hence complace $\boxed{6}$

$\boxed{total = 16}$

②

0	1	2
J2	J4	
8	10	

$\boxed{18}$ ✓

Always apply this

2nd way → place the heighest profitable always most right side.

like J4 → place at $\boxed{1 \to 2}$

Q →

	J1	J2	J3	J4	J5	J6
D:	5	3	3	2	4	2
P:	200	100	190	300	120	100

1	2	3	4	5
J4	J3			J1

→ always starts with 0

0	1	2	3	4	5
J2	J4	J3	J5	J1	

Jobs	J1	J2	J3	J4	J5
profits :	2	4	3	1	10
deadlines :	3	3	3	4	4

1. create an array for the maximum deadline.

0	1	2	3	4
	J1	J3	J2	J5

2. sort out the jobs.

J5, J2, J3, J1, J4

2 + 3 + 4 + 10 = 19

4 OPTIMAL MERGE PATTERNS :→

all things are same as Haffman coding

min Heap O(n)

2 minHeap O(logn)

Insert O(logn) → O(nlogn)

$x + mg$

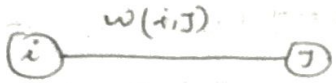
$(x_0 - x_1)$

Minimum spanning tree → [covering all vertices but does not having cycle within it]

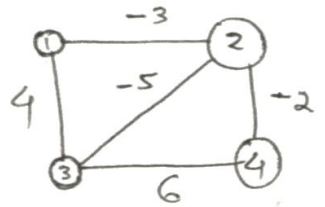
$G = (V, E)$, $w : E \to R$

↓ Undirected Graph

↳ weight Function

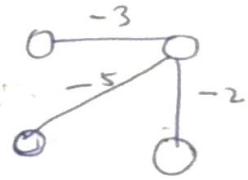


ex →



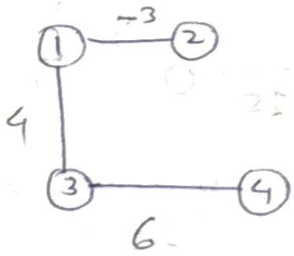
spanning tree are ⟹

(1)



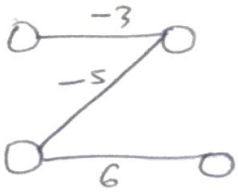
$w(T_1) = 10$

(2)



$w(T_2) = 7$

(3)



$\boxed{w(T_3) = -2}$

Hence $w(T_1)$ is minimum spanning tree.

MST problem :-

input: $\langle G = (V, E), w : E \to R \rangle$

output: A spanning tree T

$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

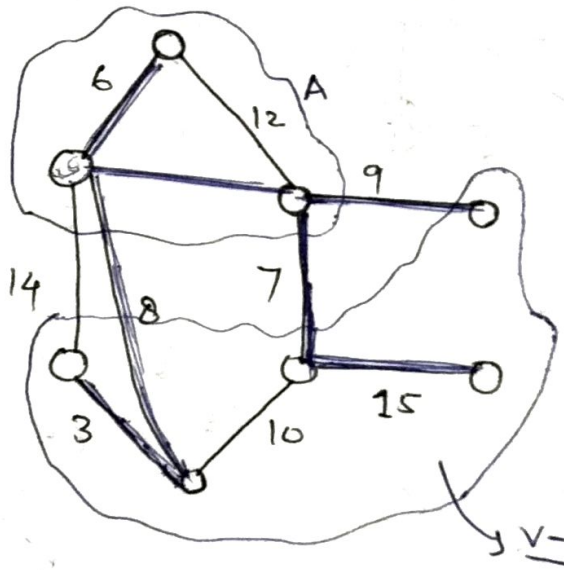
$$\min \{ T \mid \cdots \}$$

prims Algorithm :-

Hallmark for Greedy algorithm greed choice property

A locally optimal choice is globally optimal.

Theorem :- let T be a MST of $G=(V,E)$ and $A \subseteq V$ suppose $(V,U) \in E$ is the least weight edge connecting A and V−A. Then $(U,V) \in T$.



[Here edge with 7 which is minimum Among the all bridges which is connecting A and (V−A)]

proof :→

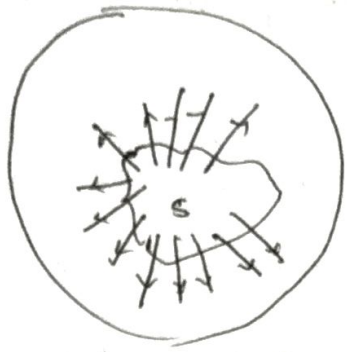
$A^C \rightarrow$ • (black)

$A \rightarrow$. (Blue)

Prim's Algorithm :-

$(G=(V,E), w: E \rightarrow R)$

idea – Maintain $(V-A)$ as a priority Queue Q. key each vertex in Q with the weight of the least-weight edge connecting it to vertex in A.



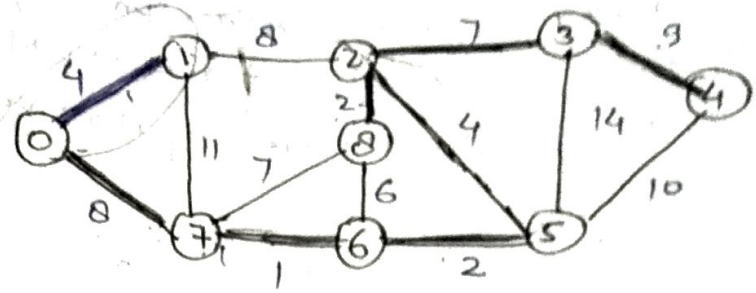
$w=\{s\}$ connect to the that bridge which having minimum weight.

psuedo code :-

```
Q ← V
key(v) ← ∞        ∀ v ∈ V   //all at ∞
key(s) ← 0        for some arbitary s ∈ V   // make s=0
while (Q ≠ 0)
        do u ← Extract.min(Q)
            for v ← adj(u)
                do if v ∈ Q and w(u,v) < key(v)   ↑ ∞ intial
                        then key(v) ← w(u,v)
                             π[v] ← u
{v, π(v)} MST
```

example -

if $w(u,v) < v.key$
$v.key = w$



vertex	0	1	2	3	4	5	6	7	8
key	0	∞	∞	∞	∞	∞	∞	∞	∞

Here 0 vertex with key 0 Hence

mstset = { }

take 0 mstset = {0}

[v → adjacent keys]

u → (source selected)

1 and 7 are adjacent to 0

Here

→ Here key(1) > w(0,1)
Hence update

$\boxed{key = 4}$

→ key(7) > w(0,8)

$\boxed{key(7) = 8}$

Now

vertex	0	1	2	3	4	5	6	7	8
key	0	$\boxed{4}$	∞	∞	∞	∞	∞	$\boxed{8}$	∞

Now take 1 which is having minimum ke
4.

mstset = {0,2}

Adjacent of (1) are (2) and (7)

Here key(2) > w(1,2)

Hence key(2) = 8

key(7) < w(1,7)

Remains same.

Now

Vertex	0	1	2	3	4	5	6	7	8
key	0	4	8	∞	∞	∞	∞	8	∞

Here 2, 7 having same minimum, take any one either 2 or 7.

Time-complexity :-

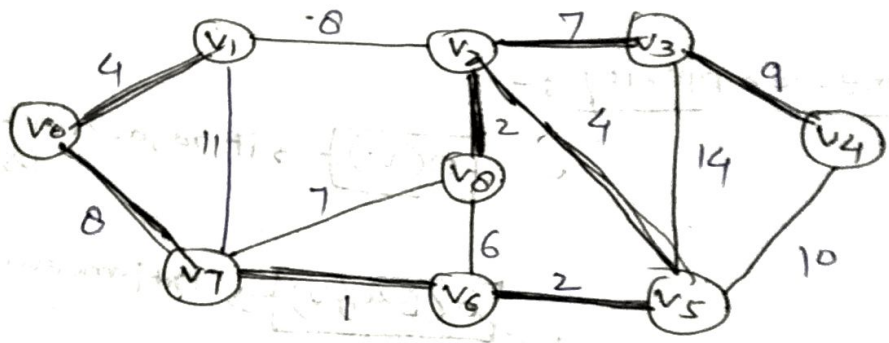
$\boxed{O(V^2)}$ → if adjancency matrix is used.

