Algorithms

Algorithm: Consists of finite num of (steps / stmts) to solve a problem
→ one or more fundamental opers
↓ satisfy
→ Definiteness (clear)
→ Finiteness. (finite amt of time)

↑
Mohd Al Khwarizmi → father of ~~Algebra~~ Algorithms

Mechanism:
→ Algo may accept 0 or more Inputs
→ Algo must produce atleast 1 output

i/p (0 or many) → [Algo / Logic / Mech] → o/p (desired) (Atleast 1)

Life cycle of Algorithm:

Problem definition
|
Requirements
|
Logic | Design
|
Develop Algo | Flowchart
|
Validation (checking the correctness) (whether all requirements met or not)
|
Analysis of Algo
|
Implementation
|
Testing and Debugging.

Need for Analysis:

Objective: Resource Consumption determines
↓ Time ↓ space ↓ Register (In sys prg) ↓ power (Mobile Algo) ↓ Bandwidth (N/w's)

→ Performance Comparision of various Algo of same problem

Analysis methodology:

Aposteriori / Experimentation Analysis
→ platform dependent Analysis
☺ report Exact values.
☹ Difficult to carry down
Non uniformity bcoz of diff platforms

Apriori Analysis
→ Independent of platform Analysis
→ No units in Apriori Analysis
CPU: Hypothetical Processor → Performs fundamental opers (+, -, *, /, <, >, =, ==) in 1 unit of time

Order of magnitude:
Determining frequency (count / num of times) the fundamental oper's involved in stmt / construct operation.

Objective of Apriori Analysis:
→ It is to represent the time complexity of the algorithm by a mathematical function of Input size 'n'. This mathematical function represents the rate of growth of time of Algo wrt Input.

T(n)
↓ Polynomial
↓ Const (c) ↓ Log (log n) ↓ Linear (n) ↓ nlogn ↓ Quad (n²) ↓ Cubic (n³) ↓ Geometric (n^k) (k>0)
↓ Exponential
a^n (a>1)

*) It is desirable to develop Algorithms having Polynomial time Complexity.

*) Imp

Problem
- Solvable (Algo Exist)
  - Deterministic
    - Polynomial (Tractable) (Optimal)
    - Exponential (Intractable)
  - Non-deterministic.
- Unsolvable (No Algo)
  - Ex:- Halting prob.

Types of Analysis:

Behavior
- Worst case behavior → Algo does max work
- Best case Behavior → Algo does min work.
- Average case behavior

Step 1: Determine all classes of I/P's that algo may take from size $I_1, I_2 \dots I_n$

Step 2: Determine the time for Each Input Class

Step 3: Determine the prob (P_i) that the Algo may take Input from the respective class.

$$A(n) = \sum_{i=1}^{k} P_i t_i$$

Note:
Relⁿ b/w Best, worst, Avg is

$$B(n) \le A(n) \le W(n)$$

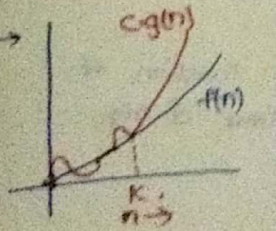
*) $B(n) = A(n) = W(n)$: Merge sort
$B(n) < A(n) = W(n)$: Linear search, Binary search
$B(n) = A(n) < W(n)$: Quicksort
$B(n) < A(n) < W(n)$: X

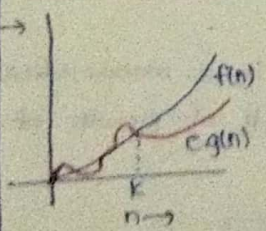
*) $A(n) \le W(n)$
- $A(n) \to O(W(n))$ — Always True
- $A(n) \to \Omega(W(n))$
- $A(n) \to \Theta(W(n))$
- $A(n) \to o(W(n))$

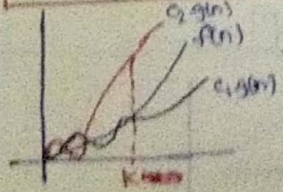
(may/maynot True)

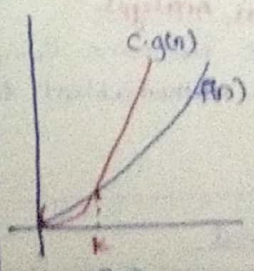
- $A(n) \to \omega(W(n))$ — Never / Always False.

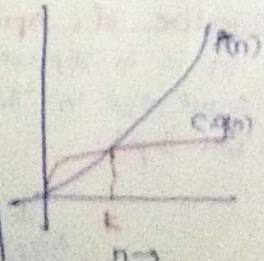
Asymptotic Notations & Apriori Analysis:

ASN
- Big (may/maynot be Asymptotically tight)
  - Big-oh (O)
    - $f(n)$ is $O(g(n))$
    - $f(n) \le c \cdot g(n)$ for some $c>0$ whenever $n>k$, $k>0$
    - 

  - Big-omega (Ω)
    - $f(n)$ is $\Omega(g(n))$
    - $f(n) \ge c \cdot g(n)$ for some $c>0$ whenever $n>k$, $k>0$
    - 

  - Theta (Θ)
    - It comes into picture when UB & LB are same
    - $f(n)$ is $\Theta(g(n))$ iff
      - $f(n)$ is $O(g(n))$
      - $f(n)$ is $\Omega(g(n))$
    - $c_1 g(n) \le f(n) \le c_2 g(n)$ for some $c_1, c_2 > 0$ whenever $n>k$, $k>0$
    - 

- Small (Always not Asymptotically tight)
  - Small-oh (o)
    - $f(n)$ is $o(g(n))$
    - $f(n) < c \cdot g(n)$ for all $c>0$ whenever $n>k$, $k>0$
    - 

  - Small omega (ω)
    - $f(n)$ is $\omega(g(n))$
    - $f(n) > c \cdot g(n)$ for all $c>0$ whenever $n>k$, $k>0$
    - 

*) Small fns are in order of big functions.

*) Small fns are in order of big functions.

*) $f(n) = n$

$n < (2n)$ closest Asymptotic

n^2
n^3
:
n^k
2^n
:
a^n

$f(n)$ is $O(n)$

*) $f(n) = 1+n+n^2$

$1+n+n^2 < (n^2)$ closest Asymptotic

n
1

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

*) $f(n) = 1+n+n^2 < \begin{matrix} O(n^2) \\ \Omega(n^2) \end{matrix} \} \Rightarrow \Theta(n^2)$

*) $f(n) = n < \begin{matrix} O(n) \\ O(n^2) \end{matrix}$

Note:

i) $f(n) = (n+k)^m \quad k, m > 0 \quad < \begin{matrix} O(n^m) \\ \Omega(n^m) \end{matrix} \Rightarrow \Theta(n^m)$

Def

Properties of log:

- $\log n^x = x \log n$
- $a^{\log_c b} = b^{\log_c a}$
- $\log ab = \log a + \log b$
- $\log \frac{a}{b} = \log a - \log b$

Summations:

- $\sum_{i=1}^{n} 1 = n \Rightarrow O(n)$
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \Rightarrow O(n^2)$
- $\sum_{i=1}^{n} n \cdot n^2 \Rightarrow O(n^3)$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = O(n^3)$
- $\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4} \Rightarrow O(n^4)$

- $\prod_{i=1}^{n} 1 = 1 \Rightarrow O(1)$
- $\sum_{i=1}^{n} 2^i \Rightarrow 2^{n+1} - 2 = O(2^n)$
- $\sum_{i=1}^{n} i 2^i \rightarrow (n-1)2^{n+1} + 2$
- $\sum_{x=1}^{n} \frac{1}{x} = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \log n = O(\log n)$
- $\prod_{i=1}^{n} i = n! \Rightarrow O(n!)$ or $O(n^n)$
- $\sum_{i=1}^{n} \log i = O(\log(n!))$ or $O(\log n^n)$ or $O(n \log n)$

Analogy b/w Asymptotic notations & Real numbers:

1. $f(n)$ is $O(g(n)) \iff a \leq b$
2. $f(n)$ is $\Omega(g(n)) \iff a \geq b$
3. $f(n)$ is $\Theta(g(n)) \iff a = b$
4. $f(n)$ is $o(g(n)) \iff a < b$
5. $f(n)$ is $\omega(g(n)) \iff a > b$

Properties of ASN:

	Ref	Sym	Transitivity	Transpose symmetry
Big-oh	✓	✗	✓	If $f(n)$ is $O(g(n))$ then $g(n)$ is $\Omega(f(n))$
Big-omega	✓	✗	✓	
Theta	✓	✓	✓	
Small-oh	✗	✗	✓	If $f(n)$ is $o(g(n))$ then $g(n)$ is $\omega(f(n))$
Small omega	✗	✗	✓	

Tricotomy property:

obeyed by real numbers

$a < b$
$a = b$
$a > b$

not obeyed by Asymptotic fns

$f > g : \Omega, \omega$
$f = g : \Theta, \bullet$
$f < g : \bullet O, o$

Ex:- $f = n^{1+\sin n} \quad g = n$

Min: $\sin n = -1$: $n^0 < n$

Max: $\sin n = 1$: $n^2 > n$

} Not have any order so, dis obeys Tricotomy property.

Dominance relation:

$$c < \frac{1}{n} < \log n < \sqrt{n} < n < n \log n < n\sqrt{n} < n^2 < n^3 < \cdots 2^n < 4^n \cdots < n^n$$

If $f(n)$ is $O(g(n))$ then $a \cdot f(n)$ is $O(g(n))$, $a>0$

*) If $f(n)$ is $O(g(n))$ and $d(n)$ is $O(e(n))$ then

(i) $f(n)+d(n)$ is $O(\text{Max}(g(n), e(n)))$

(ii) $f(n) * d(n)$ is $O(g(n) \cdot e(n))$

(3) $\log_2 n^x$ is $O(\log n)$ when $x>0$

(4) $\log_2^x n = (\log_2 n)^x$

Framework for Recursive Algorithms:

1. Derive Recurrence relation
2. Solve it by using

- Back substitution
  - → value
  - → order
- Master method
  - → order
- Recursion tree
  - → order

⇒ used to solve Asymmetric Recurrence relns

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

Master method:

$$T(n) = aT\left(\frac{n}{b}\right) + \theta\left(n^k \log^p n\right)$$

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

If $a>b^k$

$$T(n) = \theta\left(n^{\log_b a}\right)$$

If $a=b^k$

- $p>-1$: $T(n) = \theta\left(n^{\log_b a} \log^{p+1} n\right)$
- $p=-1$: $T(n) = \theta\left(n^{\log_b a} \log\log n\right)$
- $p<-1$: $T(n) = \theta\left(n^{\log_b a}\right)$

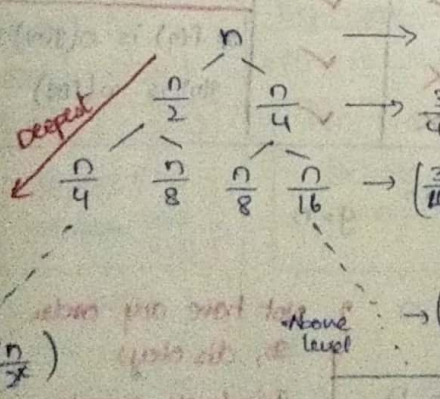
If $a<b^k$

- $p \geq 0$: $T(n) = \theta\left(n^k \log^p n\right)$
- $p<0$: $T(n) = O(n^k)$

Recursion tree method:

- used to solve Asymmetric recurrence relations

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



$n = 2^k$

$k = \log_2 n$

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

$\therefore 2^k T\left(\frac{n}{2^k}\right) + n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \cdots + \left(\frac{3}{4}\right)^{k-1} n$

$2^k \cdot T\left(\frac{n}{2^k}\right)$ (Approx)

$n(1) + n\left[1 + \frac{3}{4} + \left(\frac{3}{4}\right)^2 + \cdots\right]$

→ To get upper bound calculate so

$: n + n\left[\frac{1}{1-\frac{3}{4}}\right] + \text{so}$

$\therefore O(n)$

Space Complexity:

Space Complexity of a program 'P' is Computed as

$$S(P) = C + S_p$$

C → Const space
S_p → Instance Characteristic

Constant space:

space requirements for Instructions
Variables
Data structures
that need fixed space.

Instance characteristic function:

- It includes space for all such var & DS whose size is not known Apriori but depends on runtime instance of the problem being solved
- Also includes space for runtime stack in case of recursive Algorithms.

