Design And Analysis Of Algorithms

URBAN EDGE

Algorithm

It is a procedure or formula for solving a problem based on conducting a sequence of specified actions.

Difference b/w Algorithm & Program

Algorithm	Program
• It is used in design time.	• It is used in implementation time.
• A person who write algo. should have domain knowledge.	• A person who write program should have programming knowledge.
• You can write algo. in any language.	• You can write the program in C, C++, Java etc.
• It is independent of hardware, software and O.S.	• It is dependent on hardware, os.

Difference b/w Priori Analysis & Posteriori Analysis

Priori (Algo.)	Posteriori (Prog.)
→ It is known as Algorithm.	→ It is known as Program.
↓ same as above.	↓ same as above.

Characteristics of Algorithm

There are 5 characteristics of algorithm-

1. Input : It can take 0 or more input.
2. Output : It must generate an output.
3. Definitiveness : Every statement should be unambiguous & it must have exact meaning.
4. Finitness : It must have finit no. of steps & terminate at some point.

5. Effectiveness: We shouldn't write unnecessary statement.

Space & Time Complexity

Performance of analysis can be done on the basis of two things.

1) Space Complexity :- Total amount of memory needed by the algo. for completion of its task.

2) Time Complexity :- Total amount of time needed by the algo for completion of its task.

Space Complexity

It contains 2 parts
- fixed :- It does not depend on another variable. Ex→ Constant
- Variable :- It depends on other variable.

Ex→
```
add (x, y)
{ return (x+y);
}
```

Space Complexity can be calculated by the formula:-

$$C + VD$$

C: fixed part, VD: variable part

Q) How to write an Algorithm?

1. Swapping 2 No.

```
Algorithm Swap(x,y)
begin
    temp = x  → 1
    x = y     → 1
    y = temp  → 1
end
```

Space Complex → C + VD
⇒ 1+1+1 + 0
⇒ 3
⇒ O(1)

* Points to Remember
- No datatypes are required.
- No variable declaration is required.
- No curly braces are required.
- Each & every statement of algo take 1 unit of time.

Q2) Frequency Count Method.

```
Algorithm Sum (A, n)
begin
    s = 0 ------> 1
    for (i=0; i<n; i++) ------> (n+1)
        s = s + A[i] ------> n
    return s ------> 1
end
```

C + WD
⇒ 1 + n+1 + n + 1
⇒ 3 + 2n
⇒ O(n)

Q3) Addition of 2D Array. (Same Size) $n \times n$

```
Algorithm Sum of 2D (A, n)
begin
    s = 0 ------> 1
    for (i=0; i<n; i++) ------> (n+1)
        for (j=0; j<n; j++) ------> n*(n+1)
            s = s + A[i][j] ------> (n*n)
    return s ------> 1
end
```

∴ 1 + n+1 + n^2 + n + n^2 + 1
⇒ 3 + 2n + $2n^2$
⇒ $O(n^2)$ Ans.

URBAN EDGE

Q. Multiplication of a Matrix

```
Algorithm Multiplication (A, B, n)
begin
    for (i=0; i<n; i++)  → (n+1)
        for (j=0; j<n; j++)  → (n+1)*n
            c[i][j] = 0;  → n*n
        for (k=0; k<n; k++)  → n²(n+1)
            c[i][j] = A[i,k] + B[k,j];  → n³
end
```

So, $T(n) = O(n^3)$

$S(n) = O(n^2)$ → $A = n^2$, $B = n^2$, $C = n^2$, $i = j = k = 1$

Some important concept of finding time complexity of for loop.