$\boxed{O(E \log V)}$ → adjancency list

Kruskal's Algorithm for MST :-

STEPS :-

1. sort all the edges in non-decreasing order of their weight
2. Pick the smallest edge. check if it forms a cycle with the spanning tree formed so far. if the cycle is not formed, include this edge. Else, discard it.
3. Repeat step 2 until there are (V-1) edges in the spannings.



arrange the edges in the Ascending order of weight

Edge	(V_7, V_6) ✓	$(V_8\ V_2)$ ✓	(V_6, V_5) ✓	(V_0, V_1) ✓	(V_2, V_5) ✓	(V_8, V_6) ✗	(V_2, V_3) ✓	(V_7, V_6) ✗
wt.	1	2	2	4	4	6	7	7

	(V_0, V_7) ✓	(V_1, V_2) ✗	(V_3, V_4) ✓	(V_5, V_4)	(V_1, V_7)	(V_3, V_5)
	8	8	9	10	11	14

stop at Edges in MST = $|V-1|$

pseudocode -

MST-KRUSKAL(G, w)

1. $A = \phi$ → $\boxed{O(1)}$
2. for each vertex $v \in G.V$
3. MAKE-SET(v)
4. sort the edges of $G.E$ into nondecreasing order by weight w — $\boxed{O(E \log E)}$
5. for each edge $(u,v) \in G.E$, taken in non-dec order by weight
6. IF FIND-SET(u) $\neq$ FIND-SET(v)
7. $A = A \cup \{(u,v)\}$
8. UNION(u,v)
9. Return A.

$O(E \log V)$

overall complexity $= O(E \log E + E \log V)$

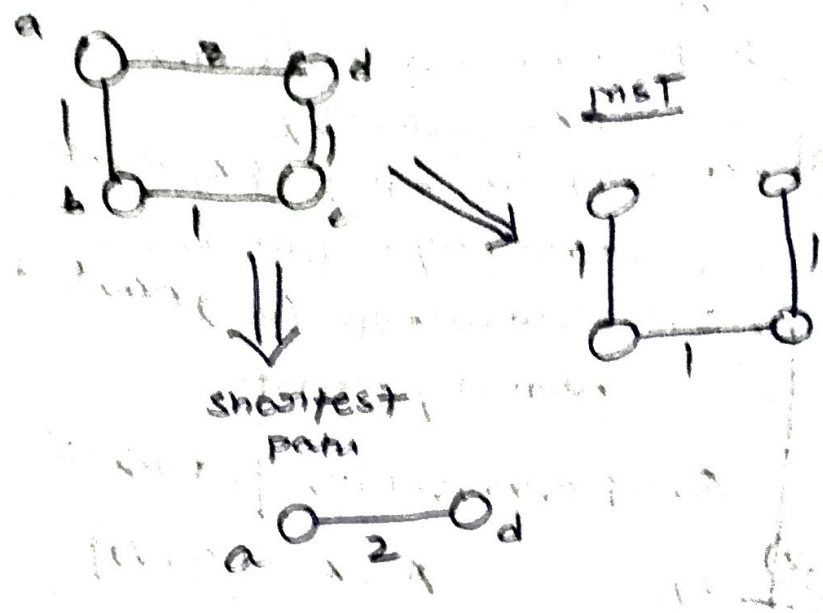
value of E can be almost $O(V^2)$, so $O(\log V)$ or $O(\log E)$

Hence complexity

↓

$\boxed{O(E \log E) \text{ or } \boxed{O(E \log V)}}$ ✓

V. Imp. Difference b/w <u>shortest path</u> and <u>MST</u>

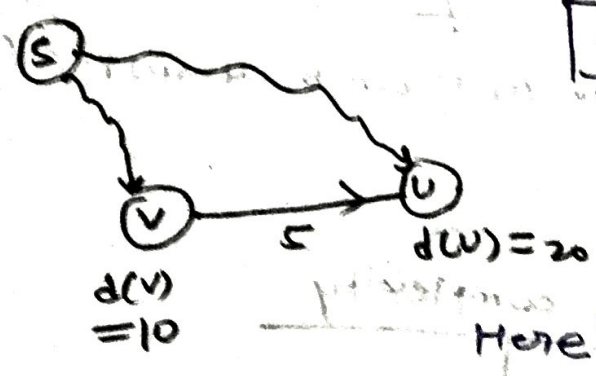


Relaxing an edge :-

let edge (v, u)

if $d(u) > d(v) + c(v,u)$

$$\boxed{d(u) = d(v) + c(v,u)}$$



Here $d(v) + c(u,v)$
$= 10 + 5 = \boxed{15}$

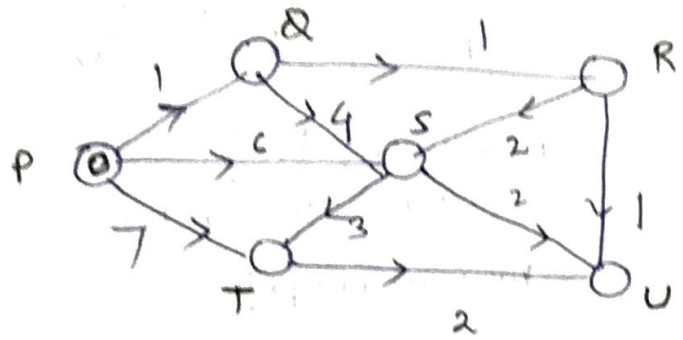
single source shortest path Algorithm

I. DIJKSTRA ALGORITHM :→

Time complexity – $\boxed{O(E \log V)}$

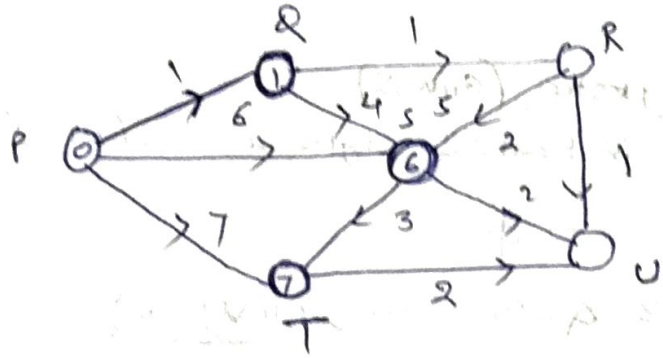
intu directed graph relax all the outgoing edges.

Q.



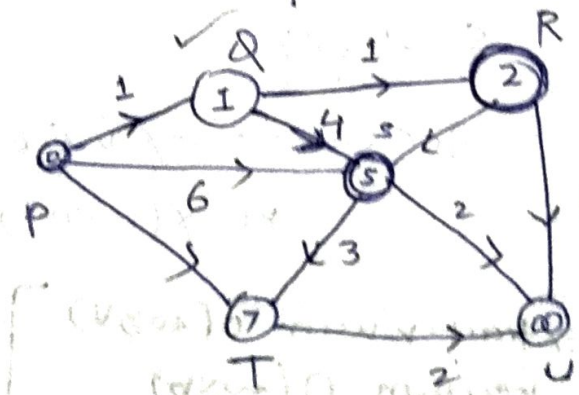
if P is source what is order of shortest path?

Relax adjacent vertex of P



Now pick the vertex with least weight

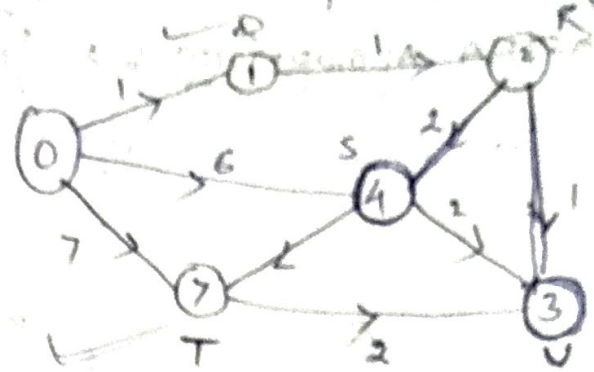
$P \Rightarrow 1$



intu Remainy one R, U, T, S

the minimum is $\boxed{R}$

Now Relax the outgoing edges from the R



Next minimum is $\boxed{V}$

which is not having outgoing

Here go for $\boxed{4}$.