Space
- Input space → memory reqd for storing i/p's
- Workspace → space used during computation other than i/p space.
  - → Runtime stack comes under workspace
  - → Algo in General is said to be space Efficient if its workspace is constant or atmost $O(\log n)$

Divide & Conquer strategy:

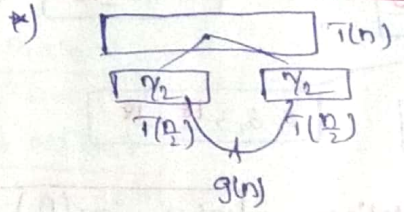
Control Abstraction:

```
Procedure DandC(A, n, P, Q)
{
  If (small(P,Q)) then
    return (G(A,P,Q));
  Else
  {
    m ← Divide(P,Q)        ⟶ Compulsory
    S1 ← DandC(A, n, P, m)
    S2 ← DandC(A, m, m+1, Q)
    Combine(S1, S2)        ⟶ Not mandatory
  }
}
```

Let T(n) rep TC of D&C then (f(n): small + g)

$$T(n) = \begin{cases} f(n), & n \text{ is small} \\ 2T\left(\frac{n}{2}\right) + g(n), & n \text{ is large.} \end{cases}$$

(g(n): small + divide + combine)



Variations:

1) $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ ⟶ Equal parts
   (a → Num of subprobs, b → size of subprob)

2) $T(n) = T(\alpha n) + T(\beta n) + f(n)$ ⟶ unEqual parts
   $\alpha \neq \beta$
   $0 < \alpha, \beta < 1$

3) $T(n) = T(\alpha n) + T((1-\alpha)n) + f(n)$ ⟶ un Equal parts whose sums to total size of the Problem.

(P.T.O)

Finding max, min :-

A:	1	2	3	4	5	6	7	8	9
	23	0	18	44	3	12	-19	8	10

```
Procedure MaxMin(A,n,max,min)
{
1. max ← min ← A[1]
2. For(i ← 2 to n)
   {
      If(A[i] > max)
         max = A[i]
      End if
      If(A[i] < min)
         min = A[i]
      End if
   }
}
```

Total Comp = 2*(n-1) + n (loop comp)

(Element Comp)

Avg of Worst

Optimal :

```
2. for(i ← 2 to n)
   {
      If(A[i] > max)
         max = A[i];
      Else
         If(A[i] < min)
            min = A[i];
   }
```

Comp Analysis :

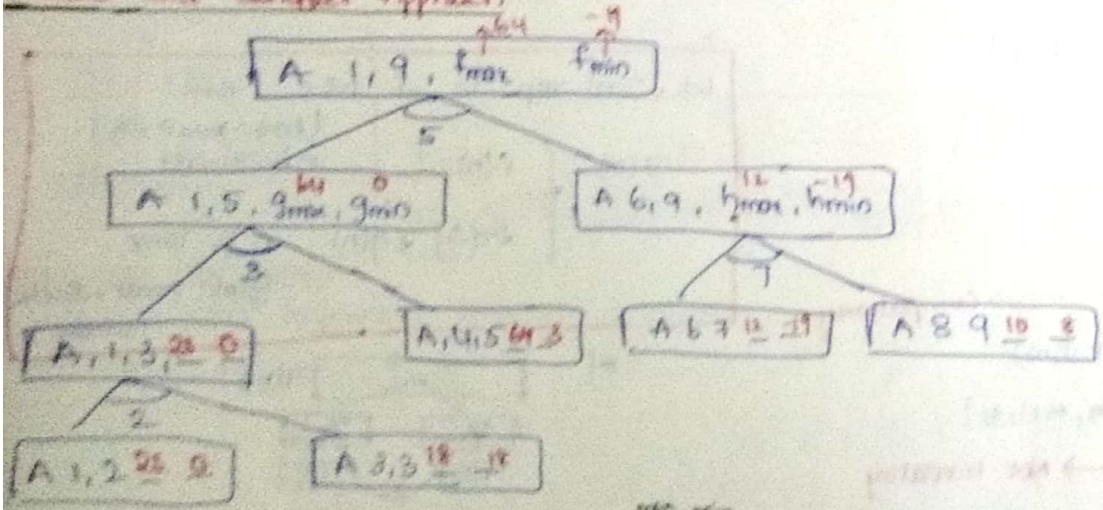
① Best : Inc order : (n-1) Comp

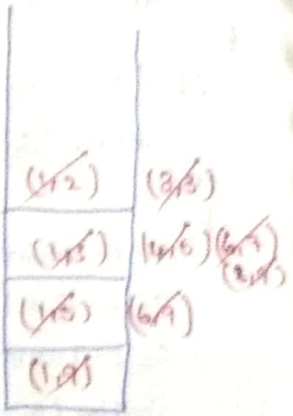
② Worst : dec order : 2(n-1) Comp

③ Avg case : First : n/2 + n/2 - 1
Second : n/2 → 2n/2 - 1 Comp

First Comp fails for 1/2 of given Elements.

Divide and Conquer Approach





Recurrence relation : (max, min)

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

Solution : $\frac{3n}{2} - 2$ → best, avg, worst

Note : Max, min

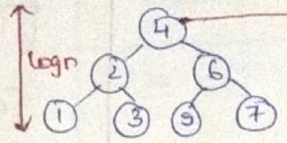
	Best (comparisons)	Avg	Worst	Time	Space (Comp)	Workspace
Normal Algo	2(n-1)	2(n-1)	2(n-1)	O(n)	O(n)	O(1)
optimized Algo	n-1	3n/2 - 1	2(n-1)	O(n)	O(n)	O(1)
DAC Algo	3n/2 - 2	3n/2 - 2	3n/2 - 2	O(n)	O(n) + O(logn)	O(logn)

② Binary search :

A<1 - - n> : Sorted array

n = 7

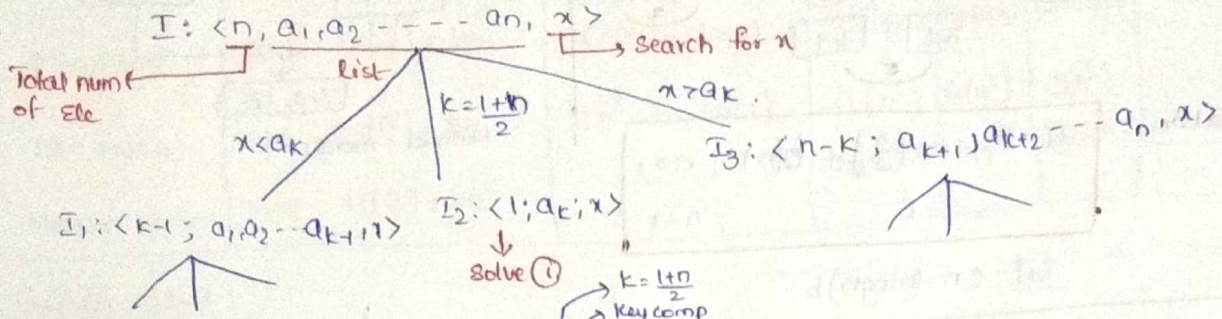
(1) 2 (3) 4 ((5) 6 (7)) ⇒



best case
Searching for middle element -

→ Worst case.

*) In Binary search we divide the problem into 3 sub problems.



k = 1+n/2
Key comp
Loop comp

Recurrence relation :

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

Sol: $c + a(\log n)$

Note :

	Best	Avg	Worst	space comp	workspace
Binary search	Ω(1)	Θ(logn)	O(logn)	c+n+logn O(n)	O(logn)

③ Merge sort :

Merging

Ex:- L_1 : < 2 5 8 > (-1) (n_1)

L_2 : < 3 7 9 12 15 > (-1) (n_2)

2 way merging

L : < 2 3 5 7 8 9 12 15 >

$n_1 \le n_2$

$n = n_1 + n_2$

Best case :

Num of Comparisions = n_1

All $|L_1| <$ first $|L_2|$

Ex:- < 2 3 5 8 > < 10 12 15 18 25 >

Worst Case :

Num of Comp = $n_1 + n_2 - 1 \Rightarrow n - 1$

$(L_1 - 1)|L_1| <$ first $|L_2|$

last $|L_1| >$ All $|L_2|$

Ex:- < 2 3 5 40 > ($n_1 - 1$) < 10 12 15 18 25 > (n_2)

Merging : O(n)

Merge sort :

A: [(1, 310)] [(2, 285)] [(3, 179)] [(4, 652) (5, 351)] [(6, 423) (7, 861) (8, 254) (9, 450) (10, 520)]

B: [285 310] [179]

[179 285 310] [351 652]

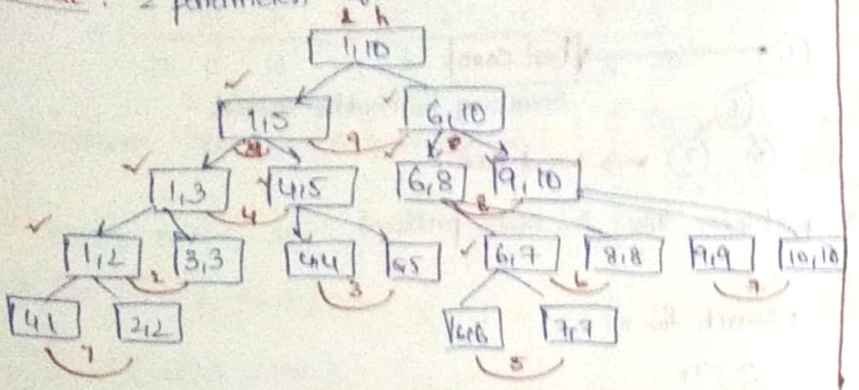
[179 285 310 351 652] [423 861] [254]

[179 285 310 351 652] [254 423 861] [450 520]

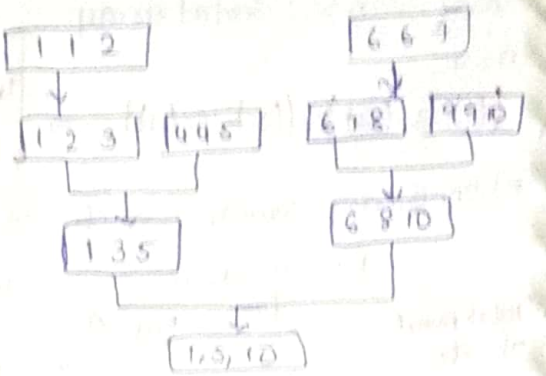
[179 285 310 351 652] [254 423 450 520 861]

[179 254 285 310 351 423 450 520 652 861] : Sorted.

Divide : 2 parameters read (l, h)



Merge : 3 parameters read (l, m, h)



Num of merges = 9

Recurrence relation :

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

Sol : $cn + (n\log n)b$

Note :

	Best	Avg	Worst	SC	Worst space
Merge	$O(n_1)$	$O(n)$	$O(n)$	$c+n+\log n+n$	$n+\log n$
Merge sort	$\Omega(n\log n)$	$\Theta(n\log n)$	$O(n\log n)$	$c+n+\log n+n$ (i/p, stack, temp array) $O(n)$	$n+\log n$

*) Merge sort is time efficient but not space efficient due to usage of temp array (not in place sorting)

(4) Matrix Multiplication :

In General :

$A_{n\times n}$; $B_{n\times n}$; $C_{n\times n}$

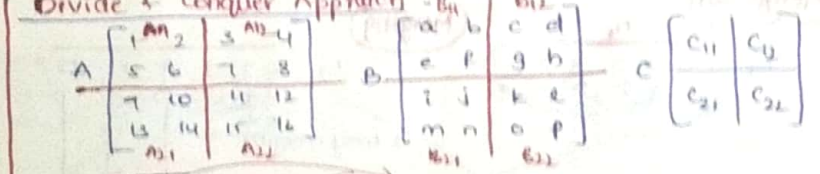
1) $A \pm B = C$: $O(n^2)$

2) $A * B = C$

```
for i = 1 to n
    for j = 1 to n
        C[i,j] = 0
        for k = 1 to n
            C[i,j] = A[i,k] + B[k,j];
```

$O(n^3)$

(ii) **Divide & Conquer Approach** :



$C_{11} = A_{11}B_{11} + A_{12}B_{21}$ → Divide Conquer

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

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

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

*mul's = 8
Add's = 4

Recurrence relation :

$$T(n) = \begin{cases} 8T\left(\frac{n}{2}\right) + bn^2 & ; n > 2 \\ c & ; n \le 2 \end{cases}$$

Sol : $c(n^3) + bn^2[n-1]$

$O(n^3)$

(iii) **Strassens matrix multiplication** :

$A_{n\times n}$; $B_{n\times n}$; $C_{n\times n}$, $P, Q, R, S, T, U, V : n/2 \times n/2$, $A_{ij}, B_{ij}, C_{ij} : \frac{n}{2} \times \frac{n}{2}$

$P = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$
$Q = (A_{21} + A_{22}) \cdot B_{11}$
$R = A_{11} (B_{12} - B_{22})$
$S = A_{22} (B_{21} - B_{11})$
$T = (A_{11} + A_{12}) (B_{22})$
$U = (A_{21} - A_{11}) (B_{11} + B_{12})$
$V = (A_{12} - A_{22}) (B_{21} + B_{12})$

$C_{11} = P + S - T + V$
$C_{12} = R + T$
$C_{21} = Q + S$
$C_{22} = P + R - Q + U$

Mul's : 7
Add's : 18

Recurrence relation :

$$T(n) = \begin{cases} 7T\left(\frac{n}{2}\right) + bn^2 & ; n \\ c \end{cases}$$

Sol : Complex.

