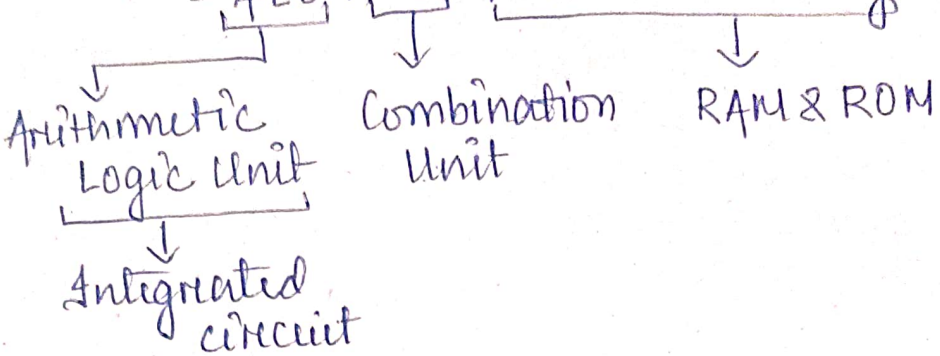
Date- 16/1/25

→ Computere - system + I/O devices

= CPU + main memory + I/O devices

= ALU + CU + main memory + I/O devices



→ Program in execution is called process.

→ steps to take particulare input to generate particular output is called algorithm.

→ Algorithm is set of finite number of steps to solve a particulare problem.

Algorithm should satisfy following criteria:-

1. Input/Output criteria- Must take 0 or more input and generate one output.
2. Effectiveness :- Each step must contribute something to generate output.
3. Finiteness :-
   → Number of steps in algorithm must be countable by execution.
4. Definiteness :-
   → No ambiguity, no duality in the steps.
   → Should be unique values.

ALGORITHM

Algorithm Name_of_algorithm(parameter)
// Define parameter
input:-
output:-
BEGIN
{
1.
2.
3.
!
!
END}

Eg:-

Algorithm Add_two_number(x,y,z)
// x and y are two input and z is output
input:- x and y
output:- z
BEGIN
1. Read x and y
2. z = x+y
3. Print z
END

→ Amount of time required to generate output is called execution time.

→ Let P is a program

E(P) = C(P) + R(P)

- E(P): execution time factor over C(P)
- C(P): compilation time
- R(P): time spent fore running the program

T(n) ∝ n
E(P) ∝ R(P)
⇓
T(P)
⇓
T(n) – Time required for n input program

T(n) ∝ R(P)
T(n) ∝ n – n - no. of inputs
↓
Time

T(n) = kn where k = +ve constant
T(n) ≤ kn
T(n) ≥ kn
? ≤ T(n) ≤ ?

Case-1 :- [T(n) ≤ kn]

T(n) = O(n)
↓
Big-Oh

k is +ve constant
& k ≥ n0
n0 = it is the initial value

Order :- compilation time + Run time
↓ (It can't be measured in ns, ms, sec)
↓

Asymptotic symbols :-

1) Big-Oh → Max^m
   - Big-Oh
   - little-Oh
2) Omega → min^m
   - Big omega
   - little omega
3) theta → Range (boundary between two values)

T(n) (≤) kn → Big-oh
T(n) (<) kn → little-oh

Example :-

T(n) = 5n+7. Find no and k value and represent it in Big-Oh.

T(n) ≤ 5n+7
T(n) ≤ 5n+n

$T(n) \leq 6n \quad k=6$

$5n+7 \leq 6n \rightarrow n_0 = 7$

(iii) $T(n) = 612$

$T(n) \leq 612n$

$k = 612$

$612 \leq 612n$

$n_0 = 1$

Dt - 28/1/25

$T(n) \leq kg(n), \; n \geq n_0$ ← initial values

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

Example:-

(1) $T(n) = 7n+6 = O(n)$

$= 7n+n$

$= 8n$

$T(n) = 8n \rightarrow k=8$

$7n+6 \leq 8n$

$\rightarrow 48 \leq 48 \quad n_0 = 6$

$\Rightarrow O(n)$

→ It requires minimum 6 number of inputs to execute the program.

2) $T(n) = 325 = O(1)$

$T(n) \leq 325 \times 1$

$\leq 325 \times n^0 \quad k = 325$

$325 \leq 325 \times n^0 \quad n_0 = 1$

$= O(n^0) = O(1)$

3) $T(n) = 2n^2+5$

$T(n) \leq 2n^2+5$

$T(n) \leq 2n^2+n$

$T(n) \leq 2n^2+n^2$

$T(n) \leq 3n^2 \quad k=3$

$2n^2+5 \leq 3n^2 \quad n_0 = 3$

$= O(n^2)$

4) $T(n) = 2n^3+n^2+2 = O(n^3)$

$T(n) \leq 2n^3+n^2+2$

$T(n) \leq 2n^3+n^2+n$

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

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

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

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

$= O(n^3) \quad k=3$

(*) Highest degree power we are considering and rest we are ignoring.

Generalize:-

(*) $T(n) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$

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

Properties:-

1) $T_1(n) = O(g_1(n))$

2) $T_2(n) = O(g_2(n))$

3) $T(n) = T_1(n) + T_2(n) = O(\max(g_1(n), g_2(n)))$

4) $T(n) = T_1(n) \cdot T_2(n) = O(g_1(n) \cdot g_2(n))$

5) $T(n) = T_1(n) \div T_2(n) = O\left(g_1(n) \cdot \frac{1}{g_2(n)}\right)$

Ex:- $T(n) = 2n^2 \log(n^2+5n+7) \log\log(n^3+7) + 2n^3 + \log(n^3+8)\, n^2+n$

$T(n)$

$T_1(n) = 2n^2 \log(n^2+5n+7) \log\log(n^3+7)$

$T_1(n) = n$

$T_3(n) = \log(n^3+8) n^2$

$T_2(n) = 2n^3$

$T_4(n) = n \Rightarrow O(n)$

$T_3(n) = \log(n^3+8)\, n^3$ ($\log(n^3+8) \to T_5$, $n^3 \to T_6$)

$T_6 = n^2 \Rightarrow O(n^2)$

$T_5 = \log(n^3+8) = O(\log n)$

$n^3+8 = O(n^3)$

$\log(n^3+8) = O(\log n^3) = O(3 \log n) = O(\log n)$

$T_3 = O(n^2 \log n)$

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

$T_2 = 2n^3 = O(n^3)$

$T_1 = \frac{2n^2}{T_7} \underbrace{\log(n^2+5n+7)}_{T_8} \underbrace{\log\log(n^3+7)}_{T_9}$

$T_7 = 2n^2 = O(n^2)$

$T_8 = \log(n^2+5n+7) = O(\log n)$

$T_9 = \log\log(n^3+7) = O(\log\log n)$

$T_1 = O(n^2 \log n \cdot \log\log(n))$

$T_2 = O(n^3)$

$T_3 = O(n^2 \log n)$

$T_4 = O(n)$

maximum $= O(n^3)$

$1 < n < \log n < n^2 + n\log n < n^3 < 2^n$

increasing order.

Step count method

Algorithm sum_of_two(x_1, x_2)	s/t	F/e
// x_1 and x_2 are two numbers	0	—
Input:- Two number x_1 and x_2	0	—
Output:- Add of x_1 and x_2	0	—
BEGIN	0	—
1. Read x_1 and x_2	1	1
2. $x = x_1 + x_2$	1	1
3. print x	1	1
4. END	0	—
	Total	3

Algorithm sum_of_n_no	s/t	F/e
BEGIN	0	—
1. for($i=0$; $i<n$; i++)	0	—
2. {	1	$n+1$
3. Read x_1 and x_2	0	—
4. $x = x_1 + x_2$	1	n
5. }	1	n
6. print x	0	—
END	1	1
	0	—
	Total	$3n+2$

sum of n no. $\Rightarrow T(n) = 3n+2$

$T(n) = O(n)$

Dt-29/1/25

→ switch and if statement will execute only once.

→ loop statement will execute more than one depending upon condition.

$$F(n) = \begin{cases} 2n+5 & n>3 \\ 5 & \text{otherwise} \end{cases}$$

$$F(n) = \begin{cases} 4F(n/2)+7 & n>0 \\ 3 & \text{otherwise} \end{cases}$$

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

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

$T(n/2) = 2[2T(n/4) + c(n/2)] + cn$

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

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

$= 4[2T(n/8) + c(n/4)] + cn$

$= 8T(n/8) + 3cn$

At 3rd step:-

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

At ith step:-

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

Let $2^i = n$

$i = \log_2 n$

putting in eqn(i)

$= 2^n T(n/n) + \log_2 n\, cn$

$= n(a) + \log_2 n\, cn$

$= an + cn\log_2 n$ → maximum of these two will considered

$= O(n \log n)$

Addition of 'n' nos using recursive funcn:-

```
Algorithm RecursiveAddn(a,n)
// a is an array contain n no. of elements
input:- array a with n no. of inputs
output:- sum of 'n' numbers
BEGIN
1. if(n>=0)
2. {
3. sum = 0
4. Return(sum)
5. }
6. else
7. {
8. return(sum + a[n] + RecursiveAddn(a,(n-1))
```

n<=0	n>0	n<=0, n>0
0		
0		
0		
0		
0		
→1		
→0		
→1		
→1		
→0		
→0		
→0		
→1		

9. }
END

$$T(n) = \begin{cases} 3 & n \le 0 \\ 3 + T(n-1) & \text{otherwise} \quad n > 0 \end{cases}$$

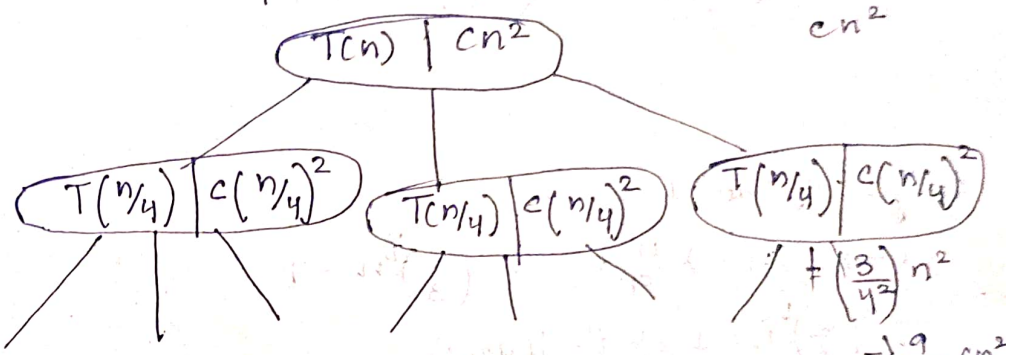
$T(n) = 3 + T(n-1)$
$= 6 + T(n-2)$
$= 9 + T(n-3)$
At 3rd step
$= 3 \cdot 3 + T(n-3)$
At ith step
$= i \cdot 3 + T(n-i)$ → 0 bit $i = n$
$= 3n + T(0)$
$= 3n + 3$
$= O(n)$

Recurrence eqn can be solved by methods:-
1) Substitution method
2) Recurrence tree method
3) Master method
4) Changing variable method

Recurrence Tree method:-
$n - (n-1)$ tree → B-tree
Ternary tree → 2 data → both recursive and non-recursive
Binary tree → 1 data → non-recursive part
coefficient of problem = no. of child.