1. `for (i=0; i<n; i++)` → (n+1)
   `{ stmnt; }` → n
   - $T(n) = O(n)$

2. `for (i=n; i>0; i--)` → (n+1)
   `{ statement; }` → n
   - $T(n) = n+1+n$
     $= (2n+1)$
     $= O(n)$

3. `for (i=1; i<n; i=i+2)` → (n/2 + 1)
   `{ statement; }` → (n/2)
   - $T(n) = \frac{n}{2} + 1 + \frac{n}{2} \Rightarrow O(n)$

4. `for (i=0; i²<n; i++)` → (√n + 1)
   `{ statement; }` → (√n)
   - $T(n) = O(\sqrt{n})$

5. `for (i=0; i<n; i++)` → (n+1)
   `for (j=0; j<n; j++)` → n*(n+1)
   `{ statement; }` → n*n
   - $T(n) = O(n^2)$

p=0

6.) `for(i=0; p<=n; i++)`
`{`
`p = p+i;`
`}`

i	p
0	0
1	0+1
2	0+1+2
3	0+1+2+3
⋮	⋮
n	0+1+2+3+...n

AP ⇒ $\frac{k(k+1)}{2}$

$\because p > n$

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

$k^2 + k > 2n$

$k > \sqrt{2n-k}$

$\boxed{T(n) = O(\sqrt{n})}$ $S(n) = O(\sqrt{n})$

7.) `for(i=1; i<n; i=i*2)`
`{`
`statement;`
`}`

i	
1	$\longrightarrow 2^0$
2×2	$\longrightarrow 2^1$
2×2	$\longrightarrow 2^2$
2×2×2	$\longrightarrow 2^3$
⋮	⋮
2^k	$\longrightarrow 2^k$

$\because i >= n$

$2^k >= n$

$\log 2^k >= \log n$

$k >= \log_2 n$

- $\boxed{T(n) = O(\log_2 n)}$

8.) `for(i=n; i>=1; i=i/2)`
`{`
`statement;`
`}`

- $\boxed{T(n) = O(\log_2 n)}$

9.)
```
for (i=1; i<n; i=i*2)
{
  p++;          → log n
  for(j=1; j<p; j=j*2)
  {
    statement;  → log p
  }
}
```

- $T(n) = O(\log_2 n * \log_2 p)$
  $= O[\log(n+p)]$

10.)
```
for(i=0; i<n; i++)        → (n+1)
{
  for(j=1; j<n; j*=2)     → log n
  {
    statement;            → (n * log n)
  }
}
```

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

Analysis of If-else

1.)
```
if (x%2==0)  → 1
  even       → 1
else
  odd        → 1
```

$T(n) = O(1)$

Types of Time Complexity

1. $O(1)$ = Const.
2. $O(\log n)$ = Logrithmic
3. $O(n)$ = Linear
4. $O(n^2)$ = Quadratic
5. $O(n^3)$ = Cubic
6. $O(2^n)$ = Exponential

URBAN EDGE

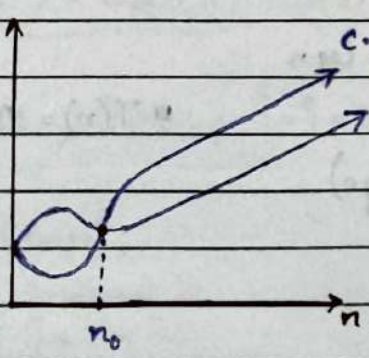
Growth Function

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

Assymptotic Notation

For a program we can implement more than one algo. of that single program so for analysing that which algo. takes less time, we can use the concept of assymptotic notation.

1. Big (oh) :- It can be represented as [O]



C = Constant
n = Input
t = Time

- If $f(n) <= C \cdot g(n)$ $\forall$ $n \geq n_0$ then $C > 0$, and not $>= 1$
  therefore $f(n) = O(g(n))$

Q. let $f(n) = 3n + 2$
$g(n) = n$ where $c = 4$? then we can represent

$$f(n) = O(g(n))$$! (CHECK ATLEAST 4 VALUE) !

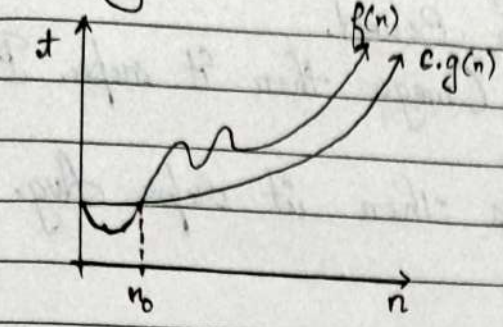
Sol. Let $n = 1$
$f(n) = 5$
$C \cdot g(n) = 4 \times 1 = 4$

Let $n = 2$
$f(n) = 8$
$C \cdot g(n) = 8$
→ (same) $(n_0 = 2)$

Let $n = 3 \Rightarrow f(n) = 11$ & $C \cdot g(n) = 12$

→ Yes, we can represent $f(n) = O(g(n))$.