$\boxed{Q\ R\ U\ S\ T}$

Pseudocode of Dijkstra Algorithm —

Dijkstra (G, w, s)

1. initialize-sigle-source (G, s) → O(V)

2. $S = \phi$
3. $Q = G.V$ → $O(V \log V)$
4. while $\Rightarrow \phi \neq \phi$ $(Q \neq \phi)$

   $u = \text{Extract-min}(Q)$

   $S = S \cup \{u\}$

   // Relax

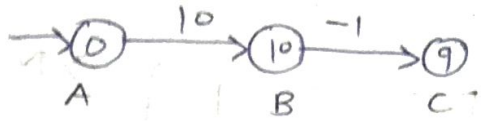
   for each vertex $v \in G.Adj[u]$

   Relax (u, v, w)

[Extract-min $O(\log V)$
key min $O(\log V)$]

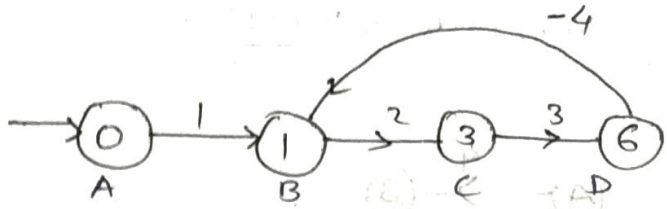
dijkstra Algorithm

a) Applicable in the case of the Native edge.
↓ Having (negative weight)



A	B	C
0	10	9

b) dijkstra algorithm is not applicable in the case of native edge loop. (Native weight cycle)



A	B	C	D
0	∞	∞	∞
	1	∞	∞
		3	∞
			6

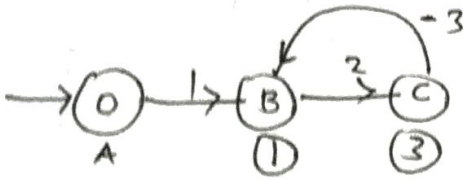
Now for the D outgoing is to B
outgoing is B.

(6−4) = $\boxed{2}$ (B) ✓ No problem here

↓ Not update bcz B is deleted.

because minimum is $\boxed{1}$

$\boxed{Ex}$ where dijkstra algorithms fails

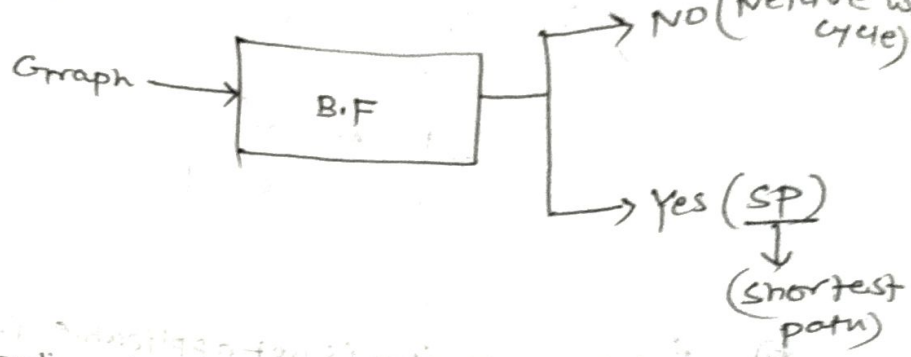


A	B	C
0	∞	∞
	1	∞
		3

$\boxed{B}$ → can't update from the tuer $\boxed{3-3}$ (0) $\boxed{C}$ not update

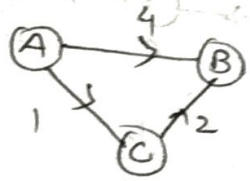
Bellman ford Algorithm $O(VE)$

(slower compare to Dijk...)

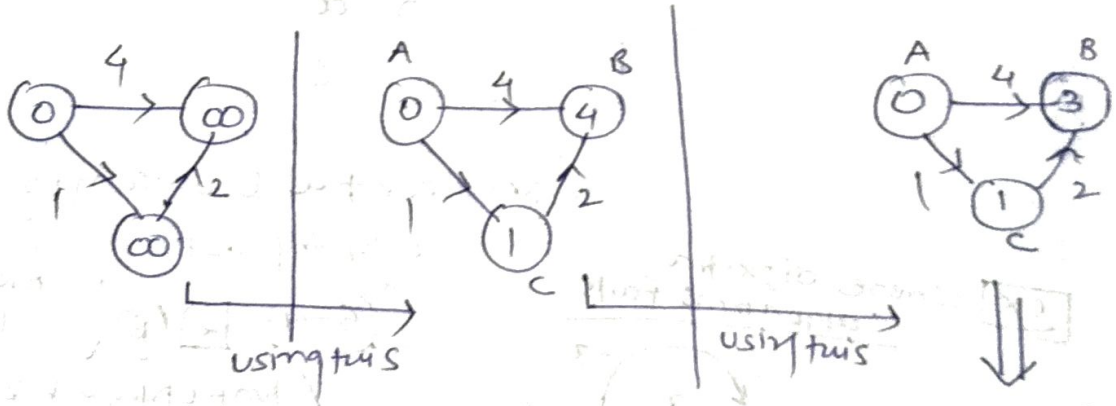


According to Bellman ford Algoritm :—

(Graph is done "n−1" Relaxations.)

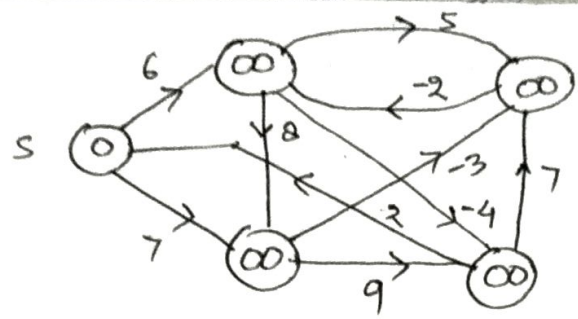


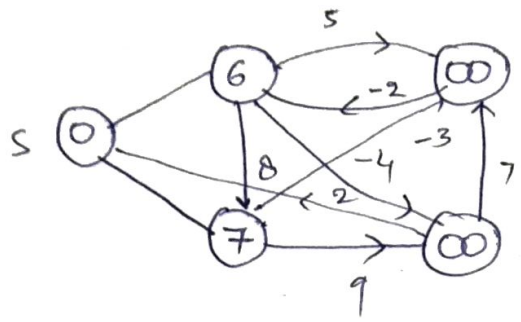
Here we done $\boxed{2}$ times of Relaxations.



if Relaxing (n−1) times, do one more times Relaxation. and look if value change then shows presence of the $\boxed{\text{Native edge cycle}}$.

ex →





Do this 4 times.

Topological order // → from L → R.

// Analysis of Bellman ford and o →

```
BELLMAN-FORD (G,w,s) {
        initialize single source (G,s) // O(V)
        For i=1 to |G.V|-1          O(V)
            foreach edge (u,v) ∈ G.E  (E)
                RELAX (u,v,w)
        for each edge (u,v) ∈ G.E → O(E)
            if (v.d > u.d + w(u,v))
                return FALSE

        return TRUE ; }
```

(v-1)(E) O(1)
= O(VE)

Dynamic programming

use of tables

whether a call a function or not.

Dynamic programming and memoization work together

the main difference b/w dynamic programming and conquer is that in the case of latter, sub problem are independent, whereas in DP there can be an overlap of sub problems.