Dt - 30/1/25

$T(n) = \underbrace{3T(n/4)}_{\text{Recursive part}} + \underbrace{cn^2}_{\text{non-recursive part}}$ → cost of problem



T(1) T(1) — —
Height of the tree
From non-recursive part:-
$n^2 \longrightarrow \left(\frac{n}{4}\right)^2 \longrightarrow \left(\frac{n}{4^2}\right)^2 \cdots \left(\frac{n}{4^i}\right)^2 = 1$

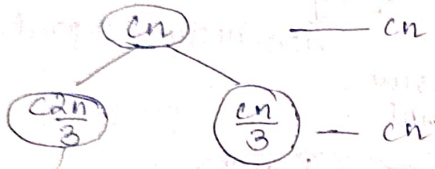
$\frac{n}{4^i} = 1$

$\Rightarrow 3 = 4^i$
$\Rightarrow i = \log_4 3$

No. of leaf node
$n^2 \longrightarrow \left(\frac{3}{4^2}\right) n^2 \longrightarrow \left(\frac{3}{4^2}\right)^2 n^2 \cdots \left(\frac{3}{4^2}\right)^i n^2 = 1$

$= \sum_0^i \left(\frac{3}{4^2}\right)^i cn^2 + n^{\log_4 3}$
$= \sum_0^{\log_4 3} \left(\frac{3}{4^2}\right)^i cn^2 + n^{\log_4 3}$ ← constant
$= cn^2 + n^{\log_4 3}$
$= an^2 + b = O(n^2)$

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



Height of the tree

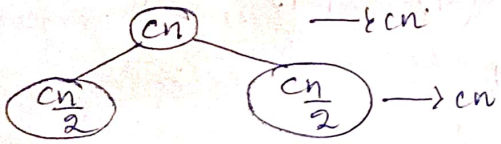
$n \to \frac{2}{3}n \to \frac{4}{9}n \cdots \left(\frac{2}{3}\right)^i n = 1$

$\left(\frac{2}{3}\right)^i n = 1 \Rightarrow i = \log_{3/2} n$

$cn + cn + cn + \cdots \log_{3/2} n$ times

$= \log_{3/2} n \; cn = O(n \log n)$

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



$n \to \frac{n}{2} \to \frac{n}{2^2} \cdots \frac{n}{2^i} = 1$

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

$n = 2^i$

$\Rightarrow i = \log_2 n$

$cn + cn + cn + \cdots \log_2 n = cn \log_2 n$

$= O(n \log n)$

Dt-4/2/25

Master method

$T(n) = aT(n/b) + F(n) \quad a \geq 1, b > 1$

$F(n) = C(n) + D(n)$

$C(n)$ ↓ combine time

$D(n)$ → Divide time

(1) If $F(n) = O(n^{\log_b a - \epsilon})$ and $\epsilon > 0$ then $T(n) = \Theta(n^{\log_b a})$

(2) If $F(n) = O(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$

(3) If $F(n) = \Omega(n^{\log_b a + \epsilon})$ and $\epsilon > 0$ then $T(n) = \Theta(F(n))$

Q1) $T(n) = 9T(n/3) + n$

$a = 9, \; b = 3$

$F(n) = n$

$F(n) = n^{\log_3 9} = n^2$

$O(n^{\log_3 9}) = O(n^2)$

Q2) $T(n) = 2T(n/2) + O(n)$

$a = 2, \; b = 2$

$F(n) = O(n)$

$F(n) = n^{\log_2 2} = n$

$O(n) = F(n)$

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

Q3) $T(n) = 1T(2n/3) + 1$

$a = 1 \quad b = 3/2$

$F(n) = 1$

$F(n) = n^{\log_{3/2} 1} = n^0 = 1$

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

Q4) $T(n) = 3T(n/4) + n\log n$

Given $a = 3$, $b = 4$

$F(n) = n\log n =$

Calculate

$F(n) = n^{\log_4 3} = n^{0.79} = n^0 = 1$

Algorithm Insertion Sort (A, n) — s/e — f/e

//A is an array consist of n no. of elements

input:- Array with n numbers of elements

Output:- sorting of elements

BEGIN			
1. For j = 2 to length(A)	→	1	n
2. {			
3. key = A[j]	→	1	n-1
4. i = j-1	→	1	n-1
5. while i > 0 and A[i] > key	→	1	$\sum_{j=2}^{n} t_j$
6. {			
7. A[i+1] = A[i]	→	1	$\sum_{j=2}^{n} t_j - 1$
8. i = i-1	→	1	$\sum_{j=2}^{n} t_j - 1$
9. }			
10. A[i+1] = key	→	1	n-1
11. }			
END			

①	②	③	④	⑤
2	1	7	5	4

Step 1 :- j = 2 to 5

Step 2 :- j = 2
key = 1
i = 2-1 = 1

1 2 7 5 4

Step 3 :- j = 3
key = 7
i = 3-1 = 2

Step 4 :- j = 4
key = 5
i = 4-1 = 3

Step 5 :- j = 5

Total cost :- $T(n) = n + n-1 + n-1 + n-1 + \sum_{j=2}^{n} t_j + \sum_{j=2}^{n} t_j - 1 + \sum_{j=2}^{n} t_j - 1$

Dt - 5/2/25

Case 1 :- All element are same

case 2 :- when elements are in reverse order.

while loop -> execute

$\sum_{j=2}^{n} t_j + \sum_{j=2}^{n} t_j - 1 + \sum_{j=2}^{n} t_j - 1$

case 3 :- when element are in increasing order

case 4 :- when element are some in sorted and some in unsorted

case 2 :-

$T(n) = n + n-1 + n-1 + \underbrace{\sum_{j=2}^{n} t_j}_{= \frac{n(n+1)}{2} - 1} + \sum_{j=2}^{n} t_j - 1 + \underbrace{\sum_{j=2}^{n} t_j - 1}_{= \frac{(n-1)n}{2}}$

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

$= O(n^2)$

case-3/case-1

$T(n) = n + n-1 + n-1 + \sum_{j=2}^{n} t_j + \left(\sum_{j=2}^{n} t_{j-1} + \sum_{j=2}^{n} t_{j-1}\right) + n-2$

$= n+n-1+n-1+n+n-1.$

$= O(n)$

$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} 1 = n$

not execute

Different Techniques

1) Divide and conquer (botton-up approach)
2) Greedy
3) Dynamic programmming.
4) Backtracking
5) batch & bound.

Algorithm Divide and Conquer

```
Algorithm DandC(P)
//P is a problem
BEGIN
1. If small(P)
2. return S(P)
3. else
4.    Divide P into P1, P2, --Pk
5.        return (S(P1), S(P2), S(P3) .. S(Pk))
END
```

$$T(n) = \begin{cases} g(n) & n \text{ is small} \\ T(n_1) + T(n_2) + \dots + T(n_k) & n \text{ is large} \end{cases}$$

$$T(n) = \begin{cases} T(1) & n = 1 \\ a\left(T\left(\frac{n}{b}\right)\right) + c(n) + D(n) & \end{cases}$$

- a: no. of sub problem
- $\frac{n}{b}$: size of each problem
- $c(n)$: combine time
- $D(n)$: divide time

$$T(n) = \begin{cases} T(1) & n = 1 \\ aT(n/b) + F(n) & \end{cases}$$

Big-O

$T(n) < cg(n)$

$\Rightarrow O(g(n))$

$T(n) \leq cg(n)$

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

$T(n) = 5n^2+6n+n = O(n^2)$ — prove

$\lim_{n\to\infty} \frac{f(n)}{g(n)} =$ ~~const / 0~~ zero

$F(n) = O(g(n))$

$F(n) = o(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{5n^2+6n+n}{n} = \lim_{n\to\infty}\left(\frac{5n^2}{n} + \frac{6n}{n} + \frac{n}{n}\right)$

$= \infty + 6 + 1 \leq \infty$ (not prove)

Dt - 6/2/25

```
Algorithm QuickSort(A, P, rc)
//A is an array which consist
 of list of elements and lower bound is p and
 upper bound is rc.
Input:- Array A
Output:- sorting of element
BEGIN
1. If p<rc
2. q = partition(A, P, rc)
3. QuickSort(A, P, q-1)
4. QuickSort(A, q+1, rc)
END
```

Algorithm partition (A, p, r)
//
Input:- array
Output:- sorting of elements

```
BEGIN                                   —
1. x ← A[r]                             — 1
2. i ← p-1                              — 1
3. for j ← p to r-1                     — r-1-p+1 = r-p
4.     if A[j] < x                      — r-p-1
5.         i ← i+1                      — r-p-1
6.         exchange A[i] ↔ A[j]         — r-p-1
7. exchange A[i+1] ↔ A[r]               — 1
8. Return i+1                           — 1
END
```

Total = $2+r-p+r-p-1+r-p-1+r-p-1+1+1$

$= 3+4r-4p-3+1$

$= 4r-4p+1 = O(r)$

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

$x=4$, $i=0$, step = 1, $j=1$, $i=1$ — No exchange

step-2: $j=2$, $i=1$

step-3: $j=3$, $i=1$

step-4: $j=4$, $i=1$, $i=2$ — exchange 1 & 8

2	1	7	8	3	5	6	4

step-5: $j=5$, $i=2$, $i=3$ — exchange 7 with 3

2	1	3	8	7	5	6	4

step-6: $j=6$, $i=3$

step-7: $j=7$, $i=3$ — exchange 4 with 8

2	1	3	4 (pivot)	7	5	6	8

Total complexity

Worst case:- Either one of the sublist is zero.

$T(n) = T(n-1) + T(0) + O(n)$

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

$= T(n-1) + cn$

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

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

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

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

Average case:-

The no. of elements is equally divided

The pivot element will divide sublist in two equal no. of elements

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

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

$a=2, b=2, F(n)=n$

$F(n) = n^{\log_2 2} = n$

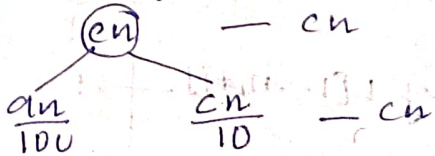
$T(n) = O(n^{\log_2 2} \log n) = O(n \log n)$

Best case:-

90% on oneside and 10% on otherside

$T(n) = T\left(\frac{90n}{100}\right) + T\left(\frac{10n}{100}\right) + cn$

$= T\left(\frac{9n}{10}\right) + T\left(\frac{1n}{10}\right) + cn$



— cn

— cn

Height of the tree:-

$n \to \frac{9n}{10} \to \frac{9^2 n}{10^2} \dots = 1$

$\left(\frac{9}{10}\right)^i n = 1$

$\Rightarrow i = \log_{10/9} n$

$cn + cn + cn + \ldots \log_{10/9} n$ times

$= O(n \log_{10/9} n)$

If in all elements are same $\rightarrow$ worst case.

$\downarrow$ same element are exchanging in repeat.

$O(n^2)$

Dt- 11/2/25

Algorithm mergesort(A, p, r)
// A is an array where p is lower bound and r is upper bound.
BEGIN
1. if p < r
2. q = (p+r)/2
3. mergesort(A, p, q)
4. mergesort(A, q+1, r)
5. merge(A, p, q, r)
END.