2. Big Omega (Ω)



- If $f(n) \geq c \cdot g(n)$ $\forall$ $n \geq n_0$ then $c > 0$, and not ≥ 1
- therefore $f(n) = \Omega(g(n))$

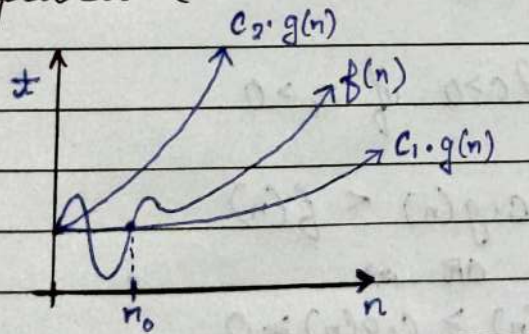
Q.) Let $f(n) = 3n+2$, $g(n) = n$ then we can write $f(n) = \Omega(g(n))$ $c = 1$

Sol.

Let $n=1$	$n=2$	$n=3$	$n=4$
$f(n) = 5$	$f(n) = 8$	$f(n) = 11$	$f(n) = 14$
$c \cdot g(n) = 1$	$c \cdot g(n) = 2$	$c \cdot g(n) = 3$	$c \cdot g(n) = 4$

So, we can write $f(n) = \Omega(g(n))$

3.) Big Theta (Θ)



- If $c_1 g(n) \leq f(n) \leq c_2 g(n)$ where $c_1, c_2 > 0$ $\forall$ $n \geq n_0$, $n_0 \geq 1$ then $f(n) = \Theta(g(n))$

Q.) Let $f(n) = 3n+2$, $g(n) = n^2$ where $c = 1$ then $f(n) = \Omega(n^2)$?

Let $n=1$	$n=2$	$n=3$	$n=4$
$f(n) = 5$	$f(n) = 8$	$f(n) = 11$	$f(n) = 14$
$c \cdot g(n) = 1$	$c \cdot g(n) = 4$	$c \cdot g(n) = 9$	$c \cdot g(n) = 16$

∴ No we can't write $f(n) = \Omega(g(n))$

URBAN EDGE

Important Points

- If we are using Big Oh then it represent Worst Case Time and Upper Bound.
- If we are using Omega then it repr. Best Case & Lower Bound.
- If we use Theta then it repr. Avg. Case.

Little Oh Notation (o)

$$o(g(n)) = \{ f(n) : \forall\ c>0 \text{ if } n_0>0 \}$$

$$0 \le f(n) < c \cdot g(n) \quad \forall\ n>n_0$$

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

Little Omega Notation (ω)