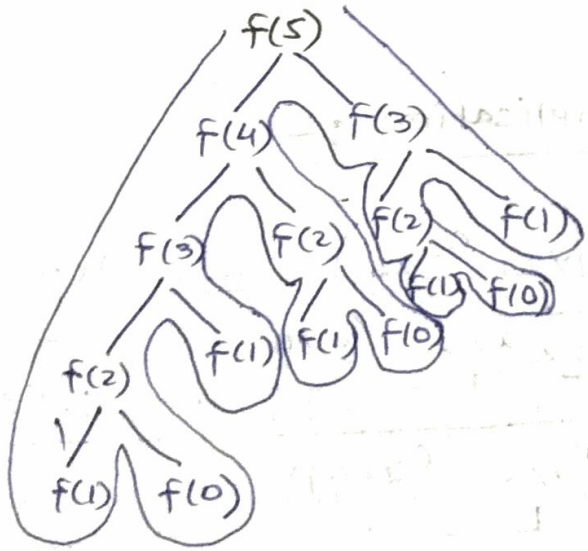
By using memoization [maintaining table of sub problem already solved], dynamic programming reduced the exponantial complexity of the polynom complexity $O(n^2)$, $O(n^3)$ etc.

$$\boxed{\text{Dynamic programming} = \text{Recursion} + \text{Memoization}}$$

ex → $f(n) = f(n-1) + f(n-2)$; otherwise

$= 1$, $n==1$

$= 0$, $n==0$



Here if we starts from f(5) call f(4), f(3), f(2), f(1), f(0) once, for again call of f(2), f(3), f(4) f(5) use the table

2	3	4	5

enstead of calling f(2) f(3) f(4) hence

$O(2^n) \rightarrow \boxed{O(n)}$

check the table
for T(n)
if T(n) is empty
f(n)
if T(n) is empty
T(n) = f(n)

1. matrix multiplication :-

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}_{3\times 2} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}_{2\times 3}$$

[no of scaler multiplication =

$3\times 2\times 3 = 18$

Note :→ $[\ ]_{(p\times q)} [\ ]_{(q\times r)}$

V.imp

No. of scaler multiplication =

$\boxed{(p\times q\times r)}$

2. chain multiplication :-

$((A_{2\times 1}\ B_{1\times 2})\ C_{2\times 4})$ ⟹ total multiplication req.

$2\times 1\times 2$ (4) $\quad 2\times 2 \quad 2\times 4$

$2\times 2\times 4 = 16$

= 4+16 = 20

E (2×4)

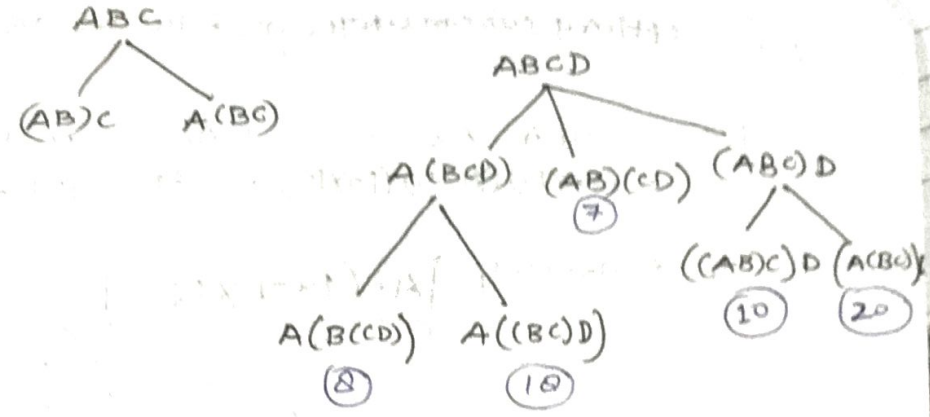
$(A_{2\times 1}\ (B_{1\times 2}\ C_{2\times 4}))$

$1\times 2\times 4 = 8$ ⇒ total = 16 multiplica

$2\times 1 \quad 1\times 4$

$2\times 1\times 4 = 8$

$\boxed{E\ (2\times 4)}$



Note: → We can reduce the no. of scaler multiplication by using the appropiate parentnesis..

if $A_{1\times2}$ $B_{2\times1}$ $C_{1\times4}$ $D_{4\times1}$

find all ways to multiply ABCD.

A(BCD) → 10

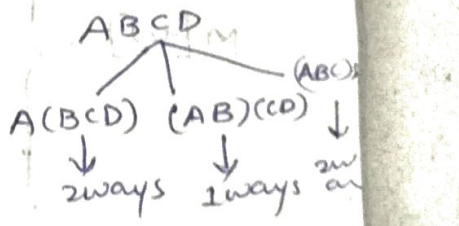
(AB)(CD) →

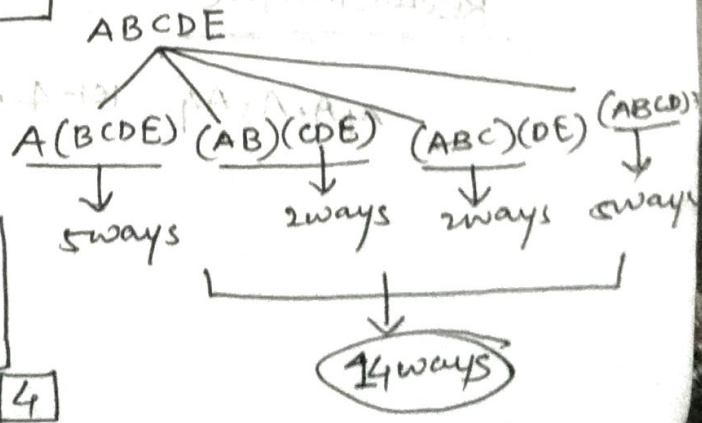
No. of ways of multiplication

no. of matrix

v. imp

3	2 ways
4	5 ways
5	14 ways





formula

$$\text{no. of ways} = \frac{(2n)!}{(n+1)!\,n!}$$

if no. of matrices = 4

put n = 3

optimal substructure and Recursive equation:

let

$A_1 A_2 A_3 \ldots A_n$ matrices.

$(P_0 \times P_1)(P_1 \times P_2)(P_2 \times P_3) \ldots (P_{n-1} \times P_n) \rightarrow$ Dimensio

in general $\boxed{A_i (p_{i-1} \times p_i)}$

$$A_i A_{i+1} A_{i+2} \ldots A_j = \underbrace{(A_i A_{i+1} \ldots A_k)}_{\boxed{p_{i-1} \times p_k}} \underbrace{(A_{k+1} \ldots A_j)}_{\boxed{(p_k \times p_j)}} + \underbrace{p_{i-1} p_k p_j}_{\text{two matrix multipli}}$$

result of $A_i \ldots A_k$

Here $i \leq k \leq j$

$[k = (i \ldots j-1)]$

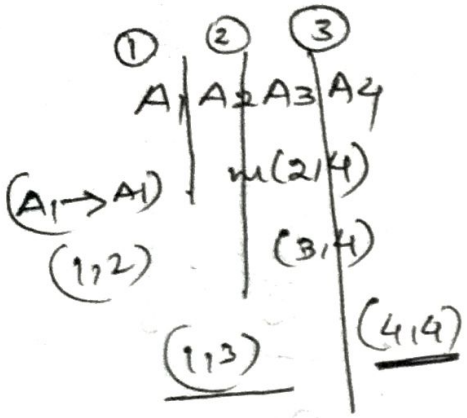
splitting can be done from anywhere from $A_i A_{i+1} \ldots A_j$.

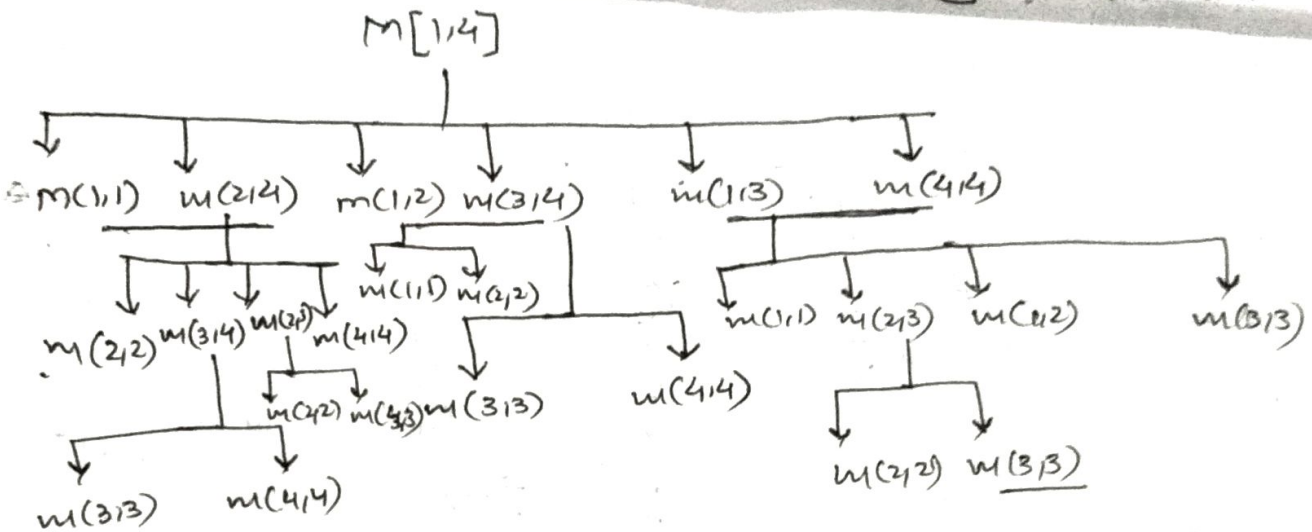
Recursive solution :→

$$M[i,j] = \begin{cases} 0, & i = j; \\ \min\{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\}, & i \leq k \leq j \end{cases}$$

Recursion tree

$A_1 A_2 A_3 A_4$ let 4 matrices $M[1,4]$





Here, we use table for the Already calculated value. Instead of calculating it again.