1	1
1	1
1	(p+r)/2
1	(p+r)/2
1	r^2

$\therefore p + 2r + 2 = O(r)$

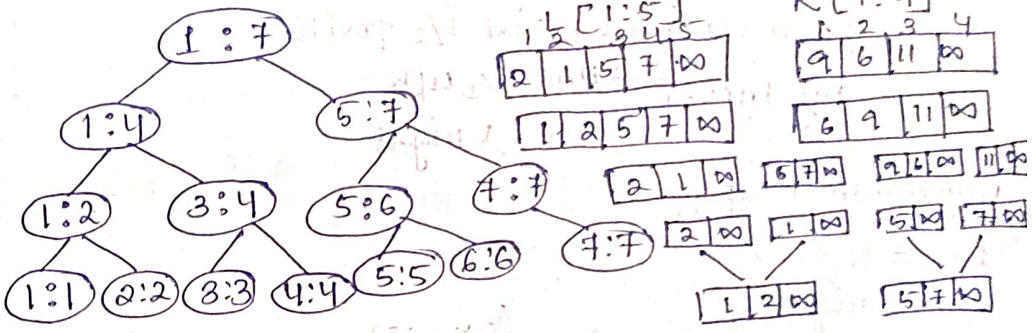
Algorithm merge(A, p, q, r)
//
BEGIN
1. $n_1 = q - p + 1$
2. $n_2 = r - q$
3. create an array $L[1 \ldots n_1+1]$ and $R[1 \ldots n_2+1]$
4. for $i = 1$ to n_1
5. $L[i] = A[p+i-1]$
6. for $j = 1$ to n_2

1	1
1	1
1	
n_1+1	
n_1	
n_2+1	

7. $R[j] = A[q+j]$
8. $L[n_1+1] = \infty$. $R[n_2+1] = \infty$

1	2	3	4	5	6	7
2	1	5	7	9	6	11

p=1 r=7



9. $i \leftarrow 1$
10. $j \leftarrow 1$
11. for $k = p$ to r
12. if $L[i] \leq R[j]$
13. $A[k] = L[i]$
14. $i \leftarrow i+1$
15. else
16. $A[k] = R[j]$
17. $j \leftarrow j+1$
18. for end
19. END

1	p
1	1
1	r-p+1
1	r-p
1	r-p
1	r-p
1	r-p
1	r-p

$6r - 6p + 3 = O(r)$

Step-1
i = 1
j = 1
k = 1

Step-2
i = 2
j = 1
k = 2

A

1	2	5	6	7	9	11

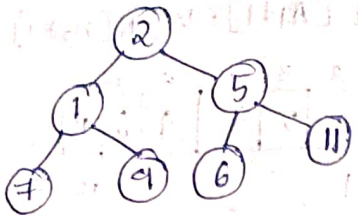
$2T(n/2) + O(n) + O(1)$

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

$F(n) = n^{\log_b a} = n^{\log_2 2} = n$

Heap Sort

1	2	3	4	5	6	7
2	1	5	7	9	6	11



Left child = $2 \times i$

Right child = $2 \times i + 1$

Element in 5th position

↳ parent will be in $i/2$ position.

In binary tree → $2n/3$ 5/2 Left
→ $2n/3$ → right

Algorithm Heapify(A, i)

1. l = left(i)
2. r = right(i)
3. If $l \leq$ heap_size(A) and A(l) > A(i)
4. then largest = l
5. else
6. largest = r
7. if $r \leq$ heap_size(A) and A(r) > A(largest)
8. largest = r
9. if largest $\neq$ i
10. exchange A[i] ↔ A[largest]
11. Max-Heapify(A, largest)

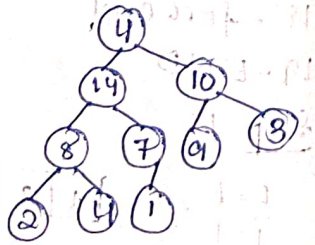
Dt - 12/2/25

Algorithm Heapsort(A)

BEGIN

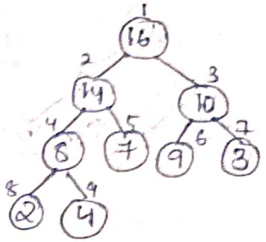
1. BUILD_MAX_HEAP(A)
2. heap_size = length(A)
3. exchange A[1] ↔ A[heap_size]
4. heap_size = heap_size - 1
5. for i = 1 to heap_size
6. Heapify(A, i)
7. for end

END



Algorithm Heapsort(A)

1. BUILD_MAX_HEAP(A)
2. for i = length(A) to 2
3. exchange A[1] ↔ A[i]
4. heap_size(A) = heap_size(A) - 1
5. Heapify(A, 1)



Algorithm BUILD_MAX_HEAP(A)

// Input :- Array

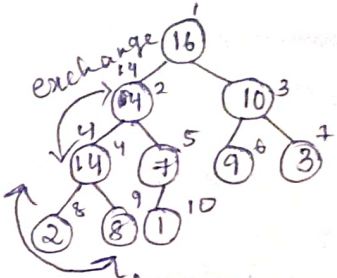
Output :- To build a max heap

BEGIN

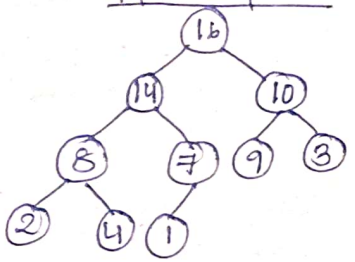
1. heapsize(A) = length(A)
2. for i = length(A)/2 to 1
3. MAX_Heapify(A, i)

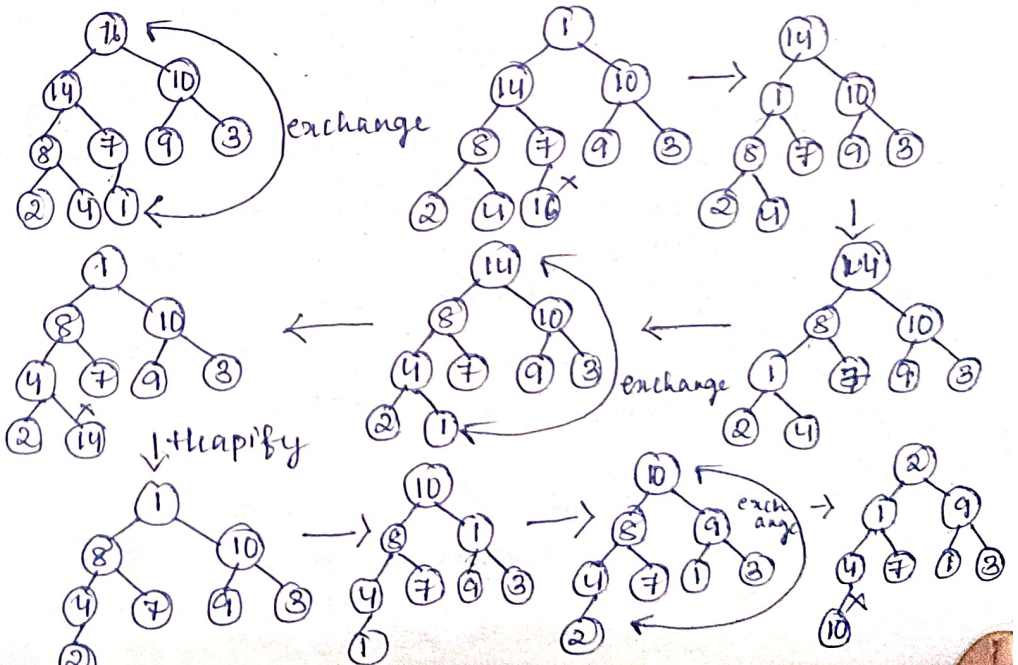
END

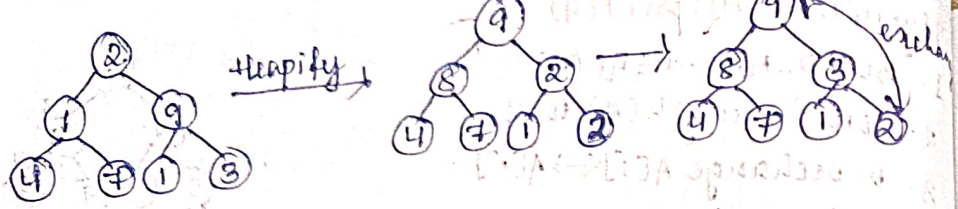
$T(n) = O(n \log n)$



MAX-heap tree







Complexity of heapsort.

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

$$= \sum_{h=0}^{\log_2 n} \frac{n}{2^{h+1}} ch$$

$$= c \sum_{h=0}^{\log_2 n} n \left(\frac{h}{2^{h+1}}\right)$$

$$= c \sum_{h=0}^{\log_2 n} n \cdot \left(\frac{h}{2^h \cdot 2}\right)$$

$$= c \sum_{h=0}^{\log_2 n} \frac{n}{2} \left(\frac{h}{2^h}\right)$$

$$= c \sum_{h=0}^{\log_2 n} \frac{n}{2} (h\, 2^{-h})$$

$$= c \sum_{h=0}^{\log_2 n} \frac{n}{2} (h 2^{-h})$$

$$= c \frac{n}{2} \left(\frac{-h}{1-(-h)}\right)$$

$$= \frac{cn}{2} \left[\frac{h}{1+h}\right]^2 = O(n)$$

$$\sum_{k=0}^{\infty} k x^k = \frac{x}{(1-x)^2}$$

DE - 13/2/25

Priority Queue:-

1) Tree should be in max heap
2) Removal of max element
3) Insertion of a new key (if the key is greater than existing element)
4) Return and remove from tree

Algorithm Maximum(A)

1. return A(1)

complexity = O(1)

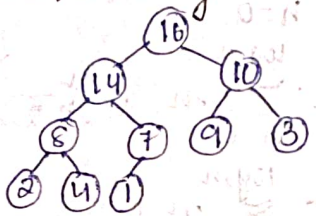
Algorithm HEAP_EXTRACT_MAX(A)

1. if heapsize(A) < 1
2. print "underflow"

3. MAX ← A[1]
4. exchange A[1] ↔ A[heapsize(A)]
5. heapsize(A) ← heapsize(A)-1
6. Heapify(A,1)
7. return max

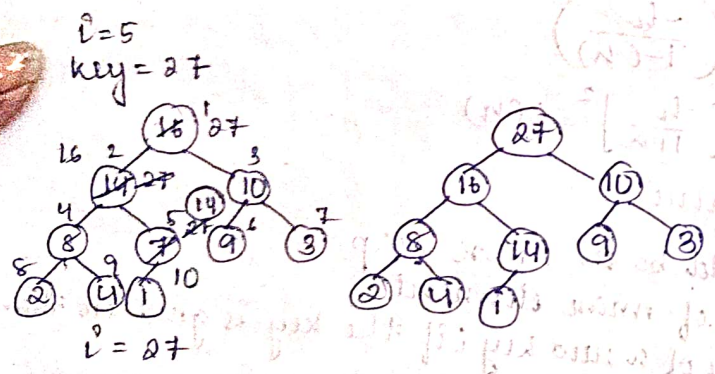
Complexity - O(log n)

Algorithm MAX_HEAP_INCREASE_KEY(A, i, key)



BEGIN
1. if key < A[i]
2. then error
3. else
4. A[i] ← key
5. while i > 1 and A[parent(i)] < A[i]
6. exchange A[i] ↔ A[parent(i)]
6. i ← parent(i)

Complexity = O(log n)



Algorithm HEAP_INSERT(A, key)
1. heap_size(A) = heap_size(A)+1
2. A[heap_size(A)] = key
3. HEAP_INCREASE_KEY(A, heapsize, key)

Complexity = O(log n)

Dt - 18/2/25

Dynamic Programming

It is a dependent programming
→ We can solve a problem independently without solving its subscription.

Cost

Matrix Multiplication:-

$[\ ]_{3\times5} \times [\ ]_{5\times4} = 3\times5\times4 = 60$ (common part should be taken once)

$[\ ]_{p\times q} \times [\ ]_{q\times r} = p\times q\times r$

$[\ ]_{n\times n} \times [\ ]_{n\times n} = n\times n\times n = n^3$

3 for loop needed for matrix multiplication common part should be in last for loop.