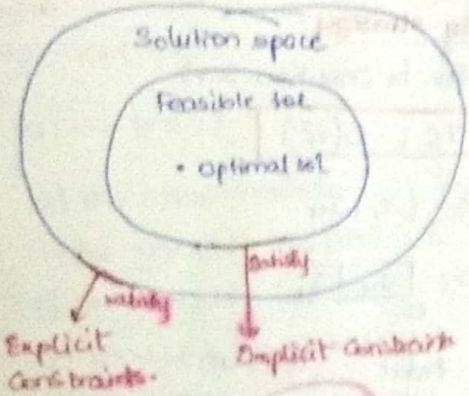
$O(n^{\log_2 7})$. $O(n^{2.81})$

Note :

D & C Algo	Recurrence relation	Solution	Best	Avg	Worst	space comp	Extra space
MinMax	$T(n) = 2T(\frac{n}{2}) + 2$	$\frac{3n}{2} - 2$	$\Omega(n)$	$\Theta(n)$	$O(n)$	$O(n)$	$O(\log n)$
Binary search	$T(n) = T(\frac{n}{2}) + a$	$c + a(\log n)$	$\Omega(1)$	$\Theta(\log n)$	$O(\log n)$	$O(n)$	$O(\log n)$
Merge sort	$T(n) = 2T(\frac{n}{2}) + bn$	$cn + b(n \log n)$	$\Omega(n \log n)$	$\Theta(n \log n)$	$O(n \log n)$		$O(n)$ $n+\log n$
DRC Mul'n	$T(n) = 8T(\frac{n}{2}) + bn^2$	$cn^3 + bn^2(n-1)$	$\Omega(n^3)$	$\Theta(n^3)$	$O(n^3)$	-	-
Strassen Mul'n	$T(n) = 7T(\frac{n}{2}) + bn^2$	-	$\Omega(n^{2.81})$	$\Theta(n^{2.81})$	$O(n^{2.81})$		-

Greedy method :

Terminology :



Objective function :
It refers to minimization / maximization of given problem

Control Abstraction :

```
Procedure Greedy (A, n)
{
  1. solution ← ϕ
  2. for i = 1 to n
     {
        x ← select (i, n)
        If (Feasible (x, solution))
           Add (x, solution);
     }
     return (solution);
}
```

≥ O(n)

Definition :
It is an Algo design technique used for solving problems whose solutions are viewed as a result of making a sequence of decisions :
- Based on local optimality principle
- In each step out of the available options greedily select that option which satisfies objective function in addition to feasibility criteria

① Knapsack problem :
- knapsack capacity = M
- n objects (O_i)

weight(w_i) profit(p_i)

Trivial Cond	Non Trivial Cond
$\Sigma w_i \le M$	$\Sigma w_i > M$
All $x_i = 1$	obj fn : Max $\Sigma p_i x_i$
Profit = Σp_i	STC : $\Sigma w_i x_i \le M$

$0 \le x_i \le 1$	$x_i < 0$
Fractional Knapsack	Binary (0/1) Knapsack
solspace : ∞	2^n

n = 3 ; M = 20

$\langle w_1\ w_2\ w_3 \rangle = \langle 18\ 15\ 10 \rangle$

$\langle p_1\ p_2\ p_3 \rangle = \langle 25\ 24\ 15 \rangle$

1) Greedy about profit

$x_1 = 1$, $x_2 = \frac{2}{15}$, $x_3 = 0$; $\sum w_i x_i = 20$

$\sum p_i x_i = 25 + \frac{2}{15} * 24 + 0$

$= 28.3$

Feasible

2) Greedy about weight

$x_1 =$, $x_2 = \frac{10}{15}$, $x_3 = 1$; $\sum w_i x_i = 20$

$\sum p_i x_i = 15 + \frac{10}{15} * 24$

$= 31$

Feasible

3) Greedy about p/w

$\frac{p_1}{w_1} = \frac{25}{18} = 1.38$

$\frac{p_2}{w_2} = \frac{24}{15} = 1.6$

$\frac{p_3}{w_3} = \frac{15}{10} = 1.5$

$x_1 = 0$, $x_2 = 1$, $x_3 = \frac{5}{10}$; $\sum w_i x_i = 20$

$\sum p_i x_i = 24 + \frac{1}{2}(15)$

$= 31.5$

optimal

4) Job sequencing with deadlines :

- n jobs
- Single cpu
- Non preemptive scheduling

Each job has: AT = 0, BT = 1, P_i (Profit), d_i (deadline)

Sol space : 2^n

"Select a subset of n-jobs such that the jobs in the subset can be completed within their deadlines and maximize profit".

Ex:- n = 4

$\langle J_1\ J_2\ J_3\ J_4 \rangle$

$\langle d_1 - - d_4 \rangle = \langle 2\ 1\ 2\ 1 \rangle$

$\langle P_1 - - P_4 \rangle = \langle 100\ 10\ 15\ 40 \rangle$

|J| = 0 : Feasible ; $^4C_0 = 1$; Sol : $\{\phi\}$

|J| = 1 : Feasible ; $^4C_1 = 4$; Sol : $\{J_1, J_2, J_3, J_4\}$

|J| = 2 : $^4C_2 = 6$; Feasible

$\{J_1, J_2\}$ J_2 J_1

$\{J_1, J_3\}$ J_1 J_3 ; J_3 J_1

$\{J_2, J_4\}$ ×

$\{J_1, J_4\}$ J_4 J_1

$\{J_2, J_3\}$ J_2 J_3

$\{J_3, J_4\}$ J_4 J_3

|J| = 3 : Not Feasible

|J| = 4 : Not Feasible

Greedy strategy :

- similar to Insertion sort

TC : $O(n^2)$

$J \leftarrow \{J_1, J_4\}$

cpu: J_4 | J_1 (0)

Profit = 100 + 40

= 140

optimal

Solution space : n!

Feasible solution : n!

3) optimal merge patterns :

- Merging of files with 2-way merging

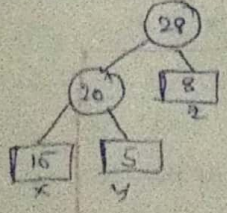
x : $\langle$ 2 5 8 10 $\rangle$ (n) y : $\langle$ 3 7 9 15 20 $\rangle$ (m)

z : $\langle$ 2 3 5 7 8 9 10 15 20 $\rangle$ (n+m)

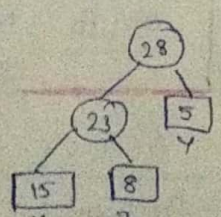
Total record movements = n+m

*) If there are n files then (n-1) steps are read

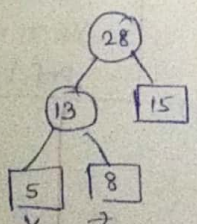
Ex:- x : 15 y : 5 z : 8



Rec mov = 20 + 28 ⇒ 48



RM : 23 + 28 ⇒ 51



RM : 13 + 28 = 41

optimal.

*) Total num of record mov's = weighted external path length

$$= \sum_{i=1}^{n} d_i q_i$$

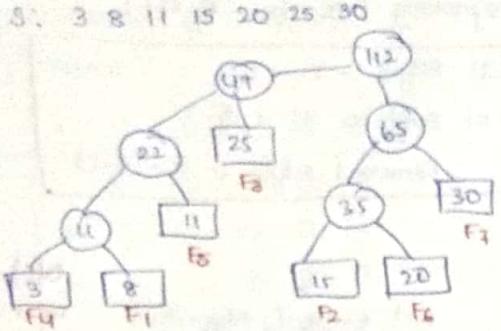
$d_i \rightarrow$ dist from root to F_i

$q_i \rightarrow$ size of q_i

Ex:- n:7 1 2 3 4 5 6 7

$\langle F_1 \ldots F_7 \rangle$: $\langle 8, 15, 25, 3, 11, 20, 30 \rangle$

S: 3 8 11 15 20 25 30



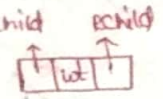
RM: 11+22+47+112+65+35

RM: 292

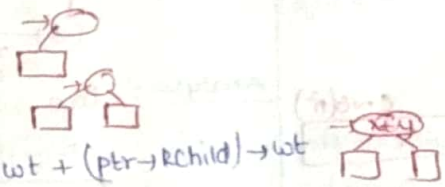
RM: 4*8 + 3*15 + 2*25 + 4*3 + 4*3 + 20*3 + 2*30

(OR) : 292

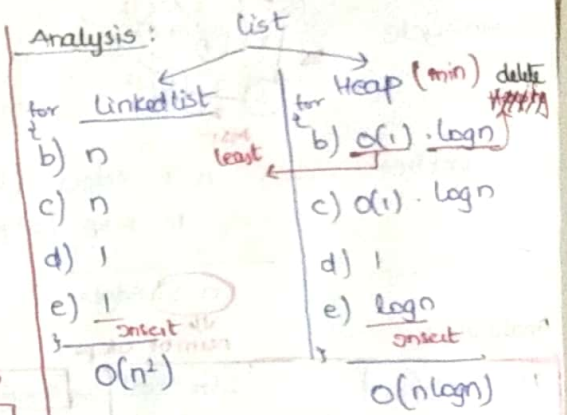
Algorithm:




```
Algo Tree (n, list)
{
  for(i=1 to n-1)
  {
    a) ptr = new(treenode);
    b) ptr → Lchild = least(list);
    c) ptr → Rchild = least(list);
    d) ptr→wt = (ptr→lchild)→wt + (ptr→Rchild)→wt
    e) insert(ptr, list)  // delete x, y and add (x+y) node into tree
  }
}
```



Analysis:



④ Huffman Coding: (Data Encoding + Data Compression)

- Application of optimal merge pattern
- Frequency dependent Encoding

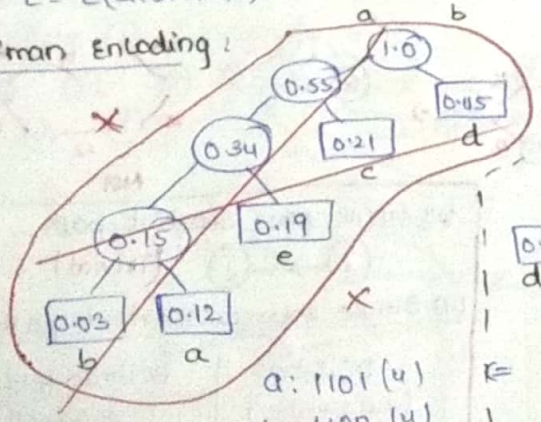
Ex:- $L = \sum(a_1 b_1 c_1 d_1 e) = \langle 0.12, 0.03, 0.21, 0.45, 0.19 \rangle$

a b c d e

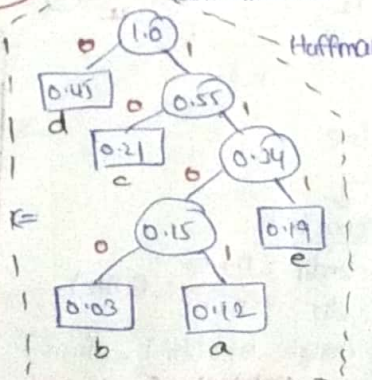
Conventional Encoding

- Each 3 bits

Huffman Encoding:



a: 1101 (4)
b: 1100 (4)
c: 10 (2)
d: 0
e: 111 (3)



Ex:- c d d c d d d e a d

- Conventional - 3(10) → 30 bits
- Huffman :- 10 0 0 10 0 0 0 111 1101 0 → 17 bits.

↓ Data Compressed

Note:

Avg num of bits/element = $\sum d_i q_i$

$d_i \rightarrow$ dist from root to alphabet

$q_i \rightarrow$ probability

Min cost spanning trees :

Spanning tree : A subtree $T(V, E')$ of $G(V, E)$ where $E' \subseteq E$ is a spanning tree iff T is a tree.