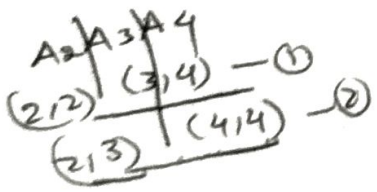
Hence $O(n)$ — instead of $\boxed{O(2^n)}$

split possible

$(j-i)$

i.e $m(2,4)$

$\boxed{4-2=2}$



Dynamic programming bottom UP

in the privious example—

	no. subchain	size
(1,1) (2,2) (3,3) (4,4)	4	1
(1,2) (2,3) (3,4)	3	2
(1,3) (2,4)	2	3
(1,4)	1	4
	4+3+2+1	

size - 2 bcz

↓

(3,4) → two matrix 3 and 4

size - 4 bcz

(1,4)

↓

1,2,3,4 ✓

Q- For matrix A(1 - - - n)

no. of chains	size
n	1
(n-1)	2
(n-2)	3
⋮	⋮
1	n

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

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

↓

order of subchains.

Multiply $A_1 \quad A_2 \quad A_3 \quad A_4$

$1\times2 \quad 2\times1 \quad 1\times4 \quad 4\times1$

$A_{(1\cdots4)}$ $\quad P_0\times P_1 \quad P_1\times P_2 \quad P_2\times P_3 \quad P_3\times P_4$

$\Downarrow$

$(14) = ((1,2)(3,4))$

0 (1,1)	0 (2,2)	0 (3,3)	0 (4,4)
2 (1,2)	0 (2,3) $\boxed{k=3}$	4 (3,4) $k=3$	
6 (1,3)	6 (2,4) $\boxed{k=2}$		
(1,4) 7 $\boxed{k=2}$			

$\boxed{k=1}$ $\quad k=2$

$\Rightarrow \underbrace{O(n^2)}_{\text{total no. of entry}} \cdot \underbrace{O(n)}_{\text{time to compute one entry}}$

$= \boxed{O(n^3)}$ time compl

$[O(n^2) \rightarrow$ space complexity$]$

Imp:- (1,2) means $A_1 \times A_2$

$(1\times2)(2\times1)$

2.

$$(1,3) = \min \begin{cases} (1,1) + (2,3) + P_0P_1P_3 \\ 0 + 0 + 8 = 16 \\ (1,2) + (3,3) + P_0P_2P_3 \\ 2 + 0 + 4 = 6 \end{cases}$$

$$(2,4) = \min \begin{cases} (2,2) + (3,4) + P_1P_2P_4 = 6 \\ 0 + 4 + 2 \\ (2,3) + (4,4) + P_1P_3P_4 \Rightarrow 16 \end{cases}$$

$$(1,4) = \min \begin{cases} (1,1) + (2,4) + P_0P_1P_4 \\ 0 \quad 6 \quad 2 \quad 8 \\ (1+2) + (3+4) + P_0P_2P_4 \\ 2 + 4 + 1 \quad 7 \\ (1,3) + (4,4) + P_0P_2P_3 \\ 6 + 0 + 4 \quad 10 \end{cases}$$

MATRIX-chain (P) → set of dimensions {

```
n = p.length-1
let m[1---n, 1---n] and s[1---n-1, 2---n-1]
                                be new tables

for i=1 to n
    m[i,j]=0
for l=2 to n //l is the chain length
    for i=1 to (n-l+1)
        j=i+l-1
        m[i,j]=∞
        for k=i to j-1
        q=m[i,k]+m[k+1,j]+ p_{i-1} p_k p_j
        if q<m[i,j]
            m[i,j]=q
            s[i,j]=k
        return m and s
}
```

(34-41)

Top Down dynamic Memoization

```
MEMOIZED-MATRIX-CHAIN(P) {
    n = P.length-1
    let m[1---n, 1---n] be a new table
    for i=1 to n
        m[i,j] = ∞
    return LOOKUP-CHAIN(m,P,1,n)
}

LOOKUP-chain(m,P,i,j) {
    if m[i,j] < ∞
    return m[i,j]
    if i==j
        m[i,j] = 0
    else for k=i to j-1
    q = LOOKUP_chain(m,P,i,k)
        + LOOKUP-chain(m,P,k+1,j)
            + P_{i-1} P_k P_j

    If q < m[i,j]
        m[i,j] = q
        return m[i,j]
}
```

Vi imp.

Time complexity $O(n^2)O(n) = O(n^3)$

space complexity $\underbrace{O(n^2)}_{\text{table}} + \underbrace{O(n)}_{\text{stack}} = \boxed{O(n^2)}$

2011

Q) Find the minimum no. of multiplication required for the

$$\underset{(10\times100)}{M_1}\ \underset{(100\times20)}{M_2}\ \underset{(20\times5)}{M_3}\ \underset{(5\times80)}{M_4}$$

$$P_0\times P_1 \quad P_1\times P_2 \quad P_2\times P_3 \quad P_3\times P_4$$

Soln

(1,1) 0	(2,2) 0	(3,3) 0	(4,4) 0
(1,2) 20,000	(2,3) 10,000	(3,4) 8,000	
(1,3) 15,000	(2,4) 50,000		
(1,4) 19,000			

$\boxed{K=1}$ $\boxed{K=3}$ $K=3$

$$(1,3) = \min \begin{cases} (1,1) + (2,3) + P_0P_1P_3 = 1500 \\ (1,2) + (3,3) + P_0P_2P_3 \end{cases}$$

$0 + 10{,}000 + 5000$

$20{,}000 + 0 \quad 1000$

$= 21{,}000$

(1,3) splits at $\boxed{K=1}$

$4-2=2$ possible split

$$(2,4) = \min \begin{cases} (2,2) + (3,4) + P_1P_2P_4 \\ (2,3) + (4,4) + P_1P_3P_4 \end{cases}$$

$0 \quad 8{,}000 \quad 160000$

$10{,}000 \quad 0 \quad 40000$

168000

$\boxed{50{,}000}$

Ans

((1)(2 3)(4))

19,000

3 splits

$$(1,4) = \begin{cases} (1,1) + (2,4) + P_0P_1P_4 \\ (1,2) + (3,4) + P_0P_2P_4 \\ (1,3) + (4,4) + P_0P_3P_4 \end{cases}$$

$0 + 50{,}000 \quad 80{,}000 = 130{,}000$

$20{,}000 \quad 8{,}000 \quad 16{,}000 \quad 44{,}000$

$15{,}000 + 0 + 4000 \quad 19{,}000$

[Longest common sequence]

Difference b/w string and sequence

→ RAVINDRA
12345678

→ if index are in sorted order with no gap they are string (substring)

RAVI → substring
1234

→ if there is gap b/w the string indexes is (sequence)

RA NA
1 2 5 8 → subsequence.

LCS :→

D_1 = AGCCTCAGT

D_2 = GCCT

D_3 = CCTCA

find LCS among $D_1 D_2 D_3$

CCTCA (5)

D_1, D_2 → GCCT (4)

D_2, D_3 → CCT (3)

✓ Hence CCTCA is LCS Between D_1 and D_3
5

Note:—

A = RAVINDRA → 2^m (m)

B = AJAY → n

{AA} → 2 → LCS

{A} → 1

{ } — φ

$O(2^m)$ Find all subseq of 'A'

$O(n2^m)$ Find for each seq whether it is a subseq 'B'

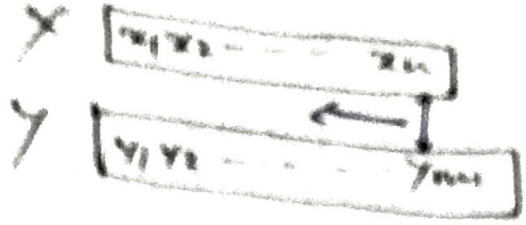
$O(2^m)$ Find the largest among common ss.