$$\omega = \{ f(n) : \forall c>0 \text{ if } n_0>0$$

$$0 \le c \cdot g(n) < f(n)$$

OR

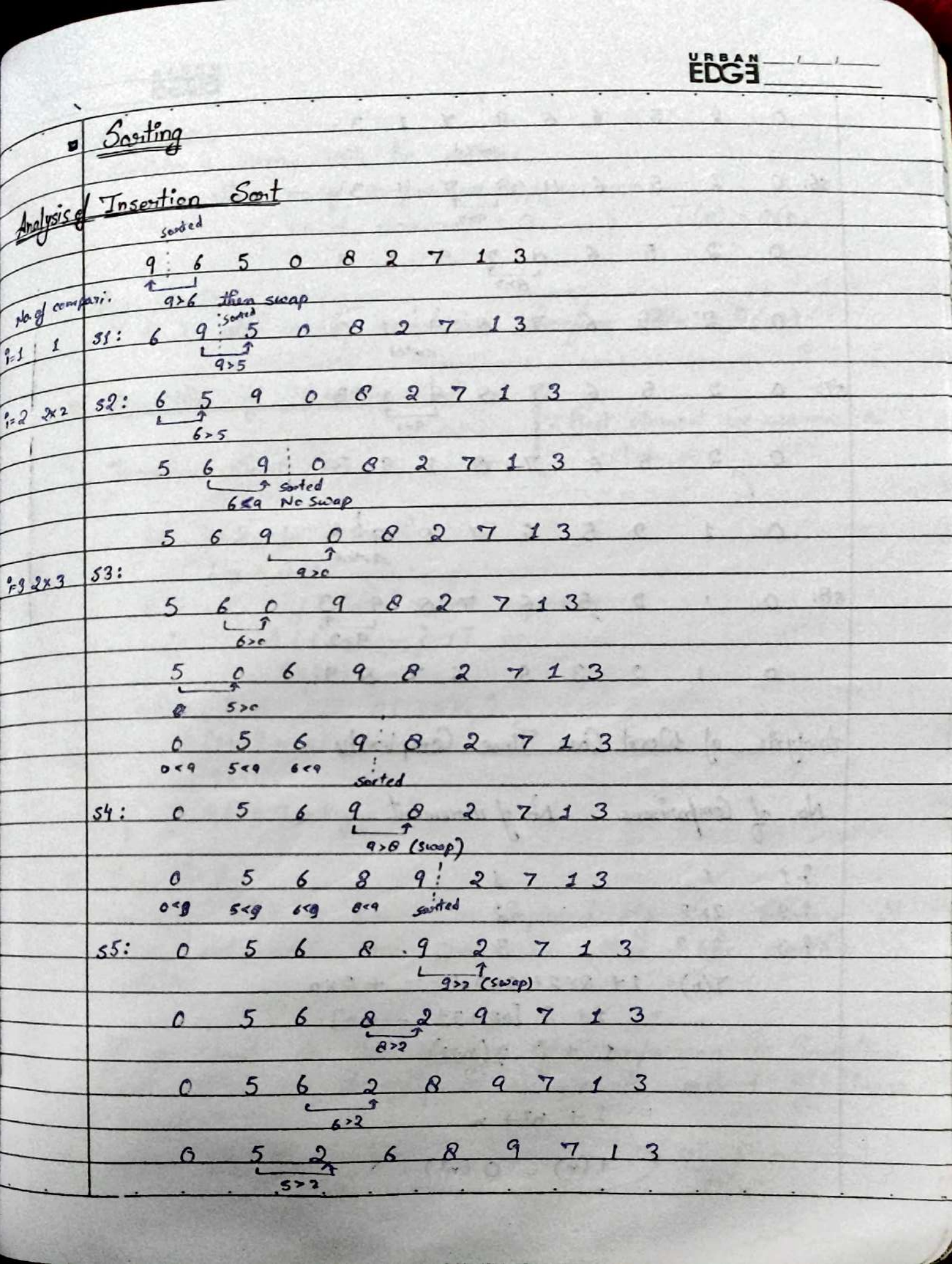
$$f(n) > c \cdot g(n) \ge 0$$

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

URBAN EDGE

Sorting

Analysis of Insertion Sort



0 2 5 6 8 9 | 7 1 3
sorted

s6: 0 2 5 6 8 9 7 1 3
9>7

0 2 5 6 8 7 9 1 3
8>7

0 2 5 6 7 8 9 | 1 3
sorted

s7: 0 2 5 6 7 8 9 1 3
9>1

0 2 5 6 7 8 1 9 3

0 1 2 5 6 7 8 9 | 3
sorted

s8: 0 1 2 5 6 7 8 9 3
9>3

0 1 2 3 5 6 7 8 9

Analysis of Worst Case Time Complexity

	No. of Comparisons	No. of movement
i=1	1	1
i=2	2×2	2
i=3	2×3	3

$$T(n) = 1 + 2\times2 + 2\times3 + \cdots + 2\times n$$
$$= 1 + 2[1+2+3+\cdots+n]$$
$$= 1 + 2\,\frac{n(n+1)}{2}$$
$$= 1 + n^2 + n$$
$$T(n) = O(n^2)$$

Best Case:-

Consider:- Array will be sorted.

then no. of comparison = 0, 1, 1 ... n

no of movement = 0 $T(n) = O(n)$

Avg. Case:-

It is also nearby worst case therefore $T(n) = \theta(n^2)$

- **Algorithm of Insertion Sort**

{ ∵ first element we assume as sorted.