Applications : Multicasting, Broadcasting

Solution space / Num of possible spanning trees $= n^{n-2}$ (Cayley's formula)

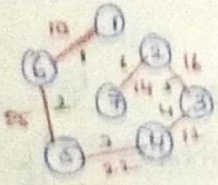
(Back) *) Num of edges to be removed from a graph having 'n' nodes and 'e' edges to get a spanning tree is $e - n + 1$

Total edges $= e$
Num of edges in ST $= n - 1$
Removed edges $= e - (n - 1)$

Min cost spanning tree

* Prims Algorithm :

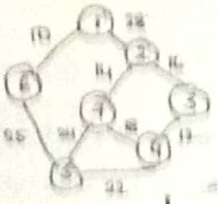
(i) Adjacency
(ii) least cost
(iii) No cycle



*) n vertices → n-1 edges (for MST)
- 1 Edge (step 1 selecting min edge)
(n-1) edges
↓ num of steps
Analysis

TC :		
$O(n^2)$	: General	Complete graph
$O((n+e)\log n)$	: Heap	Not Complete

Better



(ii) Kruskal's Algorithm

- Construct a heap with Edge costs
Costs : 10; 12; 14; 16; 18; 22; 24; 28 ; 14 ; e (edges)

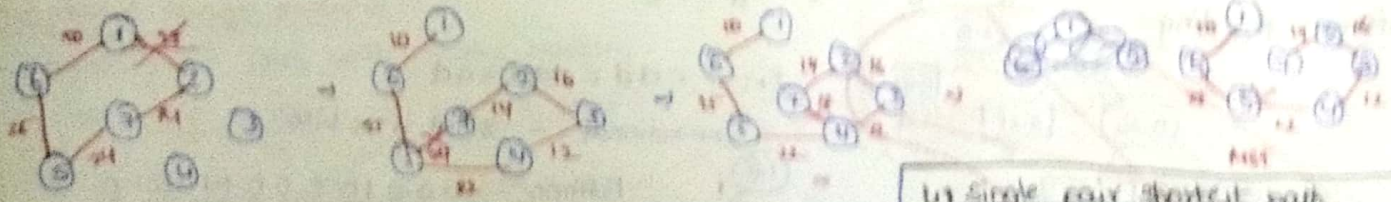
Analysis : Logic : ; min heap Priority
e : num of elements

TC : $e \log e / e \log v$

Note :
- Connectedness is always maintained in prims Algorithm
- Topologies may vary for both Algorithms if multiple edges have same cost

(iii) Dijkstra's Approach

- whenever a cycle happened in the course of addition then delete the edge with more cost



(6) Shortest paths :

Graph representation : $G_1(V, E)$ $|V| = n$ $|E| = e$

Adjacency matrix

$A(i,j) = 1$ if e_i
$= 0$

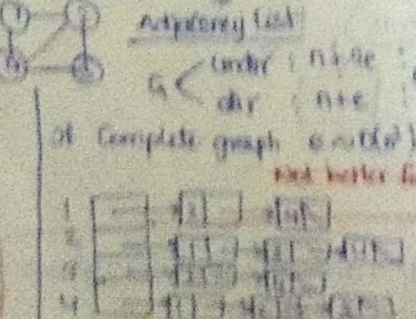
	1	2	3	4
1	-	1	0	1
2	1	-	1	1
3	0	1	-	1
4	1	1	1	-

$O(n^2)$

Adjacency list

G_1 < (undir : n+2e ; dir : n+e) (time)

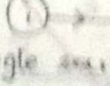
If Complete graph $e \approx O(n^2)$ $\approx O(n \cdot n^2)$
Not better for K_n graph



:) Direct access
Better for Complete graph

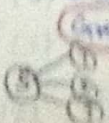
:(Seq access

(i) Single pair shortest path


(Trivial)

(ii) Single source shortest paths

Dijkstra's (+ve weights)	Bellman ford (-ve weights)
Greedy	DP



(iii) All pairs shortest paths

- Floyd warshall Algo → DP

Single Source Shortest path :

① Dijkstral Algo (Greedy method)

- matrix method
- Spanning tree method

Matrix method :

C[i,j] → edge Cost matrix

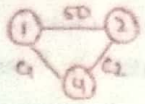
$V_0 = 1$ Source

C	1	2	3	4	5	6
1	–	50	45	10	∞	∞
2	∞	–	10	15	∞	∞
3	∞	∞	–	∞	30	∞
4	20	∞	∞	–	15	∞
5	∞	20	35	∞	–	∞
6	∞	∞	∞	∞	3	–

If $V_0 = 1$

vertex selected	1	2	3	4	5	6
{1,4}	–	50	45	(10)	∞	∞
{1,4}	–	50	45	(10)	(25)	∞
{1,4,5}	–	(45)	45	(10)	(25)	∞
{1,4,5,2}	–	(45)	(45)	(10)	(25)	∞
{1,4,5,2,3}	–	(45)	(45)	(10)	(25)	(∞)

Relaxation



$C_1 + C_2 < SD$

Update

Analysis :

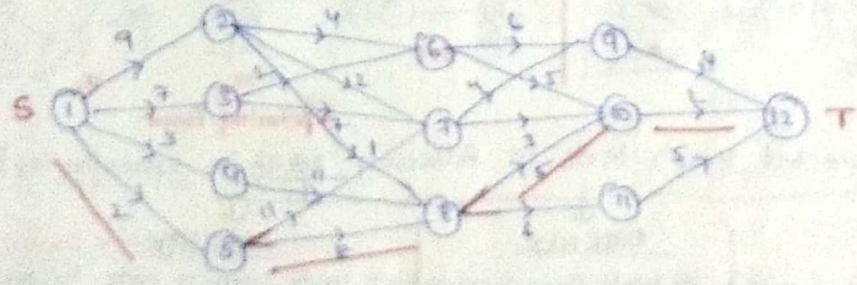
Adjacency matrix → $O(n^2)$

Heap → $O(mE \log n)$

∵ path can't be determined

Multistage graph : (edge $V_i \rightarrow V_{i+1}$)

- 1 stage multigraph (k=5)

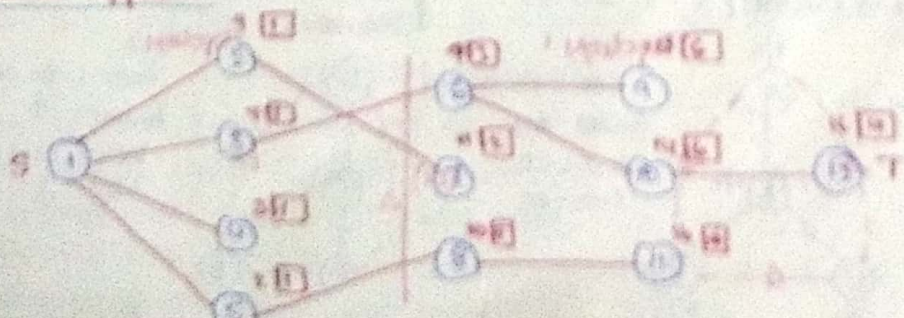


Greedy method :

Cost (1-7) : 2 + 8 + 5 + 1 = 11

1 — 3 — 6 — 10 — 12

Dijkstras Approach :

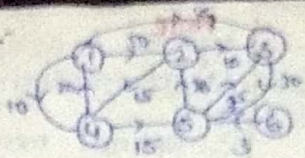


Dijkstras method

Cost (1-7) : 16

1 — 3 — 6 — 10 — 12

Ex:

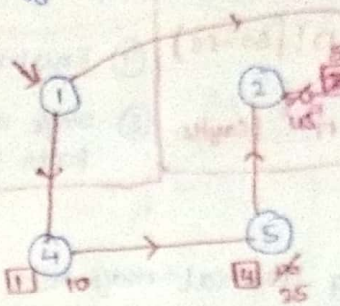


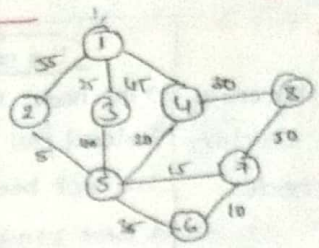
(1) Spanning tree method

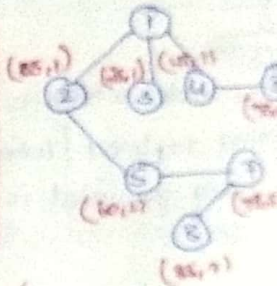
☐ : VIA

× : min cost

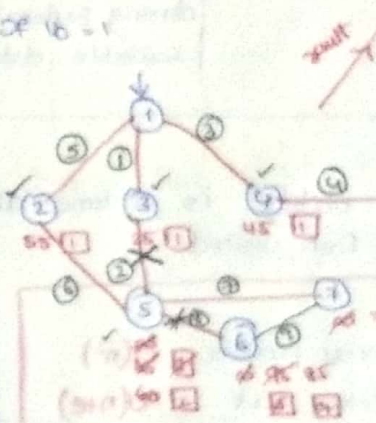
If $V_0 = 1$







If $V_0 = 1$



fault

∴ Path can be determined

Note:

Problem	Solution Space
Fractional Knapsack	∞
0/1 Knapsack	2^n
Job sequencing	2^n
Optimal merge pattern	$n!$ (SS-FS)
Min Cost spanning tree	n^{n-2} (Cayles

Note:

1. Job sequencing - $O(n^2)$ // similar to insertion sort
2. optimal merge pattern — linked list : $O(n^2)$; Heap : $O(n \log n)$
3. prim's Algorithm : $O(n^2)$ → Adj matrix : Complete (better); $O((n+e)\log n)$ → Heap : Not Complete (better)
4. Kruskal Algorithm : $O(e \log e)$ / $O(e \log n)$
5. single source shortest paths (Dijkstras) : Adj matrix : $O(n^2)$; → Heap : $O(n+e)\log n$

Graph Techniques:

① Depth First Search:

States of node during traversal may be

Enode
- The node which is currently getting Explored / whose children is being generated is enode

Live node
- The node which is not fully Explored / All of whose children has not been generated is livenode
- There can be many live nodes during Exploration and stored in suitable data structure

Dead node
- The node which is fully Explored all of whose been generated is dead node.

Timing values (d/f)

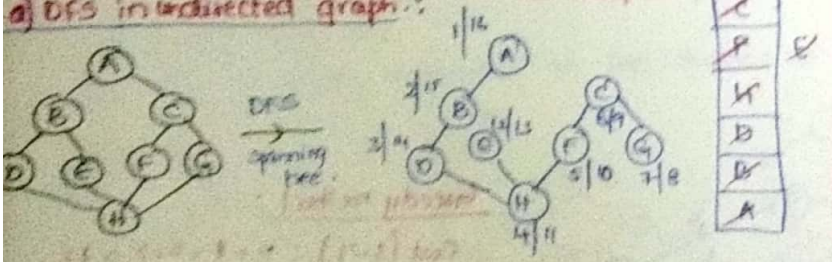
discovery time (d)
- Discovery time of node 'x' is a time at which the node is first visited

Finishing time (f)
- The time at which the node becomes dead is casted finishing time.

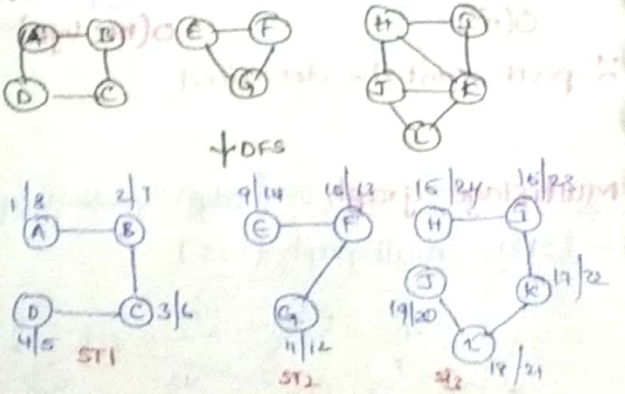
Analysis

DFS → Adjacency matrix : $O(n^2)$; Adjacency list : $O(n+e)$

a) DFS in undirected graph:



b) DFS in undirected, disjoint graph



Spanning forest

c) DFS in directed graph:

DFS when carried out in a directed graph leads to following edges in spanning tree/forest

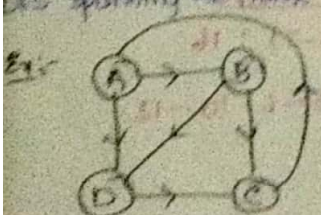
Tree edge:
- Tree edges are part of DFS spanning tree/forest

forward edge
- It leads from a node to its own child/descendant

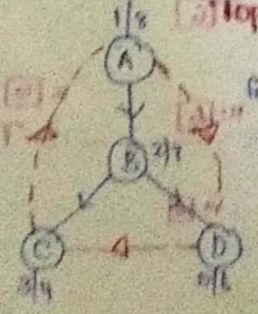
Back edge
- It leads from a node to its ancestor

Cross edge
- It leads from a node to another node which is neither ancestor nor its descendant

Ex:

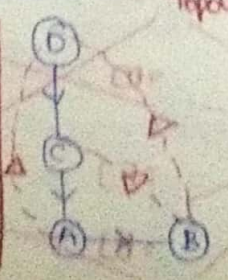


DFS:



Topology 1
Tree : AB, BC, BD
forward : AD
Back : CA
Cross : DC

(or)



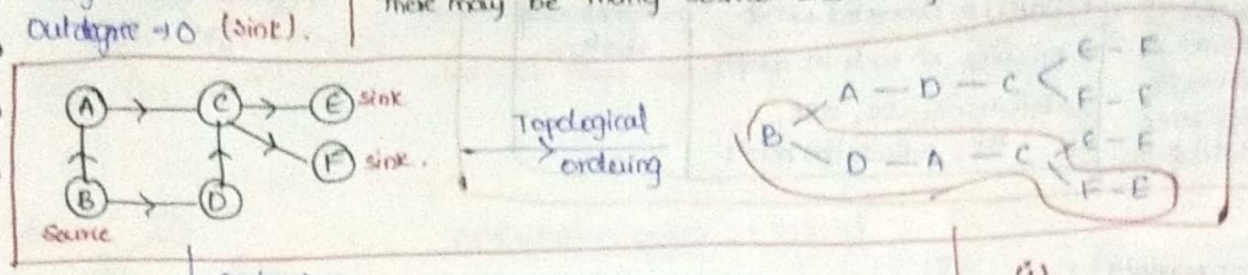
Topology 2
Tree : DC, CA, AB
forward : ϕ
Back : BC, BD, AD
Cross : ϕ

Fully determined DFS spanning tree

d) DFS in directed Acyclic graph (DAG) : (⇒ Topological sorting)

Indegree → 0 (source)
Outdegree → 0 (sink).