⇓

$O(n2^m)$ → take exponential not usable

optimal substructure and recursive equation →

to find LCS



start comparing from the end and proceed towards the left side

$$C[i,j] = \begin{cases} 0 \text{ , } i=0 \text{ or } j=0 \\ 1 + c[i-1, j-1] \text{ ; } i,j>0 \text{ and } x_i = y_j \\ \max(c[i-1,j], c[i,j-1]) \text{ } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

Base condition either no X or no Y
Hence $c[i,j] = 0$,

Best case i/p:→

$X = \{A, A, A, A\}$

$Y = \{A, A, A, A\}$

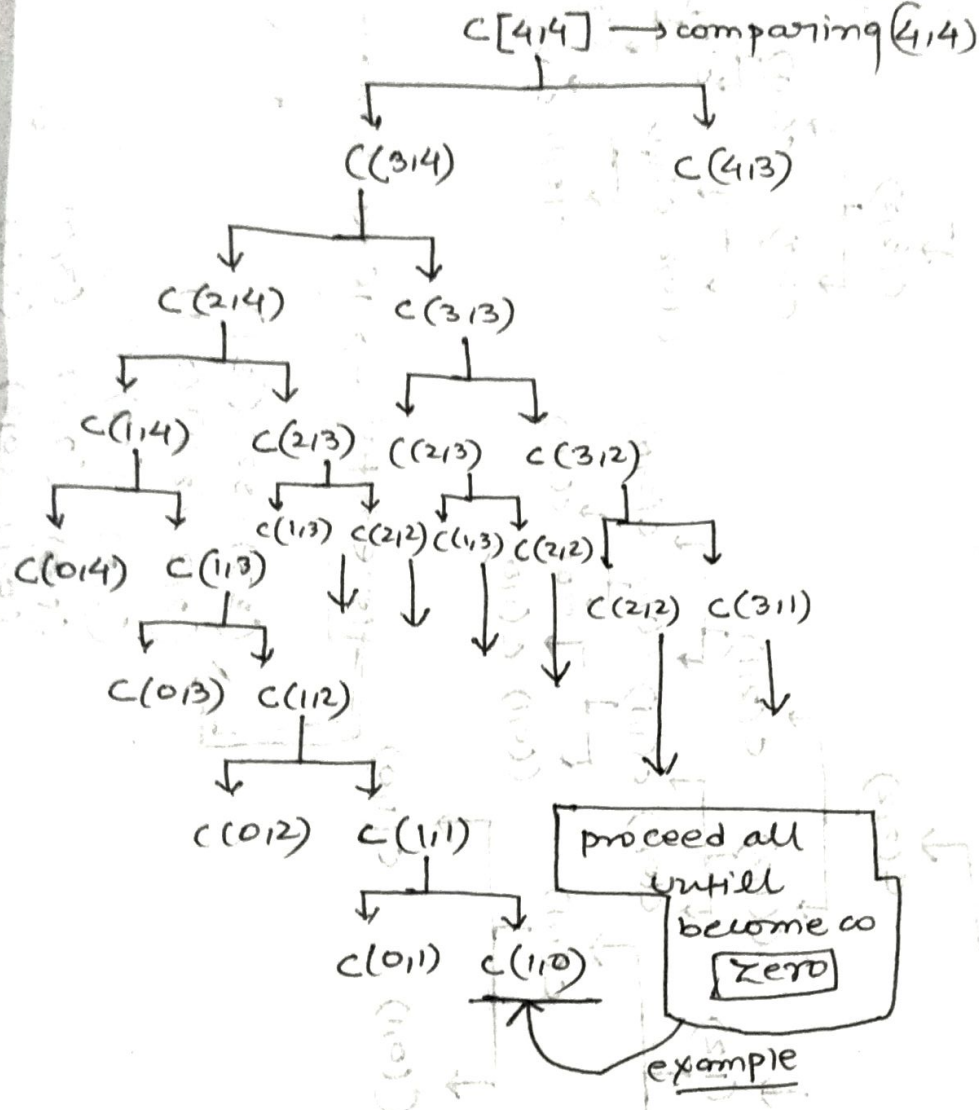
$\boxed{\begin{matrix} O(n) \\ O(n) \end{matrix}}$

$c(4,4)$
$1 + c(3,3)$
$1 + c(2,2)$
$1 + c(1,1)$
$c(0,0)$

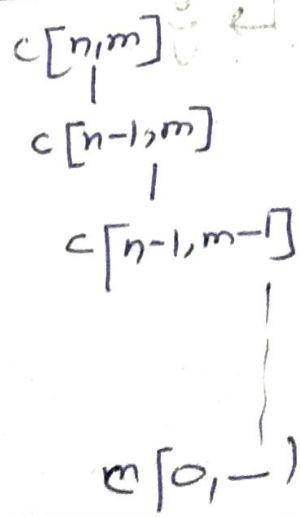
worst case :-

$$X = \{\overset{1}{A}, \overset{2}{A}, \overset{3}{A}, \overset{4}{A}\}$$

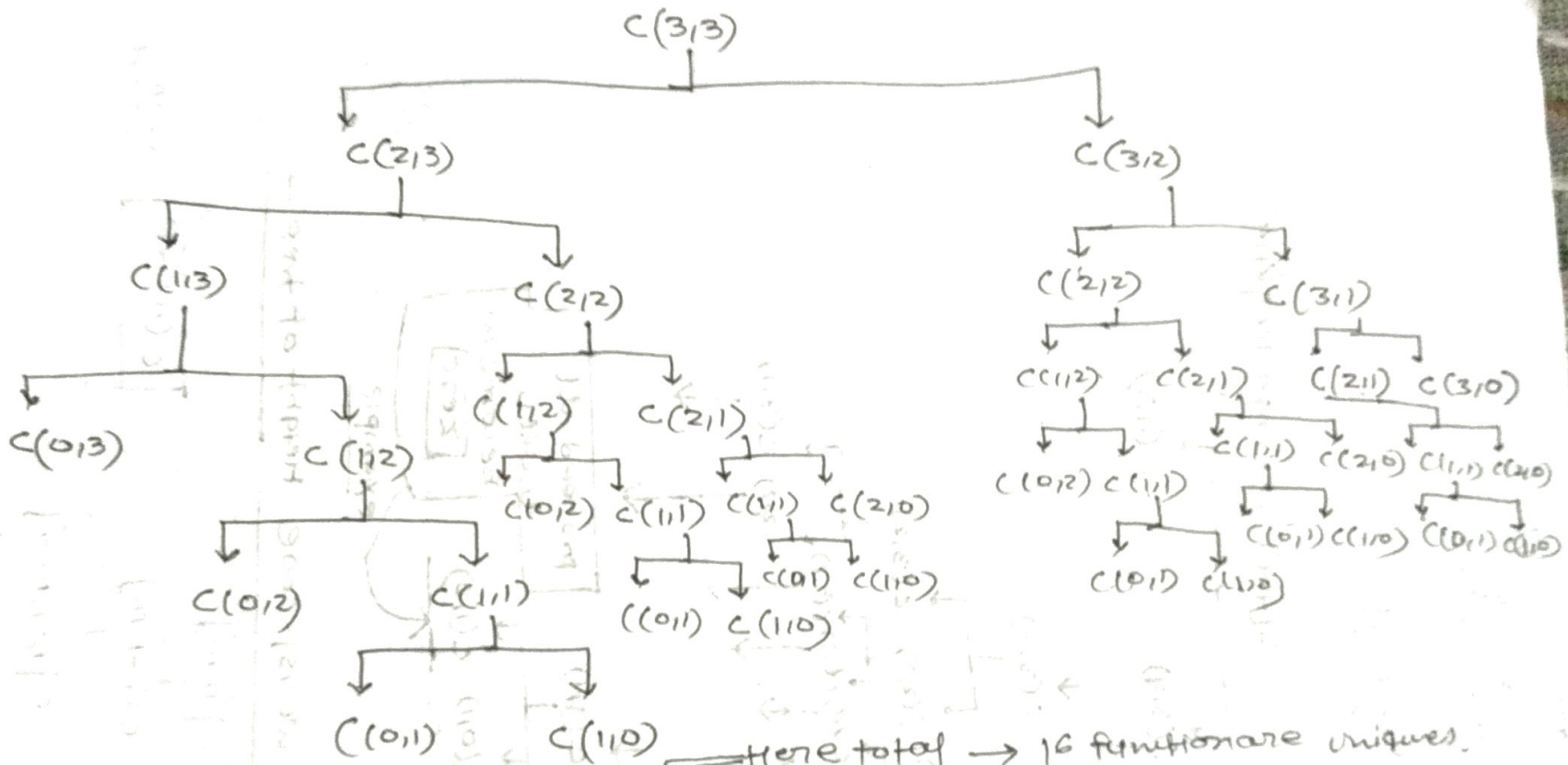
$$Y = \{\underset{1}{B}, \underset{2}{B}, \underset{3}{B}, \underset{4}{B}\}$$