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

- **Imp. Points**

- Insertion Sort is known as inplaced algorithm because it does not need any space to store the another variable.

- If we want to dec. the no. of comparison in insertion sort then we apply binary search instead of linear search.

No. of Compar. $O(\log n)$ No. of Movement $O(n)$

URBAN EDGE

- If we want to dec. the no. of movement then we applied doubly linked list (instead of normal move.)

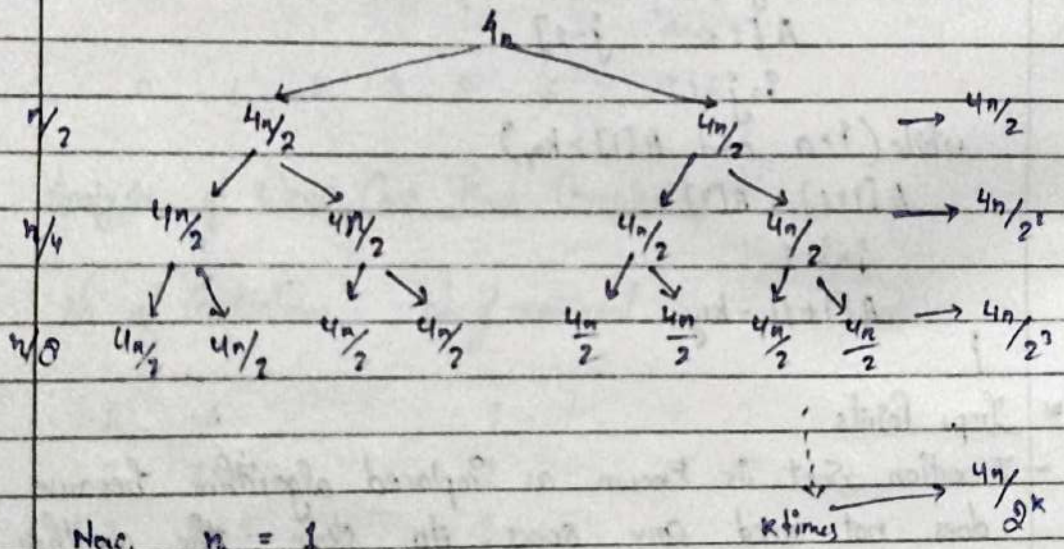
No. of Comparision	No. of movement
$O(n)$	$O(1)$

→ But these both concept can't applied simultaneously.

→ Space Complexity → $O(n)$
↳ due to size of array

- Solving recurrence relation using recurrence tree method

Q1) $T(n) = 2 \times T(n/2) + 4n$ and $T(1) = 4$
Solve this question by using Recurrence Tree Method.