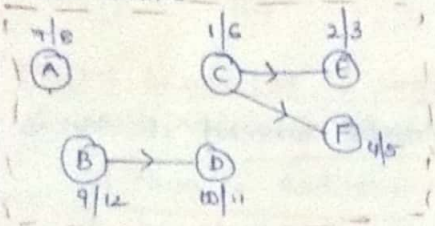
There may be many source and many sinks.



↓ Apply DFS.

↑ (i) Application of DFS with DAG.

start with c



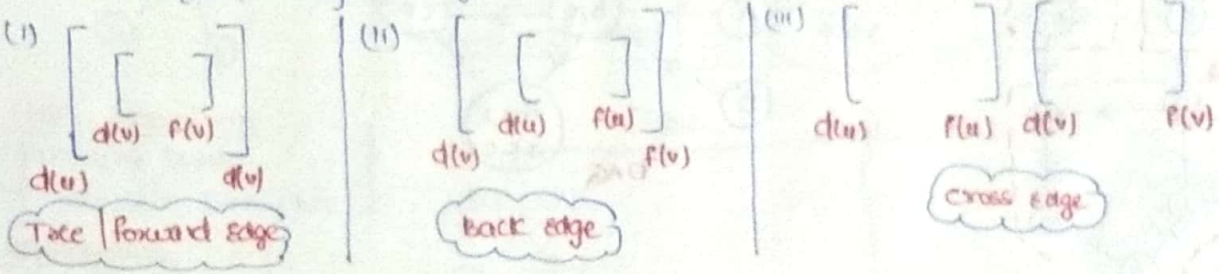
Spanning forest

Arrange the nodes in descending order of finishing time. ⇒ B — D — A — C — F — E

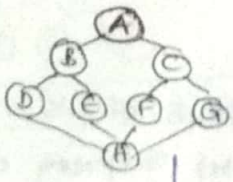
(Topological order)

(ii) parenthesization theorem :

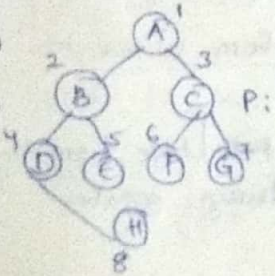
In any DFS of a directed / undirected graph, Exactly one of the following holds for any pair of vertices (u, v) being an edge.



② Breadth First Search :



FIFO BFS

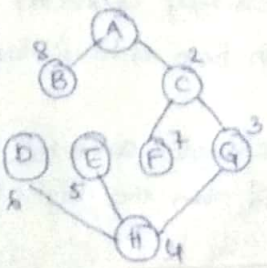


B	C	D	E	F	G	H	
A	A	B	B	C	C	D	P:

FIFO ↑ (BFS)

A B C D E F G H

LIFO BFS



B	C	D	G	H	E	F
A	A	C	C	G	H	H

LIFO BFS. → (D-search)

A C G H E D F B

(DFS)

A C G H E B D F

Applications of BFS and DFS :

→ Both BFS and DFS can be used to check whether a given graph is Connected or not
→ Both BFS and DFS can be used to determine presence of cycle in the graph
→ DFS is used in Backtracking strategy
→ BFS is used in Branch and Bound strategy
→ BFS acts as most Efficient Algorithm for the problem of single source shortest path when the graph has unit weight edge costs
→ BFS and DFS are used to determine Connected Components, Strongly Connected Components, biconnected Components and articulation points.

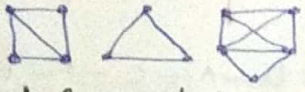
Note: Apps of BFS & DFS.

BFS		DFS
• Branch & Bound • unit weights - single source shortest paths	• Graph is connected or not • presence of cycle in graph - determine CC, SCC, BC, Articulation points	• Backtracking strategy.

Components :

① Connected Components :

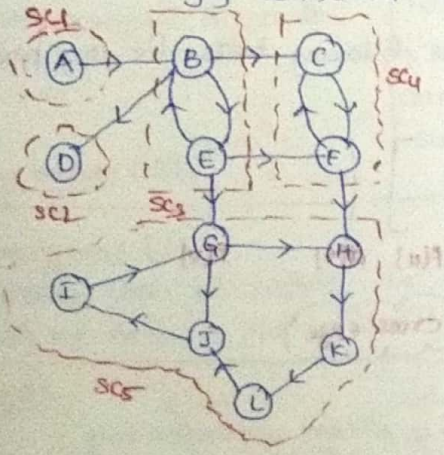
Maximal subgraph of a given undirected graph which is connected is connected component of the graph.



② Strongly Connected Components :

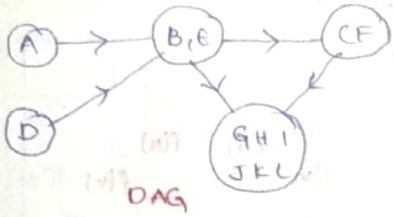
Two vertices u and v in a directed graph are said to be strongly connected if there is a path from u to v and from v to u.

This relation partitions the vertex set V of the graph into disjoint sets (maximal) known as strongly connected components of the graph.



G

5 strongly Connected Components.



DAG

Properties :

→ Every directed graph is directed Acyclic graph of its strongly connected components

→ Let G_1, G_2 be two distinct strongly connected components of the graph. Let $(u, v \in G_1)$ and $(u', v' \in G_2)$. If there is a path from u to u' then there cannot be a path from v' to v in Graph G.

→ Let G_1, G_2 be two distinct strongly connected components. If there is a path/edge from a vertex in G_1 to a vertex in G_2 then the highest finishing time of G_1 is always greater than highest finishing time of G_2 ie

$$f(G_1) > f(G_2)$$

③ Biconnected Components :

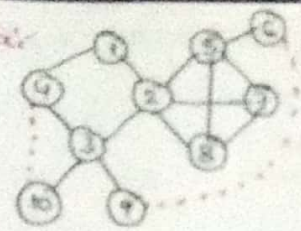
Maximal subgraph of a given graph which is biconnected is biconnected component of the graph.

④ Articulation point (cut vertex)

It is that vertex in the graph, the removal of which partitions the graph into two or more non empty components.

Graph is said to be biconnected if it has no articulation point.

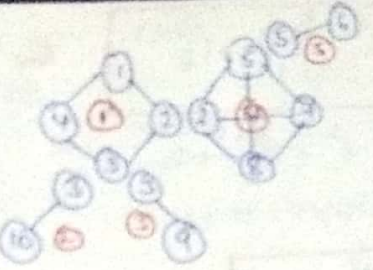
Ex:



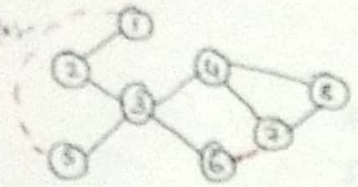
Articulation points : {2, 3, 5}

Biconnected Components : 5

Num of edges added : 2



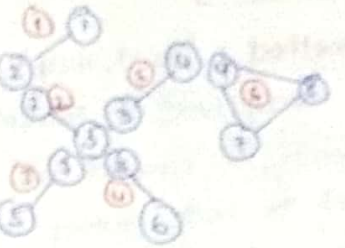
Ex:



Articulation points : {2, 3, 4}

Biconnected Components : 6

Num of Edges added : 2



Making Biconnected to single Component:

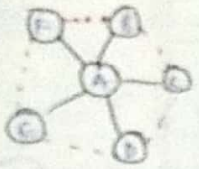
- Add Additional Edges:

> Num of Additional Edges ≤ Num of Articulation points

↓

This property holds good except for (star graph)

Ex:



Articulation point : {A}

Biconnected Components : 5

Num of edges added : 5

5 ≤ 1

False

Heap Algorithms:

→ Priority Queue

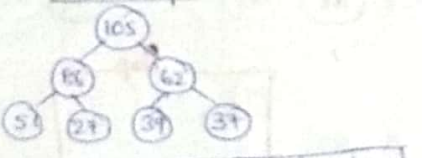
→ Complete binary tree

Operations

1. Heap creation
2. Heapify
3. Heap sort

Types

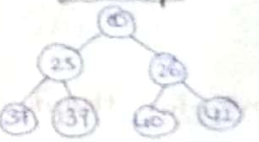
max heap



Rep:

105	86	62	51	27	39	37

min heap

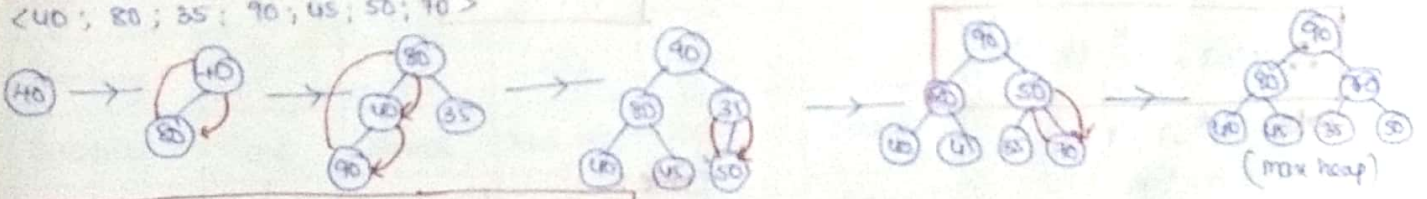


10	25	26	37	39	40	42

Heap creation:

a) Insertion method:

<40 ; 80 ; 35 ; 90 ; 45 ; 50 ; 70 >



(max heap)

Analysis:

Best Case : decreasing order ; O(n)

Worst Case : Increasing order ; O(n log n)

*) Given a heap with n Elements, Time Complexity to insert a new element into it is O(log n)

Framework to prove worst case:

- Max num of nodes at any level 'i' of a BT (start with 1) : 2^{i-1}
- Num of Comparisions needed to insert a node @ level 'i' : $i-1$
- Num of Comparisions for all nodes @ level 'i' : $(i-1)2^{i-1}$
- Total Comparisions for all nodes @ level 1...k : $\sum_{i=1}^{k} (i-1)2^{i-1}$: $T(n)$

$\therefore T(n) = \sum_{i=1}^{k} (i-1)\, 2^{i-1}$ for creation method

Sol: $k \cdot 2^k - 2^{k+1} + 2$

$\therefore n\log n - 2n + 2$

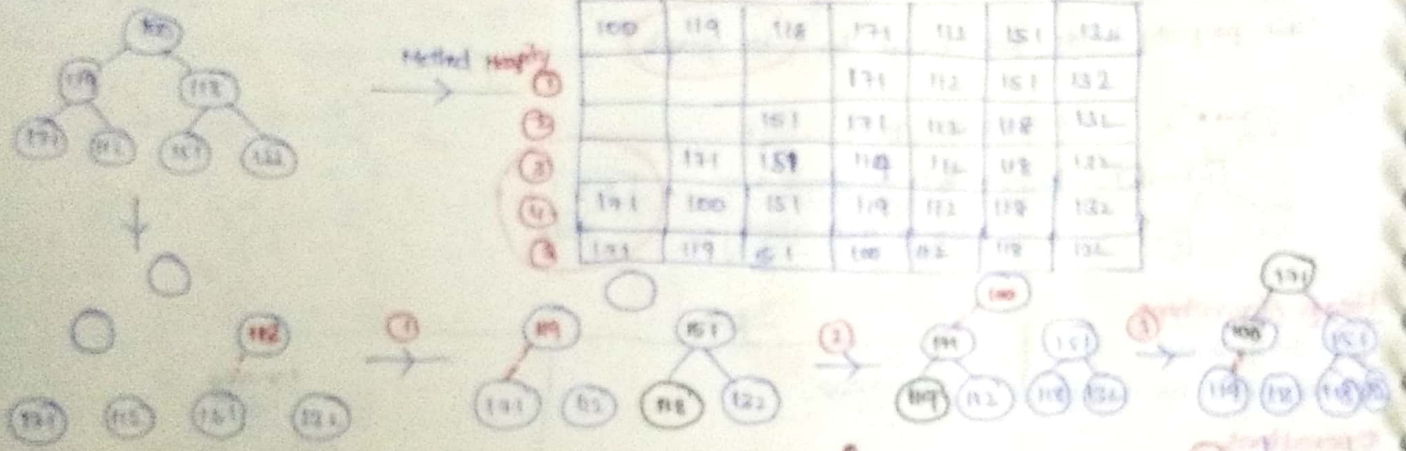
$\therefore O(n\log n)$

② Heapify method : (Build_Heap)

Assume a binary tree is Existing A(100 ; 119 ; 118 ; 131 ; 113 ; 151 ; 132)

- Heapify process is carried out by adjusting the nodes level by level
- The value of the node getting adjusted is Compared with the value of its largest child
- A swap is made if necessary and the Comparision process continues until it reach leaf level
- In the worst case the node being adjusted occupies the position of leaf

Ex:-



Some work for Heapify method :

- Max num of nodes at any level i of a BT : 2^{i-1}
- Num of Comp needed to adjust node @ level i : $k - i$
- Num of comp for all nodes @ level i : $(k-i)\, 2^{i-1}$
- Total num of Comparisions from level 1 ... k : $\sum_{i=1}^{k} (k-i)\, 2^{i-1}$ T(n)

$\therefore T(n) = \sum_{i=1}^{k} (k-i)\, 2^{i-1}$

Sol: $2^k - k - 1$

$n - \log n - 1$

$O(n)$

Heap sort :