Intre the worst case Height of tree →



$\boxed{O(n+m)}$ → time complexity



Here total → 16 functionare uniques.

exp (1,3)

(3,3) (3,2) (3,1)

(2,3) (2,2) (2,1)

(1,3) (1,2) (1,1)

(1,0) (2,0) (3,0) (0,0)

(0,1) (0,2) (0,3)

Here are 16 values.
Hence make table for 16 cells.

	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33

all circled value = $\boxed{0}$ ($\because$ either $i=0$ or $j=0$)

Now compute $\boxed{11}$ and goes row-wise or column wise (select either of one).

{ time complexity $(O(mn))$
space complexity $O(mn)$ }

example $X(AAB)$
$Y(ACA)$

$\boxed{\text{V.imp}}$ ✓

		Y	A 1	C 2	A 3
X		0	0	0	0
A	1	0	1	1	1
A	2	0	1	1	2
B	3	0	1	1	2

for 11 AA matches → Hence 1+ diag

12 AC → not match

02 UP, 11 left → which is maximum

V.imp

① Trace the path from which last element come from (Note at up arrow)
there go to begning in between (AA) (write Right to left)
If for any block i and j
Note that. index $\boxed{AA}$

Bottom up Dynamic programming Algo.

```
LCS (x,y) {
        m = x.length
        n = y.length
    let c[0---m, 0--n] be a new table
    for i=1 to m
        c[i,0] = 0
    for j=0 to n
        c[0,j]=0
    for (i = 1 to m)
        for (j = 1 to n)
        if (xi = yi)
            c[i,j] = c[i-1,j-1] + 1
        else c[i,j] = max (c[i-1,j]
                          c[i,j-1])
        return c}
```

Time complexity $O(mn)$

Space complexity $O(mn)$

Q⇒ A = qpqrr

B = pqprqrp

x → length of LCS

y → How many such LCS possible

x+10y

4 3

34

	o	q	p	q	r	r
o	0	0	0	0	0	0
p	0	0	1	1	1	1
q	0	1	1	2	2	2
p	0	1	2	2	2	2
r	0	1	2	2	3	3
q	0	1	2	3	3	3
r	0	1	2	3	4	4
p	0	1	2	3	4	4

→ matu...

this not come fm diagonal bcz if come from diagon increas by 1 wui not here.

pqrr

likewise qpqr / qprr → 3 one possible

⟹ selection of matching character is decided by

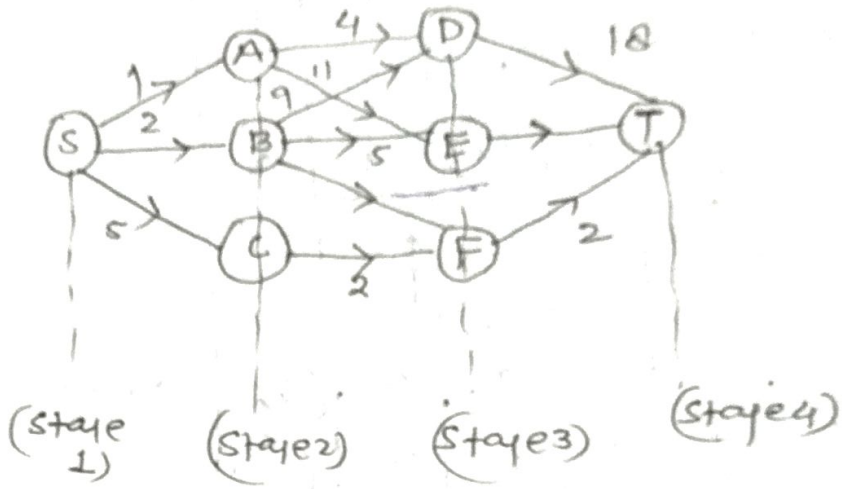
⟹ let, for down one $\boxed{4}$ → in horizontal (p) and vertical (q)

⇓

not matched Hence (Not counted)

⟺ start writing from Right side and move towards left.

Multi stage Graph



Multi-stage Graph —

1) there are "no edges" between the same stage.

example stage 2 (A) (B) (C)

2) there are only edges from (i to i+1)

To find the sortest path b/w S→T if we go to Greedy algorithm.

$1+4+18=23$ $S \xrightarrow{1} A \xrightarrow{4} D \xrightarrow{18} T$ 23

$5+2+2=9$

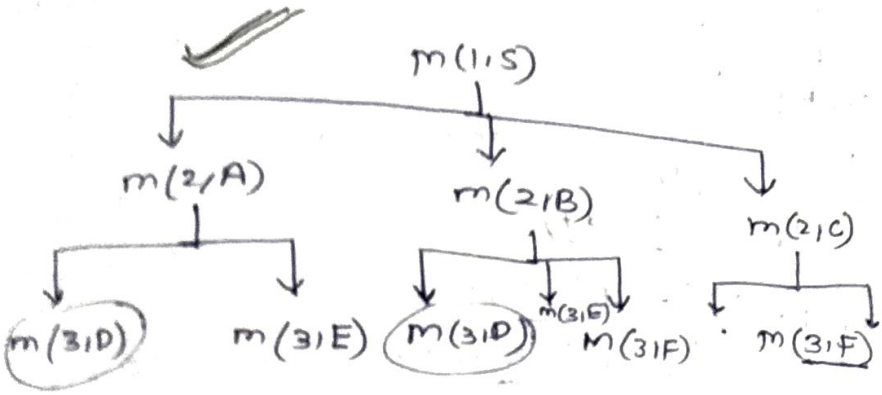
Hence Here Greedy fails.

→ For Applying dynamic programming —

1) the problem can be divided into sub parts.

2) the overlapping of the same data is required

$$M(1,S) = \min \begin{cases} S \to A + m(2,A) \\ S \to B + m(2,B) \\ S \to C + m(2,C) \end{cases}$$



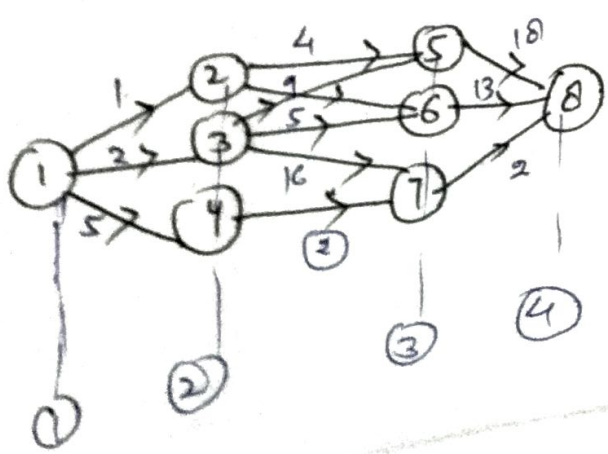
$$\boxed{M(N-1, i) = i \to T}$$

$\boxed{K^n}$

Here for Recursion $\boxed{2^K}$ time is required.

<u>Here</u> overlapping is also present Hence Dynamic programming is applicable.