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

$n = 2^k$

$k = \log_2(n)$

$\therefore T(n) = n \log_2 n$

$4 + 4 + 4 + \dots + n \text{ times} = n$

$= O(n \log_2 n)$

URBAN EDGE

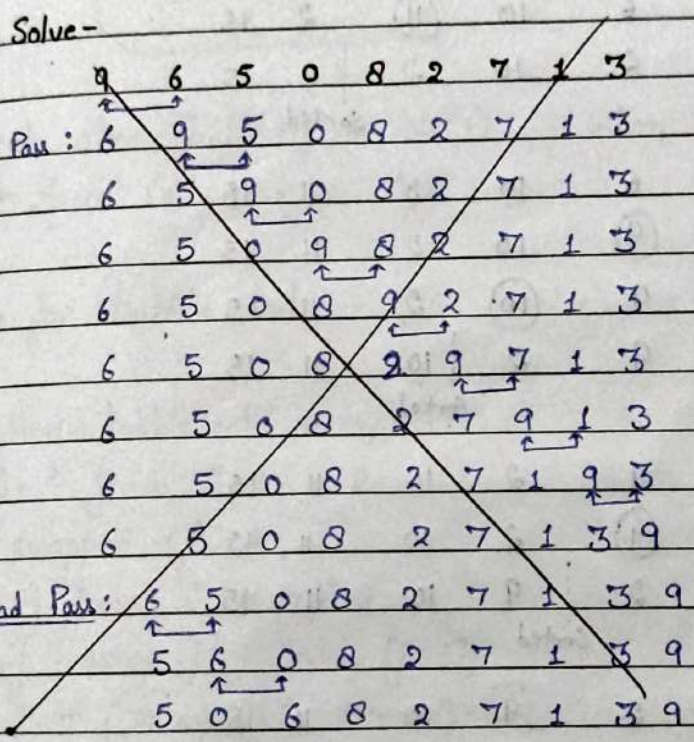
Assignment

1. Write an algo of Bubble Sort and analysis Worst, Best & Avg time complexity & find space complexity.

- It sort the array by swapping the adjacent elements if they are in the wrong order.

- This algo. is not suitable for large data set due to its time complexity.

Algo Solve -



P.T.O

URBAN EDGE

Algo. Solve –

0	1	2	3	4	5
10	9	11	6	15	2

Compare 10>9 (Swap)

Pass 1

9	10	11	6	15	2
9	10	6	11	15	2
9	10	6	11	15	2
9	10	6	11	2	15

sorted

Pass 2

9	10	6	11	2	15
9	6	10	11	2	15
9	6	10	11	2	15
9	6	10	2	11	15

sorted

Pass 3

9	6	10	2	11	15
6	9	10	2	11	15
6	9	10	2	11	15
6	9	2	10	11	15

sorted

Pass 4

6	9	2	10	11	15
6	9	2	10	11	15
6	2	9	10	11	15

sorted

Pass 5

6	2	9	10	11	15
2	6	9	10	11	15

Sorted Array

∵ Every time of comparison, no. of comp. decrease.

URBAN EDGE

Analysis of Time Complexity of Bubble Sort

- At each iteration, no. of comparison decrease. $(n-1, n-2, \dots 1)$

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

So,

Worst time Complexity = $O(n^2)$

Avg. time Complexity = $\theta(n^2)$

$S = 1 \; 2 \; 3 \; 4 \; 5 \; \dots\dots \; (n-2) \; (n-1)$

$S = (n-1) \; (n-2) \; \dots\dots \; 2 \; 1$

$2S = n \; n \; \dots\dots \; n \; n$

$$S = \frac{n(n-1)}{2}$$

∵ Best time Complexity = $\Omega(1)$ ∵ When array is sorted.

↳ Space Complexity = $O(n)$

Algo for bubble sort

```
function bubblesort( array)
  for i from 0 to n-2
    swapped = false
for j from 0 to n-i-1;
  if array [j] > array [j+1];
  swap ( array [j], array [j+1])
    swapped = true;
if not swapped
  break.
```

URBAN EDGE

Solving recurrence relation with iterative method

Steps to solve the problem:-
Step 1: Unfold the recurrance until you see the pattern.
Step 2: When pattern is identified then find the specific form.

Q) $T(n) = 8T\left(\frac{n}{2}\right) + n^2$, $T(1) = 1$ (i) Solve using iterative method.

Let $n = \frac{n}{2}$, put this value in eq^n (1)

we get $T\left(\frac{n}{2}\right) = 8T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2$ — (ii)

Put (ii) in (i)

$$T(n) = 8\left[8T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2\right] + n^2 \quad \text{— (iii)}$$

Let $n = \frac{n}{4}$ and put in eq^n (1)

we get $T\left(\frac{n}{4}\right) = 8T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2$ — (iv)

Put (iv) in (iii)