Algorithm :

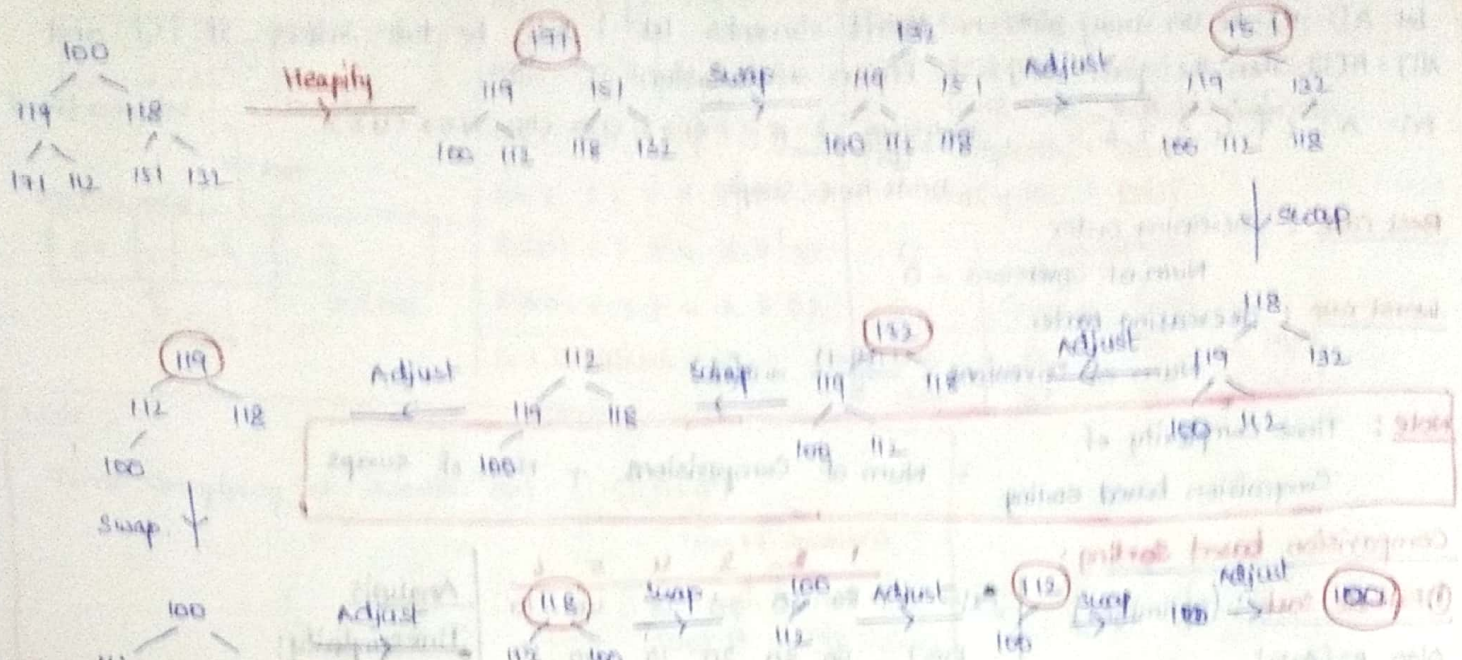
```
Algo HS (A[n])
{
1. Heapify(A[n])          : O(n)
2. for i <- n down to 2   : (n-1)[max cost of Heapify]
   {
     a) Swap(A[1], A[i])  : O(1)
     b) Adjust(A[1])      : O(log i)
   }
}
```

∴ Heapsort : $O(n\log n)$

Note :

Heap Sort	merge Sort	Quick Sort
- $O(n\log n)$	- $O(n\log n)$	- $O(n\log n)$
- Additional array not used	- Used	- not used
- Space : $O(1)$	$O(n)$	$O(n)$
- Swap : $O(1)$	$O(n \log n)$	$O(\log n)$ - Best $O(n)$ - Worst
- Inplace	not inplace	inplace
- not stable	stable	not stable
- Can be unnecessary swaps (Worst)		Can be unnecessary swaps (Worst)

Ex:- A < 100 ; 119 ; 118 ; 191 ; 112 ; 151 ; 132 > n=9



order : 100 112 118 119 132 151 191

T	100	119	118	191	112	151	132
Heapify	191	119	151	100	112	118	132
1	151	119	132	100	112	118	191
2	132	119	118	100	112	151	191
3	119	112	118	100	132	151	191
4	118	112	100	119	132	151	191
5	112	100	118	119	132	151	191
6	100	112	118	119	132	151	191

Note :
- Heap insert, Heap delete → O(logn)
- Heapify → O(n)
- Insertion method → O(nlogn)
- Heapsort → O(nlogn)

Sorting methods :

Note :

Sorting	①	②	③	④	⑤
Bubble	Comp	Internal	Non-rec	Inplace	Stable
Selection	Comp	Internal	Non-rec	Inplace	Stable
Insertion	Comp	External / Internal	Non-rec	Inplace	Stable
Quick	Comp	Internal	Rec	Inplace	Not stable
Heap	Comp	Internal	Non-rec	Inplace	Not stable
Merge	Comp	External	Rec	Not Inplace	Stable
Radix	Non-Comp	Internal	Non-rec	Not Inplace	Stable

Inversions: (used to determine num of swaps)

Let A[1..n] be an array with n distinct elements let i and j be two indices. If i<j and A[i] > A[j] then the pair (i j) is known as Inversion.

Eg:- A: (9 4 5 7 6) (1 2 3 4 5)

Inversions of A: <1 2> <1 3> <1 4> <1 5> <4 5>

↓ needs more swaps.

Best case: Increasing order

Num of Inversions = 0

Worst case: decreasing order

Num of Inversions = $\frac{n(n-1)}{2} \Rightarrow O(n^2)$

Note: Time Complexity of Comparision based sorting = Num of Comparisions + Num of swaps

Comparision based Sorting;

① Bubble sort: (optimized)

```
Algo BS(A,n)
{
  for pass ← n downto 1
  {
    flag = 0;
    for i ← 1 to pass-1
    {
      If (A[i] > A[i+1])
      {
        flag = 1;
        swap(A[i], A[i+1]);
      }
    }
    If (flag == 0)
      break;
  }
}
```

Eg:- A: 80 60 20 15 40 10 (1 2 3 4 5 6)

Pass 1:	60	80	20	15	40	10
	60	20	80	15	40	10
	60	20	15	80	40	10
	60	20	15	40	80	10
	60	20	15	40	10	(80)
Pass 2:	20	15	40	10	(60)	(80)
Pass 3:	15	20	10	(40)	(60)	(80)
Pass 4:	15	10	(20)	(40)	(60)	(80)
Pass 5:	10	(15)	(20)	(40)	(60)	(80)
Pass 6:	(10)	(15)	(20)	(40)	(60)	(80)

n-iterations.

Analysis

Time Complexity:

Best case: $O(n)$

Worst case: $O(n^2)$

Space complexity: $O(n)$

Workspace: $O(1)$

Bubble sort

	Comp	+	Swaps	
Inc	n		0	$O(n)$
dec	n^2		n^2	$O(n^2)$

② Selection sort:

- Least num of swaps in worst case: $O(n)$

```
Algo SS(A,n)
{
  for i ← 1 to n-1
  {
    min ← i
    for j ← i+1 to n
    {
      If (A[j] < A[min])
        min = j;
    }
    swap(A[min], A[i]);
  }
}
```

Eg:- 80 60 20 15 40 10 (1 2 3 4 5 6)

min

Pass 1: (10) 60 20 15 40 80

Pass 2: (10) (15) 20 60 40 80

Pass 3: (10) (15) (20) 60 40 80

Pass 4: (10) (15) (20) (40) 60 80

Pass 5: (10) (15) (20) (40) (60) 80

Pass 6: (10) (15) (20) (40) (60) (80)

n-iterations

Analysis

Time Complexity: $O(n^2)$

Space Complexity: $O(n)$

Workspace: $O(1)$

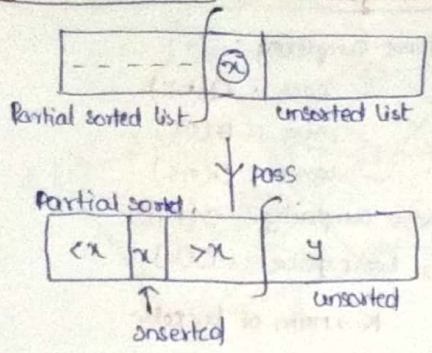
Selection sort

	Comp	+	swaps	
Any data	n^2		n	$O(n^2)$

Note: Recurrence relation for both Bubble sort and Selection sort is

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

③ Insertion sort :



Ex:- < 8 | 2 4 9 3 6 >

Pass 1: < 2 8 | 4 9 3 6 >

Pass 2: < 2 4 8 | 9 3 6 >

Pass 3: < 2 4 8 9 | 3 6 >

Pass 4: < 2 3 4 8 9 | 6 >

Pass 5: < 2 3 4 6 8 9 >

n-1 Iterations

Analysis :

Time Complexity :

Best case : $O(n)$: Inc order

Worst case : $O(n^2)$: dec order

Space Complexity : $O(n)$

Workspace : $O(1)$

Insertion sort

	Comp +	swaps	
Inc	n	0	$\Rightarrow O(n)$
dec	n^2	n^2	$\Rightarrow O(n^2)$

Note :

Time Complexity of Insertion sort : $O(n+d)$

d → num of inversions

min (Inc) 0 ; max. $\frac{n(n-1)}{2}$ (dec)

④ Quicksort : (Partition Exchange sort)

```
int partition (A,P,m)
{
  V ← A[P];
  i ← P
  j ← m
  while(1)
  {
    i ← i+1;
    until(A[i] ≥ V);
    loop
      j ← j-1;
    until (A[j] ≤ V);
    if (i < j) then
      swap(A[i],A[j])
    Else
      break;
  }
  swap(A[P],A[j])
  return j;
}
Procedure Quicksort(A,l,h)
{
  If (l<h)
  {
    1. j ← partition (A,l,h+1);
    2. Quicksort (A,l,j-1);
    3. Quicksort (A,j+1,h);
  }
```

n+1th Element

Ex:- A: < 65 70 75 80 85 60 55 50 45 | ∞ >

Pivot

Pass 1: < 65 45 75 80 85 60 55 50 70 >

< 65 45 50 80 85 60 55 75 70 >

< 65 45 50 55 85 60 80 75 70 >

< 65 45 50 55 60 85 80 75 70 >

< 60 45 50 55 > 65 < 85 80 75 70 >

i	j
1	10
2	9
3	8
4	7
5	6
6	5

Pass 2: < 55 45 50 60 > 65 < 70 80 75 85 >

< 55 45 50 > 60 > 65 < 70 75 80 > 85 >

Pass 3: < 50 45 > 55 > 60 > 65 < 80 75 > 85 >

Pass 4: < 45 > 50 > 55 > 60 > 65 > 70 > 75 > 80 > 85

Pass 5: 45 50 55 60 65 70 75 80 85

Analysis: I : < n, a_1, a_2 ---- an >

sorted

unsorted → Best : I_1 : < $(\frac{n}{2})$ □ $(\frac{n}{2})$ >

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

TC : $O(n \log n)$

SC : $O(n)$

WS : $O(\log n)$

sorted → Inc (worst) : I_2 : < □ (n-1) >

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

TC : $O(n^2)$

SC : $O(n)$

WS : $O(n)$

sorted → dec : I_3 : < (n-1) □ >

Note :

- Pivot $\frac{1}{3}^{rd}$ place $\Rightarrow T(n) = T(\frac{n}{3}) + T(\frac{2n}{3}) + O(n)$ → Generalize $T(n) = O(n) + T(\alpha n) + T((1-\alpha)n)$ → $O(n \log n)$
- n+1th Element : ∞ → If first element is largest then i Increments and may go to infinite loop because of array out of bounds. → C : No Bound checking; Java : Bound Checking → raise Exception.

Non-Comparision based Sort

① Radix sort :

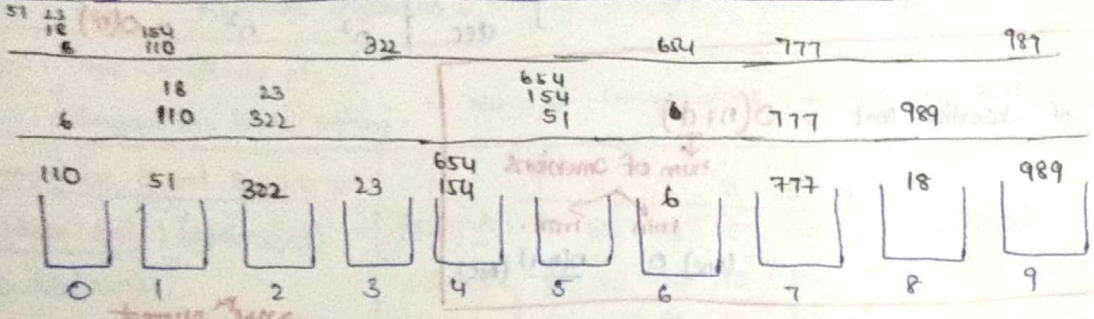
	1	2	3	4	5	6	7	8	9	10
A:	322	154	6	51	18	23	110	654	989	777
Pass1:	110	51	322	23	154	654	6	777	18	989
Pass2:	6	18	110	322	823	51	154	654	777	989
Pass3:	6	18	23	51	110	154	322	654	777	989

Analysis :

Time Complexity :
- Best : $\Omega(nk)$
- Avg : $\Theta(nk)$
- Worst : $O(nk)$

Space Complexity : $O(n+k)$
- Workspace : $O(k)$

k → num of buckets.