A_1, A_2, A_3 → 3 matrix

A_1 2×3
A_2 3×5
A_3 5×4

$(A_1(A_2A_3))$
→ 3×5×4 = 60
→ 2×3×4 = 24
] 84

$A_1 - - - i$

$i < j$ → problem is large
Non-trivial soln

LB ← i → UB

$i = j$ → Trivial solution → cost = 0

$(A_i \cdots k)(A_{k+1} \cdots j)$ k = any number b/w i & j.

Ex:- $A_1 \cdots 10$

$(A_1 \cdots 5)(A_6 \cdots 10)$

$((A_1 \cdots 3)(A_4 \cdots 5))\ ((A_6 \cdots 8)(A_9 \cdots 10))$

$$P(n) = \begin{cases} 1 & i = j \\ \sum_{k=1}^{n-1} P(k)P(n-k) & i < j \end{cases}$$

$P(3) = \sum_{k=1}^{2} P(k)P(3-k) = P(1)P(3-1) + P(2)P(3-2)$
$= 1\cdot1 + 1\cdot1 = 2$

$$P(2) = \sum_{k=1}^{1} P(k)\,P(2-k)$$

$$= P(1)\,P(1) = 1 \cdot 1 = 1$$

$$P(4) = \sum_{k=1}^{3} P(k)\,P(4-k) = P(1)\,P(3) + P(2)\,P(2) + P(3)\,P(1)$$

$$= 1 \cdot 2 + 1 \cdot 1 + 2 \cdot 1 = 5$$

$A_1\ A_2\ A_3\ A_4$

(1) $((A_1 A_2)(A_3 A_4))$ (4) $(((A_1 A_2) A_3) A_4)$

(2) $(A_1 ((A_2 A_3) A_4))$ (5) $((A_1 (A_2 A_3)) A_4)$

(3) $(A_1 (A_2 (A_3 A_4)))$

Compute optimal substructure value:-

Cost of matrix multiplication:-

$$m[i,j] = \begin{cases} 0 & i = j \\ m[i,k] + m[k+1, j] + P_{i-1} P_k P_j & i < j \end{cases}$$

A_1 (30×35)
A_2 (35×15)
A_3 (15×5)
A_4 (5×10)
A_5 (10×20)
A_6 (20×25)

i \ j	1	2	3	4	5	6
1	0	15,750				
2		0				
3			0			
4				0		
5					0	
6						0

$$m[1,2] = \min_{1 \le k < 2} m[1,1] + m[2,2] + P_0 P_k P_2$$

$$= 0 + 0 + 15,750$$

$$= 15,750$$

Dt - 19/2/25

Algorithm for chain matrix multiplication

Algorithm MATRIX-CHAIN-MULTIPLICATION(P, n)

//

1. for $i = 1$ to n
2. $\quad m[i, i] = 0$
3. for $l = 2$ to n
4. $\quad i = 1$ to $n - l + 1$
5. $\quad j = i + l - 1$
6. $\quad m[i, j] = \infty$
7. $\quad$ for $k = i$ to $j - 1$
8. $\quad\quad q = m[i, k] + m[k+1, j] + P_{i-1} P_k P_j$
9. $\quad\quad$ if $q < m[i, j]$
10. $\quad\quad\quad m[i, j] = q$, $s[i, j] = k$
11. return m and s

	P_0	P_1	P_2	P_3	P_4	P_5	P_6
A_1	30	×	35				
A_2			35	×	15		
A_3					15	×	5
A_4					5	×	10
A_5					10	×	20
A_6							20 × 25

i \ j	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2	×	0	2625	4375	7125	10,500
3	×	×	0	750	2500	5375
4	×	×	×	0	1000	3500
5	×	×	×	×	0	5000
6	×	×	×	×	×	0

$$m[1,1] + m[2,2] + P_0 P_1 P_2 + 30 \times 35 \times 15 = 15750$$

$$m[1,1] + m[3,3] + P_1 P_2 P_3 = 2625$$

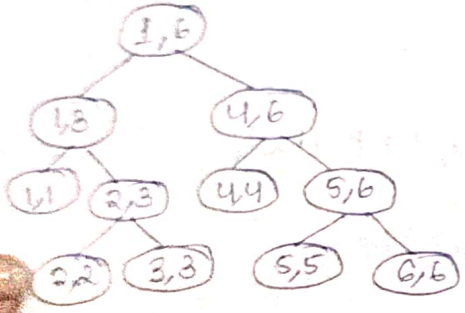
i \ j	1	2	3	4	5	6
1		1	1	3	3	3
2			2	3	3	3
3				3	3	3
4					4	5
5						5
6						

3 possible ways we can put A_1, A_6 minimum paranthesis is req.

k = no. of paranthesis

Algorithm print_optimal_parenthesis (s,i,j)

1. If i=j
2. print "Ai"
3. else print "("
4. print_optimal_parenthesis (s, i, s[i,j])
5. print_optimal_parenthesis (s, s[i,j]+1, j)
6. print ")"



((A1(A2A3))(A4(A5A6)))

Dt - 25/2/25

Knapsack problem

It is a maximisation problem.

object :- x_1 x_2 x_3 -- x_n

weight :- w_1 w_2 w_3 --- w_n

where w = weight of the knapsack

profit :- P_1 P_2 P_3 -- P_n

maximise $\sum_{i=1}^{n} x_i P_i$

$\sum_{i=1}^{n} x_i w_i \leq w$

There two types of knapsack problem on the basis types of object you are given to.

1) Zero-one knapsack (unbreakable)
2) Fractional knapsack

1) Zero-One knapsack :-

x_1	x_2	x_3	x_4	
0	0	0	0	→ 2^4 ways = 16
0	0	0	1	complexity = 2^4

To reduce complexity
i) Bit method
ii) Tabular method

Tabular Method :-

$w = 8$

	x_1	x_2	x_3	x_4
w =	2	3	4	5
P =	1	2	5	6

P	w	0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	1	1	1	1	1	1
2	3	2	0	0	1	2	2	3	3	3
5	4	3	0	0	1	2	5	5	6	7	7
6	5	4	0	0	1	2	5		6	7	(8)

$V[i,w] = \max[V[i-1,w], V[i-1, w-w_i] + P_i]$

$V[4,5] = \max[V[3,5], V[3,0]+6] = 6$

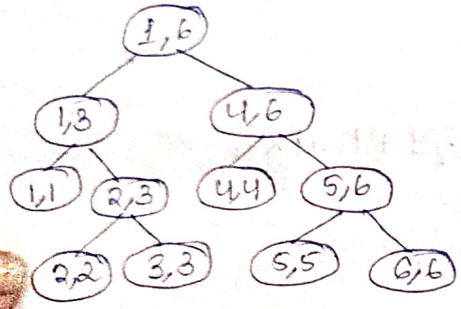
$V[4,6] = \max[V[3,6], V[3,6-5]+6] =$

$V[4,7] = \max[V[3,7], V[3,$

x_4 will be selected [illegible]
have maximum profit.

Algorithm print_optimal_parenthesis (s,i,j)

1. If i=j
2. print "Ai"
3. else print "("
4. print_optimal_parenthesis (s, i, s[i,j])
5. print_optimal_parenthesis (s, s[i,j]+1, j)
6. print ")"



((A1(A2A3))(A4(A5A6)))

Dt-25/2/25

knapsack problem

- It is a maximisation problem.

object :- x_1 x_2 x_3 -- x_n

weight :- w_1 w_2 w_3 --- w_n

where W = weight of the knapsack.

profit :- P_1 P_2 P_3 -- P_n

maximise $\sum_{i=1}^{n} x_i P_i$

$\sum_{i=1}^{n} x_i w_i \leq W$

There two types of knapsack problem on the basis types of object you are given to.

1) Zero-one knapsack (unbreakable)
2) Fractional knapsack

1) Zero-One knapsack :-

x_1	x_2	x_3	x_4
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
.	.	.	.
.	.	.	.
.	.	.	.

→ 2^4 ways = 16
complexity = 2^4

For n elements
complexity = 2^n

To reduce complexity there are two way
i) dp method
ii) Tabular method

Tabular Method :-

W = 8

	x_1	x_2	x_3	x_4
w =	2	3	4	5
P =	1	2	5	6

i \ w	0	1	2	3	4	5	6	7	8
P w 0	0	0	0	0	0	0	0	0	0
1 2 1	0	0	1	1	1	1	1	1	1
2 3 2	0	0	1	2	2	3	3	3	3
5 4 3	0	0	1	2	5	5	6	7	7
6 5 4	0	0	1	2	5		6	7	(8)

$v[i,w] = \max[v[i-1,w], v[i-1, w-w_i] + P_i]$

$v[4,5] = \max[v[3,5], v[3,0]+6] = 6$

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

$v[4,1] = \max[v[3,1], v[3,1-5]+6] = 0$

x_4 will be selected if the previous row does not have maximum profit

8-6 2 x_4 compare with x_3

2- x_2 compare with x_4

x_1	x_2	x_3	x_4	
0	1	0	1	– vector space of solution which inserted object into knapsack.

What is optimal cost of putting object into knapsack
What is vector space of soln which is inserted object into knapsack?

Sit Method:-

$P = (1, 2, 5, 6)$ $k.w = 8$ $n = 4$

$w = (2, 3, 4, 5)$

(P, w)

$S^0 = \{(0,0)\}$

$S_1^0 = \{(1,2)\}$

$S^1 = S^0 \cup S_1^0 = \{(0,0), (1,2)\}$ Add (2,3)

$S_1^1 = \{(2,3), (3,5)\}$

$S^2 = S^1 \cup S_1^1 = \{(0,0), (1,2), (2,3), (3,5)\}$

$S_1^2 = \{(5,4), (6,5), (7,7), (8,9)\}$

profit is increasing, weight is decreasing

$S^2 \cup S_1^2 = S^3 = \{(0,0), (1,2), (2,3), (3,5), (5,4), (6,6), (7,7), (8,9)\}$

weight is more, less profit is considered

$S^3 = \{(0,0), (1,2), (2,3), (3,5), (6,6), (7,7)\}$

$S_1^3 = \{(6,5), (7,7), (8,8), (9,10), (12,11), (13,12)\}$

$S^4 = S^3 \cup S^3 = \{(0,0), (1,2), (2,3), (3,5), (6,6), (7,7), (6,5), (8,8)\}$

Selection criteria for object:-

x_1	x_2	x_3	x_4
0	1	0	1

$(8,8) \in S^4$ but $(8,8) \notin S^3$

$(8,8) - (6,5) = (2,3) \in S^3$ but $(2,3) \notin S^2$

$(2,3) \in S^2$ but $(2,3) \notin S^1$ so, $(2,3) - (2,3) = (0,0) \in S^1$ and $(0,0) \in S^0$

Dt - 18/3/25

Longest common subsequence

$X = \{A, C, D, B, E\}$

$Y = \{F, C, B, E\}$

$LCS(X, Y)$

(1) Subproblem

$X = \{x_1, x_2, x_3 \dots x_n\}$

$Y = \{y_1, y_2, y_3 \dots y_m\}$

where $n \neq m$

$\rightarrow n \geq m$

(1) If $x_n = y_m$, $LCS(x_{n-1}, y_{m-1}) + 1$

(2) If $x_n \neq y_m$, $\max(LCS(x_{n-1}, y_m), LCS(x_n, y_{m-1}))$

(2) Optimal cost

$$c(i,j) = \begin{cases} 0, & i = 0 \text{ or } j = 0 \\ c(i-1, j-1) + 1 & i, j > 0, x_i = y_i \\ \max(c(i-1, j), c(i, j-1)) & i, j > 0 \\ & x_i \neq y_i \end{cases}$$