Number the vertex in Ascending order —



$T[i] = \min_{\text{for all } j=(i+1) \text{ to } n} \{ cost(i,j) + T[j] \}$

T	1	2	3	4	5	6	7	8
	9	22	18	4	18	13	2	0

$T[6] = \min \begin{cases} cost(6,7) + T(7) \\ cost(6,8) + T(8) \end{cases}$

$\infty + 2$

$13 + 0$

(if we have to find out T[6], take the vertices which are greater than 6. means, 8 and 7)

$T[5] = \min \begin{cases} cost(5,6) + T(6) \\ cost(5,7) + T(7) \\ cost(5,8) + T(8) \end{cases}$

10

2

$T[6] = \min \begin{cases} cost(6,7) + T(7) \\ cost(6,8) + T(8) \end{cases} = \min \begin{cases} \infty + 13 \\ \infty + 2 \\ 10 + 0 \end{cases}$ #

Time complexity

$O(n^2)$

$O(V^2) = O(E)$

0/1 Knapsack

capacity = 6

objects	1	2	3
weight	1	2	4
profit	10	12	28
P/W	10	6	7

By Greedy

③	28	4
①	10	1

38

③	28	4
②	12	2

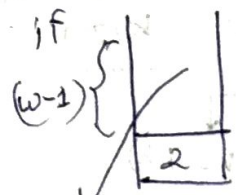
40

Here Greedy method not applicable.

why Greedy method is fails in the case of 0/1 knapsack

object	1	w
weight	2	w
profit		

if



w size

(never can be put)

Substructure and Recursive equation :-

1 2 3 4 5 ---- n

⇓ possibility to put into bag.

$2^n \times 2 \times 2 \times 2 \times \text{----} \times 2$

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

if there is repetation then we can Reduce time complexity.

#. $ks(i, w) = \begin{cases} \max(p_i + ks(i-1, w-w_i), ks(i-1, w)) & \text{(include)} \quad \text{(not include)} \\ 0, & i=0 \text{ or } w=0 \\ ks(i-1, w), & w_i > w \end{cases}$

i → no. of items

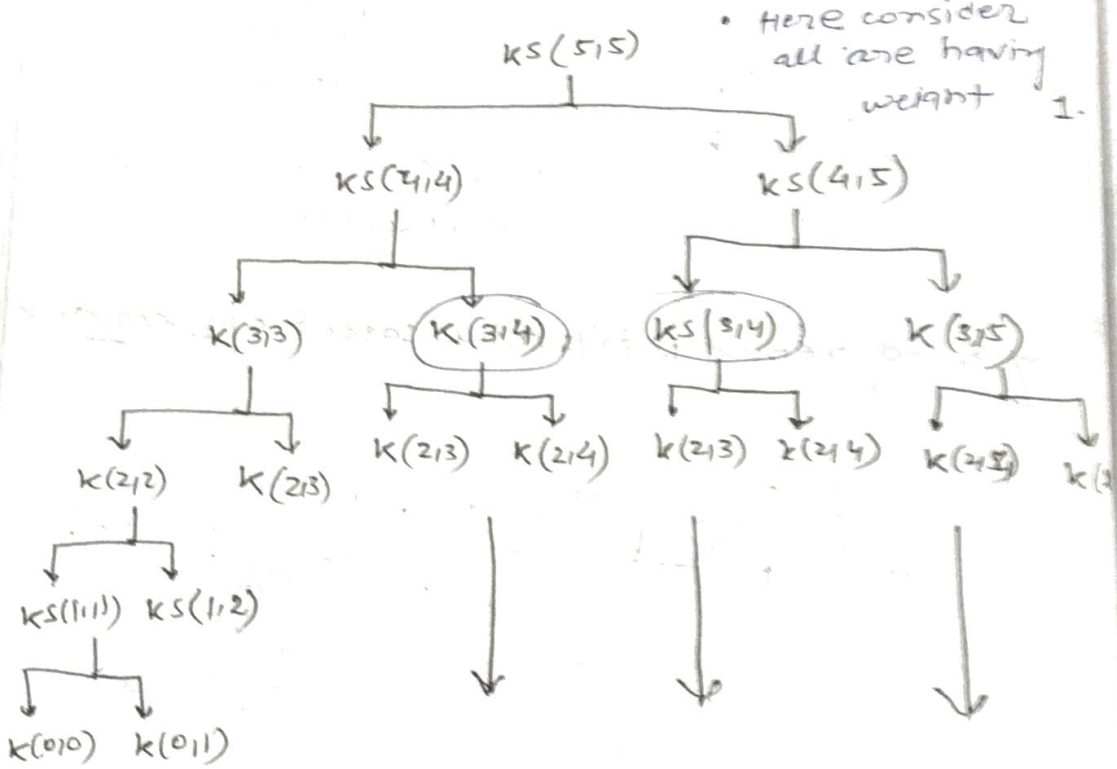
w → total capacity

$ks(n, C)$

n → no. of elements

C → capacity of bag

• Here consider all are having weight 1.



Here the substrings are repeating. from we can apply dynamic programming.

the unique substring = (5×5)

we can Reduce complexity from

$$\boxed{O(2^n) \longrightarrow O(nw)}$$

example-

C=6

w	1	2	3
P	1	2	4
	10	12	28

Bottom up

$\boxed{\text{Time complexity} = O(nw)}$

$\boxed{\text{Space-complexity} = O(nw)}$

Here $w=6$ $n=3$

(3X6) matrix

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	10	10	10	10	10	10
2	0	10	12	22	22	22	22
3	0	10	12	22	28	38	$\boxed{40}$

$T(n) = \min\{O(2^n), O(nw)\}$

$KS(1,1) =$ Here object 1 having weight 1 and capacity is also 1 Hence combe include.

$$KS(1,1) = \max\begin{cases} p_i + KS(0,0), KS(0,1) \\ 10+0, \ 0 \end{cases}$$

$= 10$

$$K(1,2) = \max\begin{cases} 10 + K(0,1) \\ K(0,3) \end{cases} = 10$$

$K(2,1) =$ Here weight of 2nd object is $\boxed{2}$ and capacity is 1 Hence not included

$$KS(2,1) = \{KS(i-1,w)\}$$

$= K(1,1) = 10$

0

$$K(2,2) = \max\begin{cases} 12 + K(1,0) \\ K(1,2) \end{cases} = \boxed{12}$$

10

Bottom up Dynamic programming algorithm and analysis

input: $\{w_1, w_2, \ldots w_n\}$, c $\{p_1, p_2 \ldots p_n\}$

output: $T(n, c)$

```
for i=0 to i=c do
        T[0,i]=0
for i=1 to n {
        T[i,0]=0
        for (j=1 to c) do
        IF (wi ≤ j) and T[i-1, j-wi] + pi >
                                T[i-1, j]
        then T[i,j] = T[i-1, j-wi] + pi
        else T[i,j] = T[i-1, j]
        }
```

Time complexity $O(nw)$ ✓

$O(2^n)$ → Brut force method.

Space complexity

Subset sum

Time-comp / Space-comp $\boxed{O(nw)}$

selecting from the subset to make the sum as given.

$\{a_1, a_2, a_3, \ldots, a_n\}$ if

$2 \times 2 \times 2 \times \ldots$ 2 times $= \boxed{2^n}$

$$SS(i,s) = \begin{cases} SS(i-1,s) & , s < a_i \\ SS(i-1, s-a_i) \text{ or } SS(i-1,s); & s > a_i \\ \text{True} & ; s = 0 \\ \text{False} & ; i = 0, s \neq 0 \end{cases}$$

not include

→ on drawing Recursion tree, we find out the subset are reaping. Hence DP can be applied.

(4)

Q→ $\{6,3,2,1\}$ is there is subset whose sum is 5. (5)

Sol→

S → Rmo or cmo

u ↓	0	1	2	3	4	5
0	T	F	F	F	F	F
1	T	F	F	F	F	F
2	T	F	F	T	F	F
3	T	F	T	T	F	T
	T	T	T	T	T	T

$SS(3,5) = S(2,3) \vee S(2,5)$

$SS(1,1)$ Here weight of 1 is greater than sum $\boxed{6 > 5}$ Hence no not include

$\Downarrow$

$S(0,1)$

$\boxed{S(2,2)}$ → Here element weight 3

sum = 2

Hence $S(2,2) = S(1,2)$