Dynamic programming : (Bellman of RAND Company in US)

- Algorithm design technique used for solving problems whose solutions are viewed as a result of making a sequence of decisions.

For some problems these sequence of decisions can be made in a step wise manner based on local info avail at each step.

↓

These type of problems are solvable by Greedy method

But some problems like multi stage graph, optimal sequence of decisions can't be made based on local info available at each step.

↓

Enumerate all possible decision sequences & select the best

↓

Brute force Approach

☹ Enormous time & space consumption

solution.

Dynamic programming

- Based on enumeration but it tries to cutoff those decision sequences where there is no possibility of getting optimal sol.
- Dp makes an appeal to Principle of optimality for making its sequence of decisions

Principle of optimality :

It states that whatever the initial state and decision are, the remaining sequence of decisions must constitute an optimal decision sequence wrt the state resulting from first decision.

→ The difference b/w Greedy & Dp is that Greedy method generates only one decision sequence whereas Dp may generate many decision sequences.

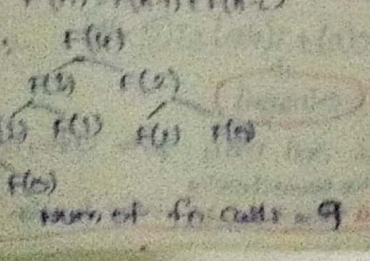
☺ Dp : optimal solutions of the subproblems are memorized (tabulated)

↓

To avoid recomputation if req'd later

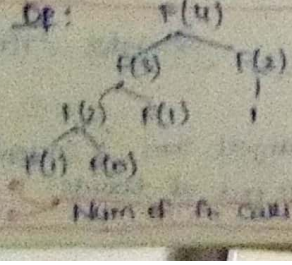
* The process of tabulating the solutions of the subproblems is known as (programming)
* In (divide and conquer) we have disjoint subproblems but in (Dp) we have overlapping subproblems

Eg: $F(n) = F(n-1) + F(n-2)$



No. of fn calls = 9

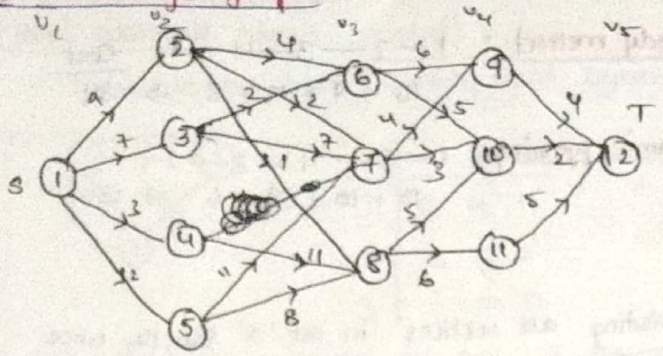
Dp:



Num of fn calls = 7

F(1)	1
F(2)	2
F(3)	3

(1) Multistage graph :

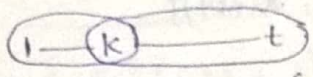


$l = 5$

Cost	1	2	3	-	-	—	12
1	=	9	7				
2							
3							
:							
(							
12							

$c[i,j]$ → Edge Cost

Let $\underset{\text{Cost}}{c}[i,j]$ represents cost of the path from vertex j present in stage i to reach dest vertex t



$Cost[1,1] = \min\{c[1,K] + Cost[2,K]\}$ where $K \in V_2$, $\langle 1,K \rangle \in$ Edge Exist

↑ stage

↓ Generalize.

$$Cost[i,j] = \min\{c[i,t] + Cost[i+1,K]\} \rightarrow (1)$$

$Cost[l-1, j] = c[i,t]$ → Terminating condition

↓ last but one stage

$D(i,j)$ - 'K' that minimizes (1)

Solution :

Level 4 :

$Cost[4,9] = 4$

$Cost[4,10] = 2$

$Cost[4,4] = 5$

Level 3 :

$Cost[3,6] = \min\{9+4, 10+2, 10+7\} = 7$ $\quad D[3,6] = 10$

$Cost[3,7] = \min\{9+4, 8+2, 10+5\} = 5$ $\quad D[3,7] = 10$

$Cost[3,8] = \min\{10+7, 11+11\} = 7$ $\quad D[3,8] = 10$

Level 2 :

$Cost[2,2] = \min\{6+7=11, 4+7=11, 2+5=7\} = 7$ $\quad D[2,2] = 7$

$Cost[2,3] = \min\{6+2+7=9, 7+5=12\} = 9$ $\quad D[2,3] = 6$

$Cost[2,4] = 8 + 11 + 7 = 18$ $\quad D[2,4] = 8$

$Cost[2,5] = \min\{7+11+5=16, 8+7=15\} = 15$ $\quad D[2,5] = 8$

Level 1 :

$Cost[1,1] = \min\{2+9+7=16, 3+7+9=16, 4+8+18=21, 5+2+15=17\} = 16$ $\quad D[1,1] = 2$

Determining path : l stages → (l-2) decisions.

Start from : $D(1,1) = 2$

$D(2, D(1,1)) \rightarrow D(2,2) = 7$

$D(3, D(2,2)) \rightarrow D(3,7) = 10$

∴ path : 1 — 2 — 7 — 10 — 12

Cost = 16.

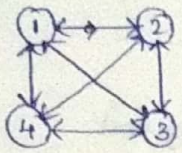
Analysis :

- Bottom up approach
- Time Complexity : $O(V^2)$

② Travelling salesman problem : (Bottom up)

- Solution space - $(n-1)!$

Ex:-



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

Greedy method : $1 - 2 - 3 - 4 - 1$ Cost

$10 + 9 + 12 + 8 \Rightarrow 39$

Dynamic programming : $1 - 2 - 4 - 3 - 1$

$10 + 10 + 9 + 6 \Rightarrow 35$

Let $g(i,s)$ rep cost of the tour from vertex i visiting all vertices in set 's' exactly once and terminating @ v_0;

$$g(1,\{2,3,4\}) = \min_{\substack{k \in s \\ \langle 1,k\rangle \in E}} \{ c[1,k] + g(k, s-\{k\}) \}$$

↓ Generalize

$$g(i,s) = \min_{\substack{k \in s \\ \langle i,k\rangle \in E}} \{ c[i,k] + g(k, s-\{k\}) \} \rightarrow (1)$$

$g(i,\phi) = c[i,v_0]$ → Terminating condition.

$J(i,s) = k$ that minimizes (1)

Solution :

$|s| = 0$

$g(4,\phi) = c[4,1] = 8$

$g(3,\phi) = c[3,1] = 6$

$g(2,\phi) = c[2,1] = 5$

$|s| = 1$

$g(2,\{3\}) = 9+6 = 15$

$g(2,\{4\}) = 10+8 = 18$

$g(3,\{2\}) = 13+5 = 18$

$g(3,\{4\}) = 12+8 = 20$

$g(4,\{2\}) = 8+5 = 13$

$g(4,\{3\}) = 9+6 = 15$

$|s| = 2$

$g(2,\{3,4\})$: $9 + g(3,\{4\}) \Rightarrow 29$; $10 + g(4,\{3\}) \Rightarrow 25$ → 25, J = via 4

$g(3,\{2,4\})$: $13 + g(2,\{4\}) \Rightarrow 31$; $12 + g(4,\{2\}) \Rightarrow 25$ → 25, J = via 4

$g(4,\{3,2\})$: $8 + g(2,\{3\}) \Rightarrow 23$; $9 + g(3,\{2\}) \Rightarrow 27$ → 23, J = via 2

$|s| = 3$

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

$1 \rightarrow 2 + g(2,\{3,4\}) \Rightarrow 10 + 25 \Rightarrow 35$

$1 \rightarrow 3 + g(3,\{2,4\}) \Rightarrow 15 + 25 \Rightarrow 40$

$1 \rightarrow 4 + g(4,\{2,3\}) \Rightarrow 20 + 23 \Rightarrow 43$

→ 35, J = via 2

Path determination :

$1 - 2 - 4 - 3 - 1$

(1) → J = via 2

via 4

via 3

1

Analysis :

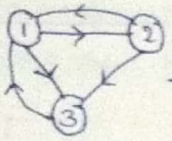
Time complexity : $O(n^2 2^n)$

All pairs shortest path : (Bottom up)

Floyd warshall Algo : $O(n^3)$ — $O(n^3)$

- solvable by both Greedy and Dynamic programming

Ex:-



C	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

Let $A^k(i,j)$ represents the cost of the path from vertex i to vertex j not going through intermediate of order greater than 'k'.

$$A^k(i,j) = \min_{k=n \text{ to } 1} \{ A^{k-1}(i,k) + A^{k-1}(k,j) , A^{k-1}(i,j) \}$$

$$A^0(i,j) = C(i,j) \rightarrow \text{Terminating Condition}$$

$A^0 = C$

A^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

A^1

A^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

A^2

A^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

A^3

A^3	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

Algorithm :

```
Algo FW (G, V, E, C, A)
{
  int C[1--n, 1--n], A[1--n, 1--n];
  1. for i ← 1 to n
       for j ← 1 to n                    } → O(n^2)
         A[i,j] = C[i,j];
  2. for (k ← 1 to n)
       for i ← 1 to n                    } → O(n^3)
         for j ← 1 to n
           A[i,j] = min [ A[i,k] + A[k,j] , A[i,j] ];
}
```

Analysis :-

Time Comp : $O(n^3)$

(4) Longest Common subsequence (LCS) :

X : < B D C B B A B C D B A > : n

Substring : $\frac{n(n+1)}{2}$

- → Relative order
- → Contiguous

Ex:- < B > < B B A >

Subsequence : $2^n - 1$

- → Relative order
- → non Contiguous

Ex:- < B > , < A A >

ABC : string of length 3

Substring

L = 1 ; A, B, C → 3
L = 2 ; AB, BC → 2
L = 3 ; ABC → 1

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

$O(n^2)$

Subsequence

L = 1 ; A, B, C → 3
L = 2 ; AB, BC, CA → 3
L = 3 ; ABC → 1

$2^n - 1$

$O(2^n)$

Let X, Y be two strings having n and m characters. The problem of LCS is to determine a subsequence that is common to both x and y and of longest length.

Applications :

- DNA strand matching
- Plagarism checking
- Google suggestion in search bar.

Let i & j be two indices into the strings X and Y as shown

$X \langle x_1 x_2 \cdots x_m \rangle$ (i) $\quad Y \langle y_1 y_2 \cdots y_n \rangle$ (j)

Let L[i,j] be length of Common Subsequence of the above two string as defined.

It is derived as follows:

Case 1: X = <GTCCTAATA>, Y = <CGATAATTGAGA>

Case 2: X = <GTCCTAATA>, Y = <CGATAATTGAGA>

If x[i] = x[j]
L[i,j] = 1 + L[i-1, j-1]

If x[i] ≠ x[j]
L[i,j] = max { L[i,j-1], L[i-1,j] }

L[-1,j] = L[i,-1] = 0 → Terminating cases

Tree Approach: (better for shorter strings)

Lcs("ABBAB", "ACBAB") : 4

+1

Lcs(ABBA, ACBA) 3

+1

Lcs(ABB, ACB) 2

+1

Lcs(AB, AC) 1

max

Lcs(AB, A) 1 | Lcs(A, AC)

Lcs(AB, ϕ) 0 | Lcs(A, A) | Lcs(ϕ, AC) 0

+1

Lcs(ϕ, ϕ) 0 | Lcs(ϕ, ϕ) 0

∴ Lcs → 4 < ABAB / ABAB (ABAB)

Matrix Approach:

X = <ABCBDAB> Y = <BDCABA>

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0↑	0↑	0↑	1_A	1←	1_A
2	B	0	1_B	1←	1←	1←	2_B	2←
3	C	0	1↑	1↑	2_C	2←	2↑	2↑
4	B	0	1_B	1↑	2↑	2↑	3_B	3←
5	D	0	1↑	2_D	2↑	2↑	3↑	3↑
6	A	0	1↑	2↑	2↑	3_A	3↑	4_A
7	B	0	1_B	2↑	2↑	3↑	4_B	4↑

Analysis:

O(n*m)

LCS → 4

BCBA BCAB

(6) matrix chain product: (mcp)

Let $A_1, A_2, A_3 \cdots A_n$ be a chain of n-matrices where matrix A_i is of dimension $P_{i-1} \times P_i$. The mcp problem is to fully parenthesize the given chain of matrices so as to minimize the num of scalar multiplications.

Matrix

Square:

$A_{mxn} * B_{mxn} = C_{mxn}$

Num of scalar mul = n^3

Non-Square

$A_{2\times10} * B_{10\times15} = C_{2\times15}$

Num of scalar mul = 2*10*15 = 300

Ex: <BCD> A

$B_{2\times10} * C_{10\times20} * D_{20\times20} = A_{2\times20}$

<BCD>?