$X = \{A, B, C, B, D, A, B\}$

$Y = \{B, D, C, A, B, A\}$

Algorithm lcs_length(X, Y)

1. m = length(X)
2. n = length(Y)
3. for i = 1 to m
4. c[i, 0] ← 0
5. for j ← 0 to n
6. c[0, j] = 0
7. for i ← 1 to m
8. for j ← 1 to n

9. if $x_i = y_i$

10. $c[i,j] \leftarrow c[i-1,j-1] + 1$

11. $b[i,j] = "\nwarrow"$

12. else if $c[i-1,j] \geq c[i,j-1]$

13. $c[i,j] \leftarrow c[i-1,j]$

14. $b[i,j] = "\uparrow"$

15. else

16. $c[i,j] \leftarrow c[i,j-1]$

17. $b[i,j] \leftarrow "\leftarrow"$

18. return c & b.

i↓ / j→		0	1 (B)	2 (D)	3 (C)	4 (A)	5 (B)	6 (A)
	0	0	0	0	0	0	0	0
A	1	0	0↑	0↑	0↑	1↖	1←	1↖
B	2	0	1↖	1←	1←	1↑	2↖	2←
C	3	0	1↑	1↑	2↖	←2	2↑	2↑
B	4	0	1↖	1↑	2↑	2↑	3↖	3←
D	5	0	1↑	2↖	2↑	2↑	3↑	3↑
A	6	0	1↑	2↑	2↑	3↖	3↑	4↖
B	7	0	1↖	2↑	2↑	3↑	4↖	4↑

$i = 1, j = 3$

$x_1 = y_3$ - false.

elsif $c[0,3] \geq c[1,2]$

$c[1,3] \leftarrow c[0,3]$

$j = 4$

$x_1 = y_4$

$c[1,4] \leftarrow c[0,3] + 1$

$j = 5$

$c[0,5] \geq c[1,4]$

$0 \geq 1$

$c[1,5] \leftarrow c[1,4]$

$j = 3$

$c[1,3] \geq c[2,2]$

$0 \geq 0$

$c[2,3] \leftarrow c[1,3]$

$m = 7, n = 6$

$i = 2$

$j = 1$

$x_2 = y_1$

$c[2,1] \leftarrow c[1,0] + 1$

$= 0 + 1$

$j = 2$

$c[1,2] \geq c[2,1] \Rightarrow 0 \geq 1$ - false

else $c[1,2] \leftarrow c[1,1]$

Sequence - <BCBA>

Length = 4

Dt-19/8/25

Greedy approach

Algorithm Greedy(a,n)
// a is an array contain n inputs
BEGIN
1. solution = ϕ
2. x = select(a)
3. if fessible (solution, x)
4. solution = solution $\cup$ x
or
union (solution, x)
5. Return (solution)
END

Fractional knapsack

We are given n objects and a knapsack. A object i has a weight w_i & knapsack has a capacity m if a fraction x_i, whose value in from 0 to 1 $0 \leq x_i \leq 1$ of object i is place into the knapsack a profit of $P_i \times x_i$ is earned.
The objective to obtain filling of knapsack that maximise total profit earn.
Maximize $\Sigma P_i x_i$ $1 \leq i \leq n$
subject to condition $\Sigma w_i x_i \leq m$ and $0 \leq x_i \leq 1$, $1 \leq i \leq n$

8) Consider the following instance of knapsack problem.
$n = 3$
$m = 20$
$(w_1\ w_2\ w_3) = (18, 15, 10)$
$(P_1\ P_2\ P_3) = (25, 24, 15)$

(*) Arrange according to profit

x_1	x_2	x_3	$w_i x_i$	$P_i x_i$
1. 1	2/15	0	20	25 + 2/15 + 24 = 28.2
2. 0	10/15	1	20	10/15 × 24 + 15 = 31
3. 1/2	1/3	1/4	16.5	1/2 × 25 + 1/3 × 24 + 1/4 × 15
4. 0	1	5/10	20	1 × 24 + 5/10 × 15 = 31.5
5. 5/18	1	0	20	5/18 × 25 + 1 × 24

(1) After x_1 consider next profit $P_2 = 24$ remaining weight of knapsack = 20 - 18 = 2 so we will take 2/15 = x_2 $x_3 = 0$

(2) Consider less weight i.e. 10 which is of 3rd object remaining weight 20 - 10 = 10 consider next less weight i.e. 15 = 10/15

(3) $25 + \frac{2}{15} \times 24 = 28.2$

Final max profit = 31.5

x_1	x_2	x_3
0	1	2

Profit to weight ration

$\frac{P_1}{w_1} = \frac{25}{18}$

$\frac{P_2}{w_2} = \frac{24}{15}$

$\frac{P_3}{w_2} = \frac{15}{10}$

Arrange in increasing order

Algorithm GreedyKnapsack(m,n)
// P[1--n] and w[1--n] contain profit and
// weights respectively of n object ordered such
// that P[i]/w[i] ≥ P[i+1]/w[i+1] is the
// knapsack size and x[1--n] is soln vector.
BEGIN
1. for i=1 to n
2. x[i] = 0.0
3. U = m-1
4. for i=1 to n
5. {
6. if w[i] > U then break;
7. x[i] = 1.0 ; U = U - w[i]
8. }
9. if (i ≤ n) then x[i] = U/w[i]
END

$w_1\ w_2\ w_3 = (18, 15, 10)$

$P_1\ P_2\ P_3 = (25, 24, 15)$

$\frac{P_1}{w_1} = 1.38$ $\frac{P_3}{w_3} = 1.5$

$\frac{P_2}{w_2} = 1.6$

① $x_1\ x_2\ x_3$
0 0 0

$U = m = 20$

② $i = 1$ to $n = 3$

④ if $18 > 20$

$x[1] = 1$ $U = 20 - 15 = 5$

$i = 2$

if $10 > 5$ then $x[2] = 1$

8. if $2 \leq 3$ then $x[2] = 1/2$

$x_1\ x_2\ x_3$
0 1 0
1 5/10

order according to ratio

$w_1 = 15$	$P_1 = 24$
$w_2 = 10$	$P_2 = 15$
$w_3 = 18$	$P_3 = 25$

← this order will be supplied to algorithm

$x_1\ x_2\ x_3$
0 1 5/10

Dt - 20/3/25

Hoffman coding
→ variable code length
→ Fixed code length

(*) a:5 b:15 d:18 f:6
Fixed length for each character = 3
Size of = (5×3 + 15×3 + 18×3 + 6×3) bits

Variable length
→ character present more time take less bits
→ " " less " " more "

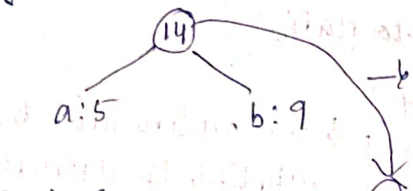
a:5 b:15 d:18 f:6
4 2 1 3
$= 1\times18 + 15\times2 + 6\times3 + 5\times4$

(*) If we go for variable length code number of bits stored is less than fixed length code.

(*) a:5 b:9 c:12 d:13 e:16 f:45

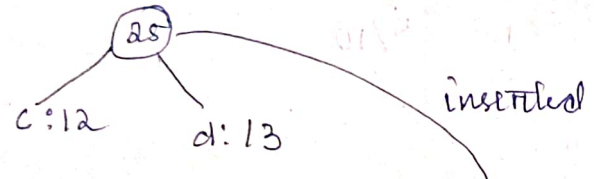
Arrange in increasing order of their value construct binary tree

(i)



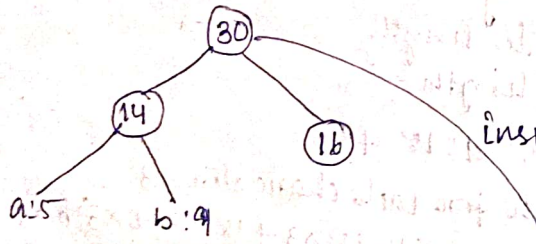
a:5 b:9 c:12 d:13 (14) e:16 f:45

(ii)



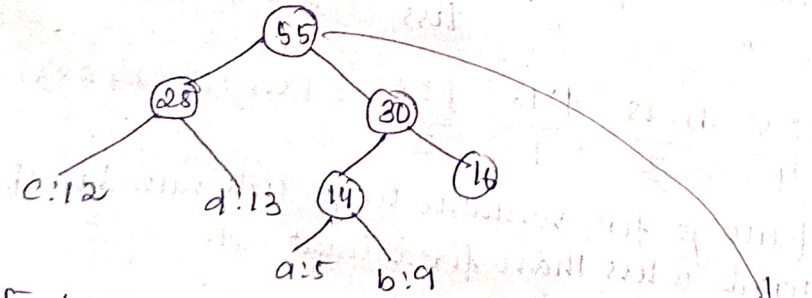
a:5 b:9 c:12 d:13 14 e:16 (25) f:45

(iii)



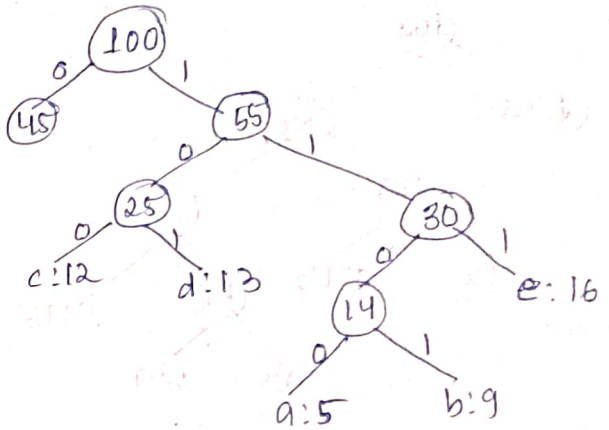
a:5 b:9 c:12 d:13 14 e:16 25 (30) f:45

(iv) 25, 30



a:5 b:9 c:12 d:13 14 e:16 25 30 f:45 (55)

(v)



From each node move left 0 & move right 1

Code length of

character	No. of bits	bit value
f	1 bit	0
c	3 bits	100
d	3 bits	101
a	4 bits	1100
b	4 bits	1101
e	3 bits	111

Algorithm HOFFMAN(C)

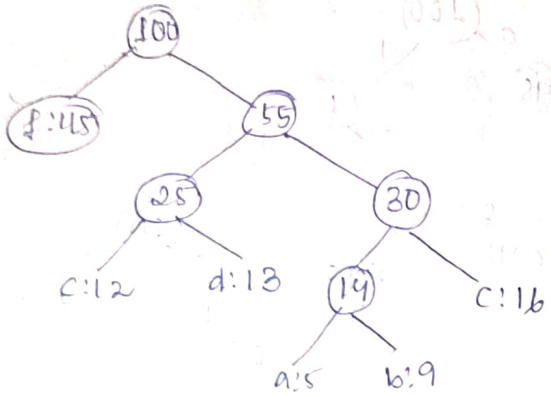
C - a set consist of all character.

BEGIN

1. $n \leftarrow |C|$
2. $Q \leftarrow C$
3. for $i \leftarrow 1$ to $n-1$
4. allocate a new node z
5. left(z) $\leftarrow x \leftarrow$ EXTRACT_MIN(Q)
6. right(z) $\leftarrow y \leftarrow$ EXTRACT_MIN(Q)
7. $f(z) = f(x) + f(y)$
8. INSERT(Q, z)
9. return EXTRACT_MIN(Q)

$B(T) = \sum \text{depth}(c)$

↓ no. of bits char $\in C$



Active Selection

(si fi) (sj fj)

si, sj → start time of i & j

fi fj → finish time of i & j.