(BC)D | B(CD)

2*10*20 + 2*20*20 | 2*10*20 + 10*20*20

800 | 10,400

min

Let $A_{i,j}$ be resultant product matrix generated by multiplying the chain of matrices $\langle A_i * A_{i+1} * A_{i+2} * \cdots * A_j \rangle$

Ex: $A_{1\cdots4}$: $\langle A_1 * A_2 * A_3 * A_4 \rangle$

$A_1(A_2A_3A_4)$ (1)(2) | $(A_1A_2)(A_3A_4)$ (3) | $(A_1A_2A_3)A_4$ (5)(6)

Num of possibilities = 5

- Let m[i,j] rep the num of scalar muls needed to get the matrix A[i,j]

*) Any optimal parenthesization must split the chain 'i to j' between matrices A_K and A_{K+1} such that the num of scalar multiplications is minimum

The number of scalar muls is given by sum of first subchain, second subchain and multiplying them together.

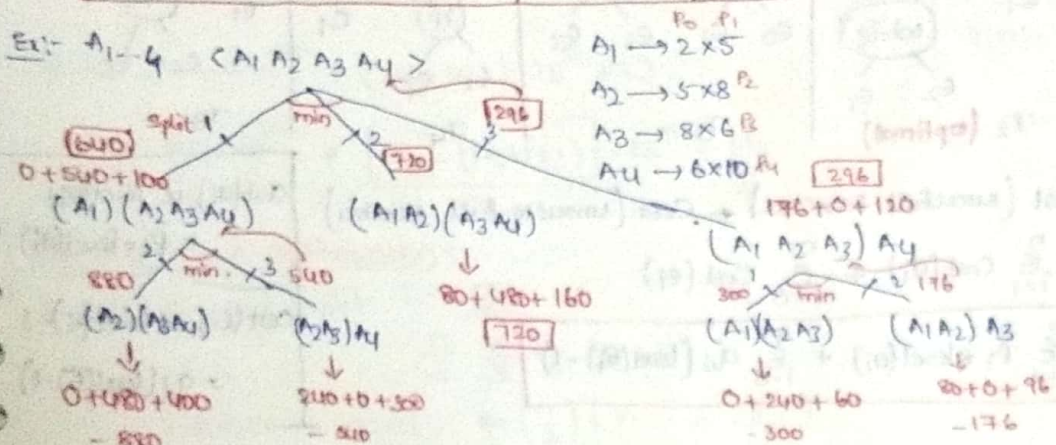
$\langle i \ldots K, K+1 \ldots j \rangle$

$$M[i,j] = \min_{i \le K < j} \{ M[i,K] + M[K+1,j] + P_{i-1} * P_K * P_j \}$$

$M[i,i] = 0 \rightarrow$ Terminating cond.

Ex:- A_{1-4} $\langle A_1 A_2 A_3 A_4 \rangle$

$A_1 \rightarrow 2 \times 5$ ($P_0 P_1$)
$A_2 \rightarrow 5 \times 8$ ($P_1 P_2$)
$A_3 \rightarrow 8 \times 6$ ($P_2 P_3$)
$A_4 \rightarrow 6 \times 10$ ($P_3 P_4$)



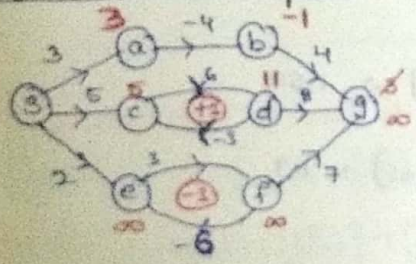
Num of scalar muls = 296,

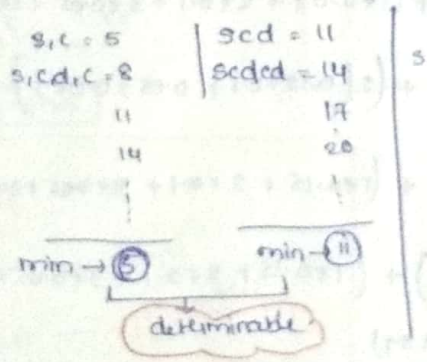
Analysis:

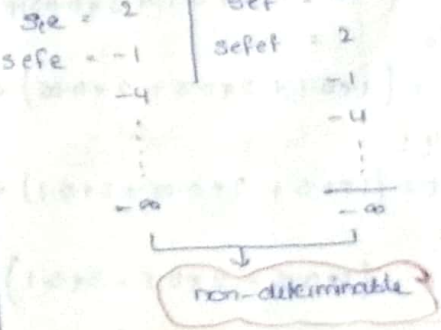
Time Complexity: $O(n^3)$ n: num of matrices in chain

(6) Bellman fold Algorithm

Single Source Shortest path.







*) In case of the graph has positive weight cycles with -ve weight edges then the shortest path to the vertices in the cycle are determinable

*) In case of the graph has -ve weight cycle reachable from source then the shortest path to the vertices in the cycle and the vertices reachable along with the cycle are non determinable

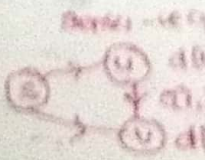
(*) Dijkstra's Algo may fail for negative edges

Algorithm:

```
Algo BF(G, s, c, d)
{
  c[1..n, 1..n], d[1..n]
  1. Init_sce(G, s, d) - O(n)
  2. for i=1 to n-1 - n
     {
       for each edge (u,v) ∈ E - e
         Relax(u,v,c,d)
     }
}
```

O(ne)

```
3. for (each edge (u,v) ∈ E)
     if (d[v] > d[u] + c[u,v])
       return false;
   return true;
}
```



Analysis

Time Comp : O(ne)

For complete graph $e = c(n^2)$

TC: $c(n^3)$

Leftovers

Cost of Binary search tree :

Ex:- n = 3 $\langle a_1\ a_2\ a_3 \rangle$ = $\langle$ do if while $\rangle$

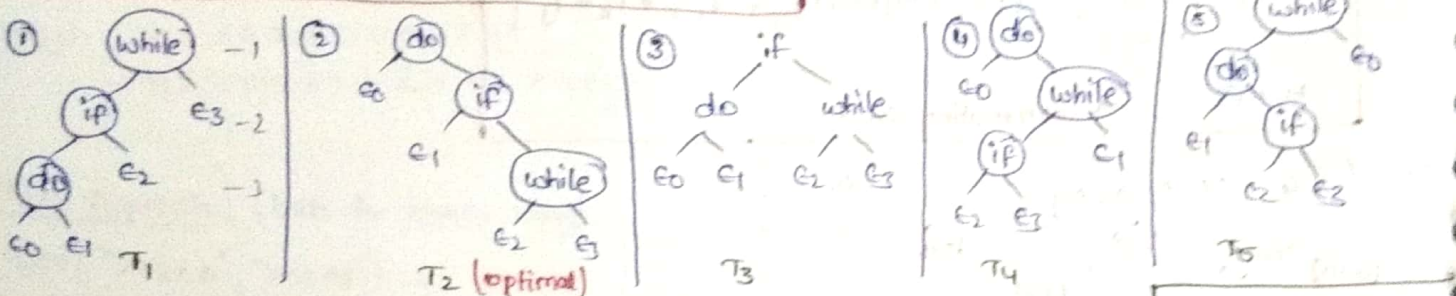
$\langle p_1\ p_2\ p_3 \rangle$ = $\langle 0.5\ 0.1\ 0.05 \rangle$ - successful search

$\langle q_0\ q_1\ q_2\ q_3 \rangle$ = $\langle 0.15, 0.1, 0.05, 0.05 \rangle$ - unsuccessful search

*) Num of distinct BST (catalan num) = $\frac{2n\,C_n}{n+1}$

↓

Distinct BST = $\frac{6C_3}{4}$ = 5



$\therefore$ Cost of BST = Cost (succesfull searches) + Cost (unsuccesfull searches)

$$= \sum_{i=1}^{n} Cost(a_i) + \sum_{i=0}^{n} Cost(e_i)$$

$$Cost(BST) = \sum_{i=1}^{n} p_i * level(a_i) + \sum_{i=0}^{n} q_i\,(level(e_i) - 1)$$

$Cost(a_i) \propto level(a_i)$

$= p_i * level(a_i)$

$Cost(e_i) \propto level(e_i) - 1$

$= q_i\,(level(e_i) - 1)$

For T_1 (succesful search, unsuccesful search)

Cost = $(1*0.05 + 2*0.1 + 3*0.5) + (3*0.15 + 3*0.1 + 2*0.05 + 1*0.05) \Rightarrow 2.65$

For E_2

Cost = $(1*0.5 + 2*0.1 + 3*0.05) + (1*0.15 + 2*0.1 + 3*0.05 + 3*0.05) \rightarrow 1.5$ optimal BST

For T_3

Cost = $(1*0.1 + 2*0.5 + 2*0.05) + (2[0.15 + 0.1 + 0.05 + 0.05]) \Rightarrow 1.9$

For T_4 :

Cost = $(1*0.5 + 2*0.05 + 3*0.1) + (1*0.15 + 2*0.1 + 3*0.05 + 3*0.05) \Rightarrow 1.55$

For T_5

Cost = $(1*0.05 + 2*0.5 + 3*0.1) + (1*0.15 + 2*0.1 + 3*0.05 + 3*0.05) \Rightarrow 1.9$

② Long integer multiplication : (LIM)

Let u and v be two long integers of n digits each represented in an array of size 'n'. Then the problem of LIM is to multiply the two long integers to get the resultant product

long int a,b,c ⟶ certain limitations of size $\xrightarrow{sol^n}$ use arrays

↓

A = B + C : O(n)

A = B * C : $O(n^2)$ (minimize it)

Divide & Conquer approach :

Let u, v be two long int of n digits. $m = \lfloor \frac{n}{2} \rfloor$

Ex:- u = 1234 v = 5678

$= 12*10^2 + 34$ $= 56*10^2 + 78$

$u = w*10^m + x$ | $v = y*10^m + z$

$w = \frac{u}{10^m}$, $x = u \% 10^m$ | $y = \frac{v}{10^m}$, $z = v \% 10^m$

Now,

$u \cdot v = (w*10^m + x)(y*10^m + z)$

$T(n) = 4T(\frac{n}{2}) + bn \; ; n>1$
$= c \; ; n=1$

↓

$O(n^2)$: no change

Karatsuba Multiplication optimization:

let x : n/2 digit num

$x = (w \cdot 10^{n/2}) + y$

$z = (x \cdot 10^{n/2}) + y$

$= wx \cdot 10^n + wy \cdot 10^{n/2} + xy \cdot 10^{n/2} + yz$

$wy + xy = k - (wx + yz)$

let $P_1 = wx$

$P_2 = k$

$P_3 = yz$

$xy = wx \cdot 10^n + (wy + xy) \cdot 10^{n/2} + yz$

$= P_1 \cdot 10^n + [P_2 - (P_1 + P_3)] \cdot 10^{n/2} + P_3$

$T(n) = 3T(\frac{n}{2}) + bn \; ; n>1$
$= c \; ; n=1$

↓

$O(n^{\log_2 3}) \cdot O(n^{1.58})$

Time and Cost optimization:

Case 3 way split:

$\frac{n}{3} \; \frac{n}{3} \; \frac{n}{3}$

i) BFS : $T(n) = 9T(\frac{n}{3}) + n \rightarrow O(n^2)$

ii) MS : $T(n) = 8T(\frac{n}{3}) + n \rightarrow O(n^{\log_3 8}) \cdot O(n^{1.89})$

iii) TC : $T(n) = xT(\frac{n}{3}) + n$

such that

$n^{\log_3 x} < n^{1.58}$

$x = 5$

$T(n) = 5T(\frac{n}{3}) + n \; ; n>1$
$= c \; ; n=1$

↓

$O(n^{\log_3 5}) \cdot O(n^{1.47})$

Ans

Note:

If k way split is used in MM

(i) BFS : $T(n) = k^2 T(\frac{n}{k}) + n$ — $O(n^2)$

(ii) MS : $T(n) = (k^2-1) T(\frac{n}{k}) + n$ — $O(n^{\log_k k^2-1})$

(iii) TC : $T(n) = (2k-1) T(\frac{n}{k}) + n$ — $O(n^{\log_k 2k-1})$

× — The end — ×

No. of possible min heaps : $T(n) = \sum_{k=1}^{n-1} {}^{n-1}C_k \, T(k) \, T(n-k-1)$ where k = num of nodes in left subtree

$T(1) = T(2) = 1$

$T(3) = 2$

Master theorem for sub & conquer:

$T(n) = \begin{cases} c & \text{if } n \le 1 \\ aT(n/b) + f(n) & n>1 \end{cases}$

Where, $a \ge 1$, $b > 0$, $k \ge 0$ and if $f(n)$ is $O(n^k)$ then,

(i) If $a < b^k$ then $T(n) = O(n^k)$

(ii) If $a = b^k$ then $T(n) = O(n^{k+1})$

(iii) If $a > b^k$ then $T(n) = O(n^k a^{n/k})$