Objective - To use some resources to do different job without overlapping each other.

$s_j \geq f_j$

or

$s_i \geq f_j$

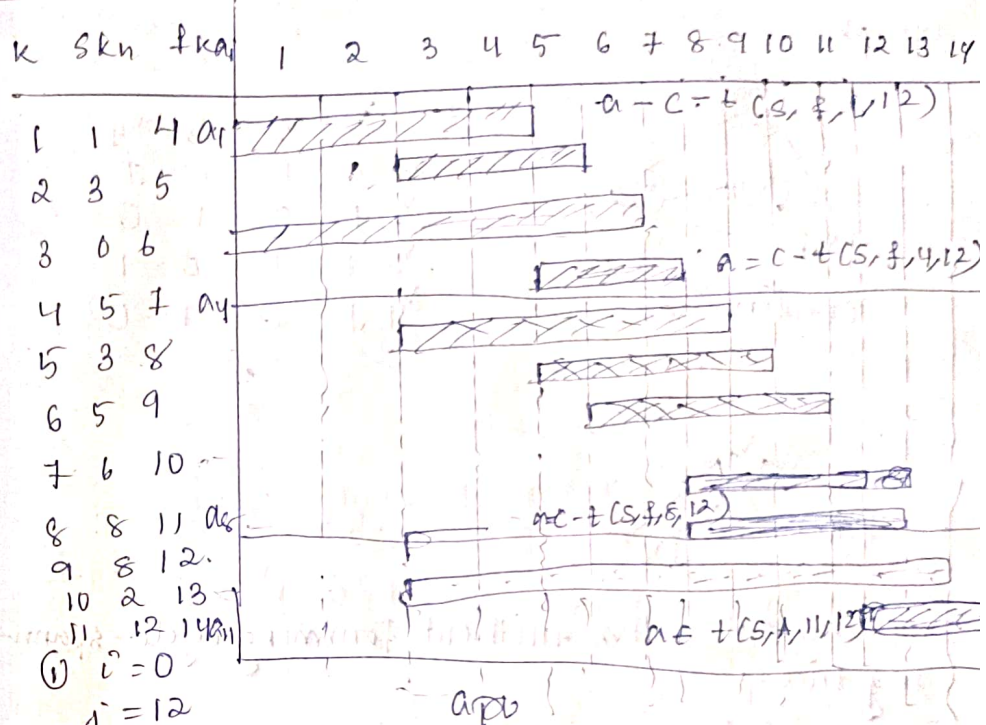
i	1	2	3	4	5	6	7	8	9	10	11
si	1	3	0	5	3	5	6	8	8	2	12
fi	4	5	6	7	8	9	10	11	12	13	14

↳ arrange in ↑se order.

Algorithm.

Algorithm Recursive_active_selection(s, f, i, j)

1. m ← i+1
2. while m<j and sm<fi
3. m ← m+1
4. if m<j
5. return {am} ∪ Recursive_active_selection(s,f,m,j)
6. else return ∅



① i = 0
j = 12
m ← 1
1 < 12 & 1 < 0
if 1 < 12
return {a0} ∪ Recur (s, f, 1, 12)

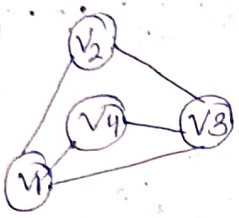
② i = 1, j = 12
m = 2
2 < 12 3 < 4
if m = 3
4
3 < 12 0 < 4
m = 4
4 < 12 5 < 4
4 < 12
return {a1} ∪ (s, f, 4, 12)

③ i = 4, j = 12
m = 5
5 < 12 & 3 < 7
m = 6
6 < 12 & 5 < 7
7 < 12 & 6 < 7
8 < 12 & 8 < 7
8 < 12
return {a8} ∪ (s, f, 8, 12)

Time complexity = O(n) a1 ∪ a4 ∪ a8 ∪ a11 ∪

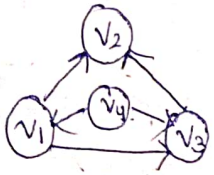
n = no. of jobs.

Graph



(undirected)

	v_1	v_2	v_3	v_4
v_1	0	1	1	1
v_2	1	0	1	0
v_3	1	1	0	1
v_4	1	0	1	0



Visiting all vertex without forming cycle - spanning tree.

Aim → minimum cost spanning tree

i) Kruscal algorithm → min-cost edge are connected and no. cycle.

ii) Prime algorithm

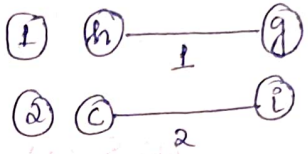
04-05/3/22

MST_Krushal (G, w)

1. $A = \phi$
2. for each vertex $v \in G.V$
3. MAKE_SET(v)
4. sort the edges of $G.E$ into increasing order by weight w.
5. for each edge $(u, v) \in G.E$ taken

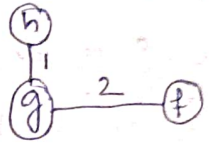
$\{(h,g), (c,i), (g,f), (c,f), (a,b), (i,g), (h,i), (c,d), (b,c), (a,h), (d,e), (f,e), (b,h), (d,f)\}$

1.

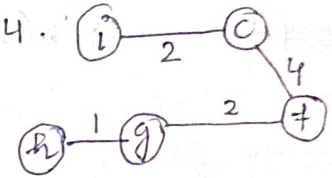


2.

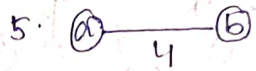
3.



4.



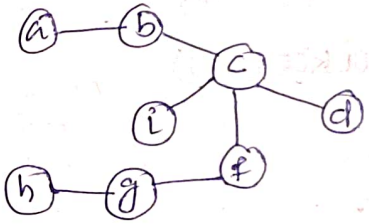
5.



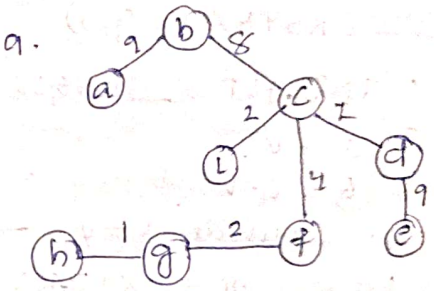
6. $d \neq \phi$

7. $f \neq \phi$

8.



9.



37 → minimum cost.

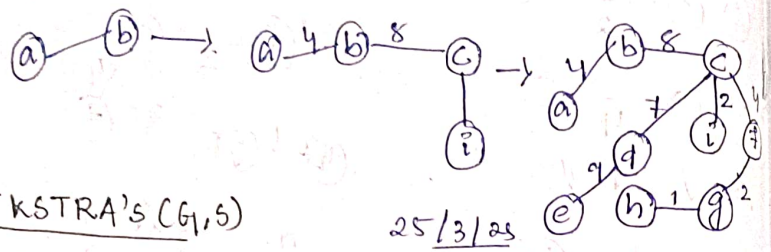
Write the condn for edge is safe for the spanning tree free u, v are element of graph $G(U,V) \in G$ if find(u) ≠ find(v)

8 → min heap

MST_PRIM (G, w, r)

1. for each $u \in G.V$
2. $u.key = \infty$
3. $u.\pi = NIL$
4. $r.key = 0$
5. $Q = G.V$
6. while $Q \neq \phi$
7. $u = EXTRACT\_MIN(Q)$
8. for each $v \in G.adj(u)$
9. if $v \in Q$ and $w(u,v) < v.key$.
10. $v.\pi = u$

key	a	b	c	d	e	f	g	h	i
d	0	∞	∞	∞	∞	∞	∞	∞	∞
π	N	N	N	N	N	N	N	N	N
d	0	4	8	7	10	4	6	11	2
π	N	a	b	c	d	c	f	g	c



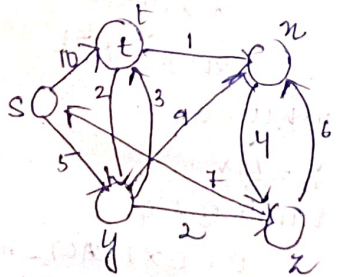
DIJKSTRA'S (G, S) 25/3/25

1. INTIALIXE_SINGLE_SOURCE (G, S)
2. S = ϕ
3. Q = G.V
4. while Q = ϕ — V
5. u = EXTRACT_MIN — log V
6. S = S ∪ {u}
7. for each vertex V ∈ G.adj(u) — e
8. RELAX (U, V, w)

V(log V + e)

INITIALIXE_SINGLE_SERVICE (G, S)
1. for each V ∈ G.V
2. V.d = ∞
3. V.π = NIL
4. S.d = 0

RELAX (U, V, w)
1. if V.d > U.d + w (U, V)
2. V.d = U.d + w(U+V)
3. V.π = U



1. s = s
adj(s) = {t, y}
v = t
w = 10
v = y
w = 5

	s	t	x	y	z
d	0	~~10~~	∞	~~5~~	∞
π	N	~~N~~ s	N	~~N~~ s	N

2. s = y
adj(y) = {t, x, z)
v = t
w = 5+3 = 8
v = x
w = 5+9 = 14
v = z
w = 5+2 = 7

	t	x	z
y	~~10~~ 8	~~14~~	~~7~~
s	s y	~~y~~	~~y~~

3. s = z
adj(z) = {x, s)
v = s
w ! > 0
v = x
w = 7+6 = 13

	t	x	z
z			
7	8	~~13~~	
y	y	y z	

4. s = t
adj(t) = {y, x}
v = y
no update
v = x
w = 9

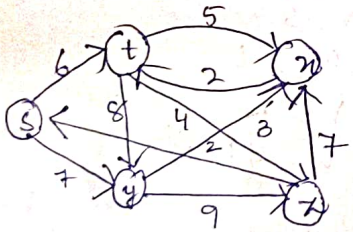
	x
t	
s	13 9
y	~~t~~

5. v = x
w = 9

x
9
t

s ⟶ x
total cost :- = 7 s → y → x
total cost s to x = 9 s → y → t → x

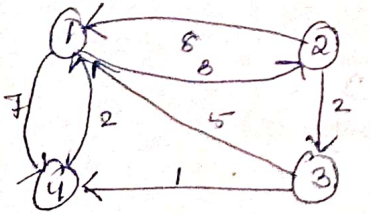
```
BELLMAN_FORD(G,w,s)
1. INITIALIZE_SINGLE_SOURCE(G,s)
2. for i=1 to |G.V|-1
3.     for each edge (u,v) ∈ G.E
4.         RELAX(u,v,w)
5. for each edge (u,v) ∈ G.E
6.     if v.d > u.d + w(u,v)
7.         return FALSE
8. return true.

INITIALIZE_SINGLE_SOURCE(G,s)
1. for each v ∈ G.V
2.     v.d = ∞
3. s.d = 0

RELAX(u,v,w)
1. if v.d > u.d + w(u,v)
2.     v.d = u.d + w(u,v)
3.     v.π = u
```



i = 1

s	t	x	y	z
0	~~6~~	~~7~~	~~11~~	~~2~~
IN	~~s~~	~~t~~	~~s~~	~~t~~

i = 1



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

A^1 = considering ① as a intermediate vertex

$A^1(2,3) = A^0(2,1) + A^0(1,3)$ As $2 < \infty$

$= 8 + \infty$ $A^1(2,3) = 2$

$= \infty$

$A(3,2) = A^0(3,1) + A^0(1,2)$

$= 5 + 3 = 8$

$A(3,4) = (3,1) + (1,4)$

$= 5 + 1 = 6$

$A^1(2,4) = A^0(2,1) + A^0(1,4) = 8 + 7 = 15$

$A^1(4,3) = (4,1) + (1,3) = 2 + \infty = \infty$

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

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

$A(1,3) = (1,2) + (2,3)$

$= 3 + 2 = 5$

$A(1,4) = (1,2) + (2,4)$

$= 3 + 15 = 18$

$A(3,1) = (3,2) + (2,3)$

$= 8 + 8$

$= 16$

$A(3,4) = (3,2) + (2,4)$

$A(4,3) = (4,2) + (2,3)$

$= 5 + 2 = 7$

$A(4,1) = (4,2) + (2,1)$

$= 5 + 8 = 13$

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

$A^3(1,2) = A^2(1,3) + A^2(3,2) = 5+8 = 13$

$A^3(1,4) = A^2(1,3) + A^2(3,4) = 5+1 = 6$

$A^3(2,1) = A^2(2,3) + A^2(3,1) = 2+5 = 7$

$A^3(2,4) = A^2(2,3) + A^2(3,4) = 2+1 = 3$

$A^3(4,1) = A^2(4,3) + A^2(3,1) = 7+5 = 12$

$A^3(4,2) = A^2(4,3) + A^2(3,2) = 7+8 = 15$

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

$A^4(1,2) = A^3(1,4) + A^3(4,2) = 6+5 = 11 > 3 = 3$ - Ans

$A^4(1,3) = A^3(1,4) + A^3(4,3) = 6+7 = 13$

$A^4(2,1) = A^3(2,4) + A^3(4,1) = 3+2 = 5$

$A^4(2,3) = A^3(2,4) + A^3(4,3) = 3+7 = 10$

$A^4(3,1) = A^3(3,4) + A^3(4,1) = 1+2 = 3$

$A^4(3,2) = A^3(3,4) + A^3(4,2) = 1+5 = 6$

$A^k(i,j) = \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\}$

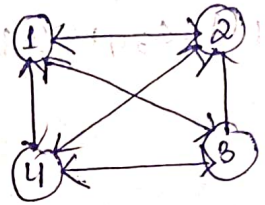
```
for(k=1; k<=n; k++){
    for(j=1; j<n; j++){
        A^k[i][j] = min{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)}}
```

$A^k(i,j) = \min\{A^{k-1}(i,j), A^{k-1}(i,k) + A^{k-1}(k,j)\}$

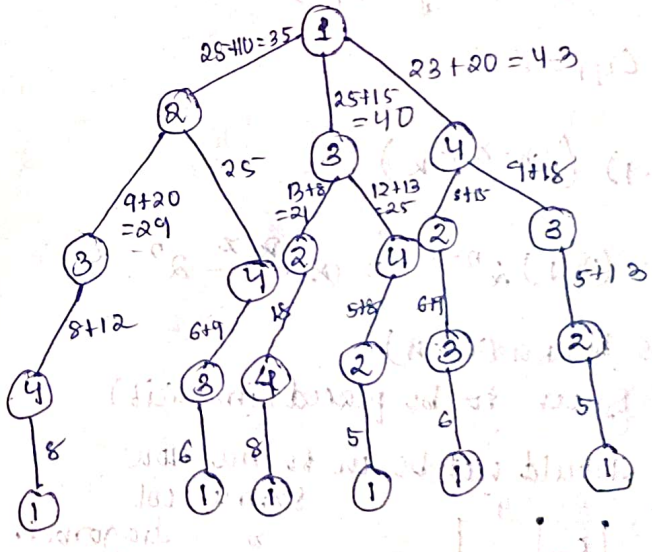
```
for(k=1; k<=n; k++){
    for(i=1; i<=n; i++){
        for(j=1; j<=n; j++){
            A^k[i][j] = min{}}}}
```

2/4/25

Travel Salesperson Problem (Imp)



$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix} \end{array}$$



$g(1, \{2,3,4\})$

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

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

$g(1,\{2,3,4\})$

$= \min\{c_{12} + g(2,\{3,4\}), c_{13} + g(3,\{2,4\}),$
$c_{14} + g(4,\{2,3\})\}$

$= \min\{10 +$

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

$= \min\{c_{23} + g(3,\{4\}) + c_{24} + g(4,\{3\})\}$

$g(3,\{2,4\}) = \min\{c_{32} + g(2,\{4\}), c_{34} + g(4,\{2\})\}$

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

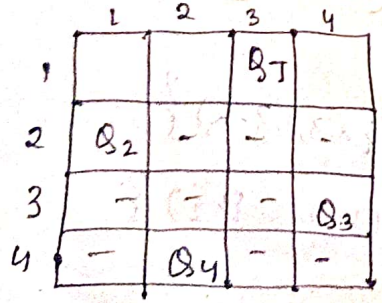
$= \min\{c_{34} + g(4,\phi)\}$

$g(4,\phi) = c_{41} = 8$

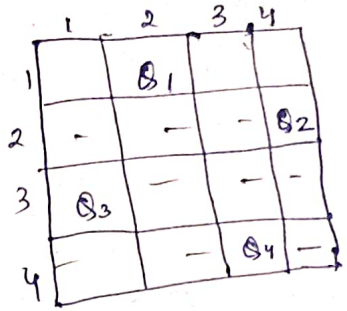
$$\sum_{k=0}^{n-2} (n-1)\,(n-k\,c_k)$$

$= (n-1)\,2^{n-k} = n\,2^{n-k} - 2^{n-2}$

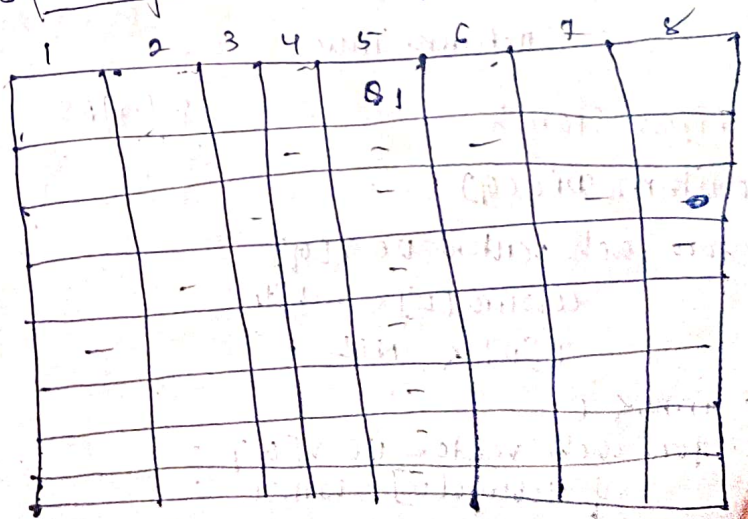
1. 4×4 chess board (n×n)
2. n no. of Queen to be placed (implicit)
3. 2 Queen should not be in same row
   same col
   & diagonal



$x = \langle 3,1,4,2 \rangle$



$x = \langle 2,4,1,3 \rangle$



```
1. for i=1 to n
2. {
3.     if place(k,i) then
4.       {
5.          x[k]=i
6.          if (k==n) then write (x[1:n])
7.          else NQueens(k+1,x)
8.          }
9.          }
```

Algorithm Place(k,i)

```
1. for j=1 to k-1
2. {
3.     if ((x[j]=i) or abs(x[j]-i)
4.                 =abs(j-k))
           then return False
5.  else
6.       return True
```

Depth First Search 3/4/25

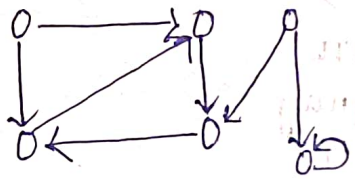
1) Algorithm DFS(G)

```
1. for each vertex u∈V[G]
2.          colour[u] ← white
3.          π[u] ← NIL
4. time ← 0
5. for each vertex u∈V[G]
6.      if colour[u] = WHITE
7.          DFS_VISIT(u)
```

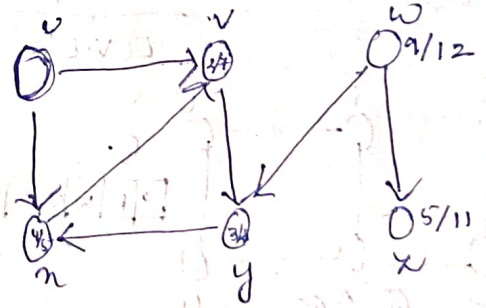
Algorithm DFS_visit(x)

```
1. colour[x] ← GRAY
2. time ← time+1
3. d[u] ← time
4. for each v∈adj(x)
5.  if colour[v]=WHITE
6.          π[v] ← u
7.          DFS_VIST
8. colour[u] ← BLACK
9. f[u] ← time ← time+1
```

$O(n^2)$
n = no. of vertices



v → u v w x y z
π(u) → N ~~v~~ N ~~y~~ ~~v~~ ~~w~~



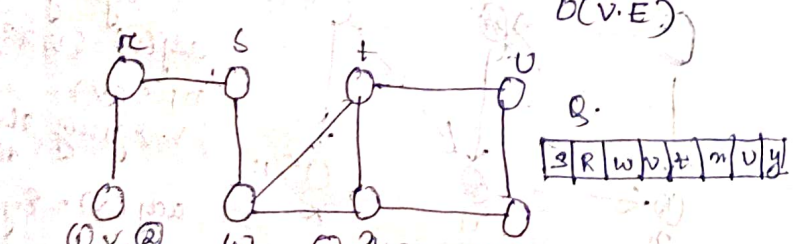
time = 1+0 = 1
u → grey
adj(u) = {v, x}
DFS(v)
v → grey π(v) = u
time = 2
adj(v) = {y}
π(y) = v time = 3
adj{y} = {x}
x → grey
time = 4
adj(x) = {v}

Breadth First Search

Algorithm BFS(G,s)

1. for each vertex $u \in V[G] - \{s\}$
2. colour[u] ← w
3. d[u] ← ∞
4. π[r] ← NIL
5. colour[s] ← GRAY
6. d[s] = 0
7. π[s] ← NIL
8. ENQUEUE(Q,s)
9. while Q ≠ ϕ
10. u ← DEQUEUE(Q)
11. for each v ∈ adj(r)
12. if colour[v] = WHITE
13. colour[v] = Gray
14. d[v] = d[u] + 1
15. π[v] ← u
16. ENQUEUE(Q,v)
17. colour[u] ← Black

$O(V+E)$



	① v B	② B	③ B	④ B	⑤ B	⑥ B	y ⑦ B	⑧ B
	s	u	v	t	w	x	u	y
	N	N	N	N	N	N	N	N
distance		1	2	2	1	2	3	3
parent		s	π	w	s	w	t	x

adj(w) = {s, t, x}

t → colour + 1

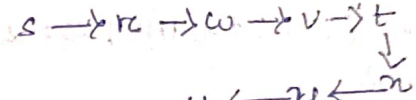
= 1 + 1

adj(v) = {x}

adj(t) = {v, w, x}

adj(x) = {t, y}

adj(u) = {t, y}



8/4/25

Robin_karp_matches(T, p, d, q)

1. n ← length(T)
2. m ← length(P)
3. h ← $d^{m-1} \bmod q$
4. p = 0
5. to = 0
6. for i = 1 to m
7. p = (dp + p[i]) mod q
8. to = (dto + T[i]) mod q
9. for s ← 0 to n-1
10. if p = ts
11. if p[1..m] = T[s+1..s]
12. print[pattern occurs after s shift]
13. if s < n-m
14. $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$

$O(n-m)$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	2	3	5	9	0	2	3	1	4	1	5	2	6	7	5	9	9	2	1

	1	2	3	4	5	6
P	3	1	4	1	5	2

Do consider q as 13

i=1

p = (10×0 + 3) mod 13 = 3

t0 = (10×0+2) mod 13 = 2

i = 2

p = (10×3 + 1) mod 13 = 5

t0 = (10×2 + 3) mod 13 = 10

i = 3

p = (10×5 + 4) mod 13 = 2

t0 = (10×10 + 5) mod 13 = 1

i = 4

p = (10×2 + 1) mod 13 = 8

t0 = (10×1 + 9) mod 13 = 6

i = 5

p = (10×8 + 5) % 13 = 7

t0 = (10×6 + 2) % 13 = 8

i = 6

p = (10×7 + 2) % 13 = 7

t0 = (10×8 + 2) % 13 = 4

s = 0

0 < 19 - 6 = 13

t1 = (10(4 - 2×4) + 3 = 2

t2 = (10(2 - 3×4) + 1) mod 13

Boyer-Moore string matching

T = A I N A I S E S T I X A I N A I N E N n = 19
(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18)

P = A I N A I N E N m = 8
(0 1 2 3 4 5 6 7)

scan input right to left

value = length of pattern - i - 1

Bad match table

A	I	E	N	*
4	3	1	8	8

value(e) = 8 - 6 - 1 = 1

value(I) = 8 - 4 - 1 = 3

value(A) = 8 - 3 - 1 = 4

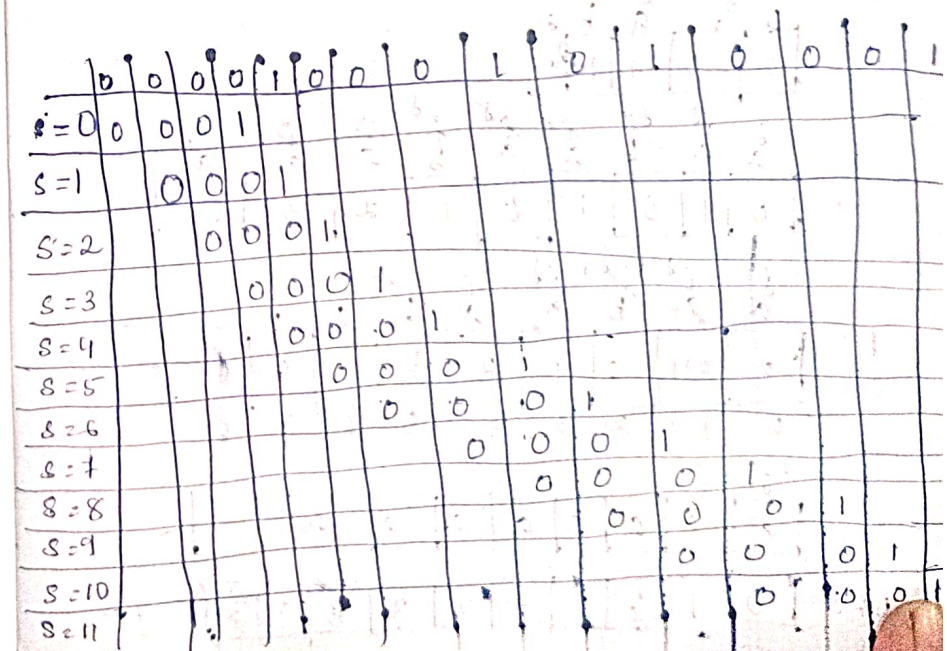
First shift 8 shift required

second » 3 » s required.

Naive string matching

1. m = length(T)
2. n = Length(P)
3. for s ← 0 to m-n
4. if P[1 - - n] = T[s+1 - - s+n]
5. print ("matched after s shifts")

O(m-n)



We have to process = 15-4 = 11 shift

string is match after shift 1
string is match after shift 5
string is match after shift 10.

Complexity = O(n-m)

→ problem which cannot be solvable is called non deterministic problem.

Dt - 10/11/25

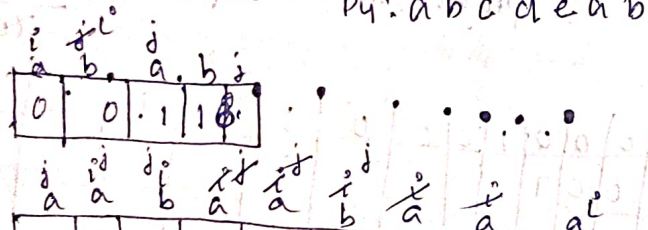
NP
P
NPC
NP Hard

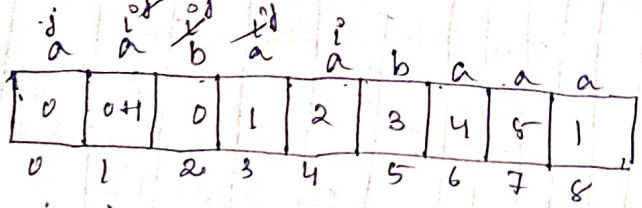
KNP algorithm

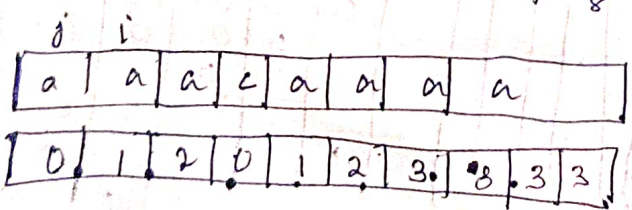
a a b c d

P	S
a	d
a a	c d
a a b	b c d
a a b c	a b c d

P1: a b a b
P2: a b c d a b c
P3: a b c d a b c a b f
P4: a b c d e a b f a b c







→ if i & j are same then both will increment
→ if i & j are not same increase i

```
LPS[0] = 0
length = 0
i = 1
while(i<m)
  { if P[i] == P[length]
      { length++
        LPS[i] = length
        i++ }
    else { if (length != 0){
             length = LPS[length-1] }

    else { LPS[i] = 0
           i++; }}
```

geek

0	1	2	3
g	e	e	k

LPS:-

0	1	2	3
0	0	0	0

① i=1, L=0
P[1] == p[0]
LPS[1] = 0
i = 2

② i=2, L=0
LPS[2] = 0
i = 3

③ i=3, L=0
LPS[3] = 0
i = 4

④ i=4 - will not work.

8)

0	1	2	3	4	5	6	7
a	b	c	d	a	b	c	y

LPS:

0	0	0	0	1	2	3	0

① L=0, i=1
LPS[1] = 0
i = 2

② L=0, i=2
LPS[2] = 0
i = 3

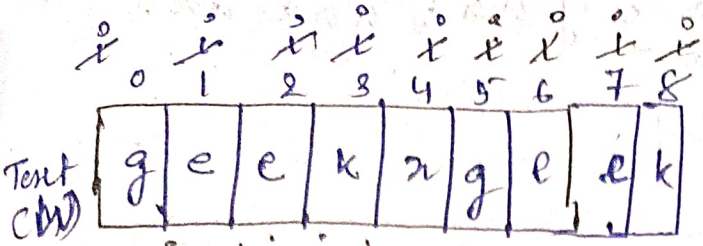
③ L=0, i=3
i = 4

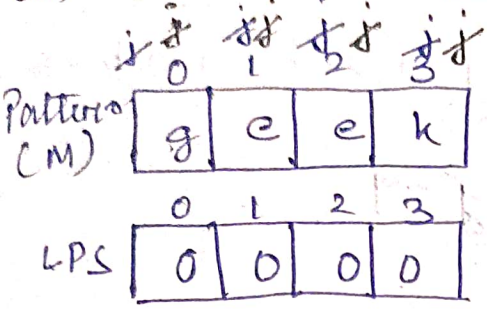
④ L=1, L=0
L=1
LPS[4] = 1
i = 5

⑤ L = ~~LPS[0]~~ 2
LPS[5] = 2
i = 6

⑥ L=3
LPS[6] = 3
i = 7

Dt-15/4/25



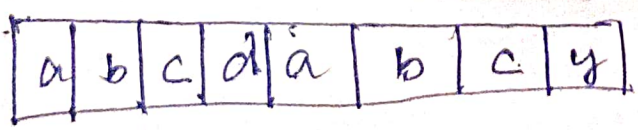


j = 4 | j = 4 = for and time
i - j = 4 - 4 = 0 | 2nd match will be from 5th posn
i - j = 9 - 4 = 5
For indexing start at 1
we will write i - j + 1
j - 1 = 4 - 1 = 3
LPS[3] = 0
so we will go to 0 posn of pattern

```
i = 0
j = 0
while(i < N)
{ if(i = j) if(Text[i] = Pattern[j])
        { i++; j++; }
    if(j = M)
      {
        result.print(i-j)
          j = LPS[j-1] }
    else if(j != 0)
          { j = LPS[j-1]
          }
    else{
          i++; } }
print __
```

→ KNP will take N time
complexity = O(N)
→ LPS will take M time
complexity = O(M)
Combining these
TOD - O(M×N)

Ex:-



Ex-
Text - a a b a a b a a a a y
Pattern - a b a a b a a a

LPS	0	1	0	1	2	3	4	5	2

i=0
a a b a a b a a a x y a a b a a b a a a
j
j=0 1 2 3 4 5 6 7 8 j=9
a a b a a b a a a

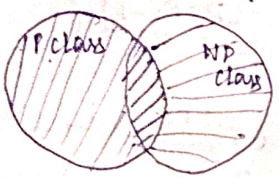
j = j-1 = 9-1 = 8

LPS[8] = 2 ⇒ j

j will start from 2.

→ Non-deterministic problem - No algorithm exist to solve the problem and it has no solution.

- $(c_1 \vee c_2) \wedge (c_3 \vee c_4) \wedge (c_5 \vee c_6)$ — ⇒ (conjunctive normal form)

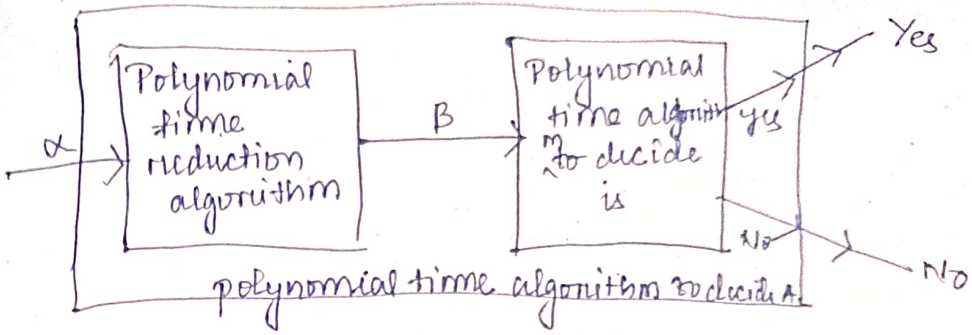


$an^2 + bn + c = 0$

$5n = 7$

$10n^2 + 5n - 7 = 0$

If $P_1 \in NP$ then $P_1 \in P$



Dt - 16/4/25

np - non-deterministic problem.

If two condition satisfy then we can say npc

(i) A language L subset equal to 0, 1

$L \subseteq \{0,1\}^*$ is NPC if

1. $L \in NP$
2. $L' \leq_p L$ for every $L' \in NP$

$P = NP = CO-NP$