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

$$\vdots$$

$$T\left[\frac{n}{2^k}\right] +$$

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

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

$n = 2^k$

$k = \log_2 n$

$8^n \log_2 n$

$8^1 + 8^2 + 8^3 + \dots + 8^n \Rightarrow 8^n$

URBAN EDGE

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

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

$$= 8^n \cdot \log_2(n) + \log_2(n)$$

$$= 2^{3n} \log_2 n$$

$$= 2 \log_2 n^{3n}$$

$$= O(n^{3n})$$

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

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

$n = 2^k$

$k = \log_2 n$

Solving recurrance relation using substitution method

Q.) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ n \times T(n-1) & \text{if } n > 1 \end{cases}$ — (1)

(Solve using substitution)

Put $n = (n-1)$

$T(n) = (n \quad T(n-1) = (n-1)\, T(n-2)$ — (2)

Now from (1)

$$T(n) = n\left[(n-1)\, T(n-2)\right] \quad \text{— (3)}$$

Put $n = (n-2)$

$T(n-2) = (n-2)\, T(n-1)$ — (4)

Now put (4) in (3)

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

--->

$$T(n) = n\,(n-1)\,(n-2)\,(n-3)\, T(n-4) \cdots = n!$$

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

Imp

Master's Theorm

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

where $a \geq 1$, $b > 1$, $k \geq 0$ and P is real number.

Conditions to be checked:-

Case1: if $a > b^k$, then $T(n) = \theta\left(n^{\log_b a}\right)$

Case2: if $a = b^k$, then

a.) if $P > -1$, then
$$T(n) = \theta\left(n^{\log_b a} \cdot \log^{(P+1)} \cdot n\right)$$

b.) if $P = -1$, then
$$T(n) = \theta\left(n^{\log_b a} \cdot \log \log n\right)$$

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

Case3: if $a < b^k$, then

a.) if $P \geq 0$, then
$$T(n) = \theta\left(n^k \cdot \log^P n\right) (\log n^P)$$

b.) if $P < 0$, then
$$T(n) = \theta(n^k)$$

Q1) $T(n) = 3T(n/2) + n^2$

$a = 3, \quad b = 2, \quad k = 2, \quad P = 0$

Now, $a < b^k$ bcz $a = 3, \; b^k = 2^2 = 4$

$3 < 4$

So we go with case 3 ($a < b^k$) & $P = 0$

$$T(n) = \theta(n^k \cdot \log n^P)$$

$$\boxed{T(n) = \theta(n^2)}$$ Ans.

Q2) $T(n) = 4 \cdot T(n/2) + n^2$

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

$\boxed{a = b^k} \Rightarrow \boxed{4 = 4}$ So, Case 2 where $P = 0$

$$T(n) = \theta\left(n^{\log_b a} \cdot (\log^{P+1} n)\right)$$

$$= \theta\left(n^{\log_2 2^2} \cdot (\log n)^1\right)$$

$$\boxed{T(n) = \theta(n^2 \cdot \log n)}$$ Ans

Q3) $T(n) = T(n/2) + n^2$

$a = 1, \quad b = 2, \quad k = 2, \quad P = 0$

$$T(n) = \theta(n^k \cdot \log n^P)$$

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

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

Q4) $T(n) = 2^n \cdot T(n/2) + n^n$

$a = 2^n, \quad b = 2, \quad k = n, \quad P = 0$

$\boxed{a = b^k}$ $2^n = 2^n$ Case 2 $\Rightarrow T(n) = \theta\left(n^{\log_b a} \cdot (\log n)^{P+1}\right)$

Can't Apply MT, 15

$$= \theta\left[n^{\log_2 2^n} (\log n)^1\right]$$

$$\Rightarrow \theta(n^n \cdot \log n)$$ Ans

Q5) $T(n) = 16T(n/4) + n$

$a = 16,\ b = 4,\ K = 1,\ P = 0$

$a > b^K \Rightarrow 16 > 4$

Case 1: $P = 0$

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

$= \theta(n^2)$ Ans

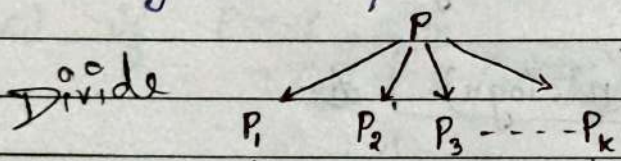
Q6) $T(n) = 0.5T(n/2) + 1/n$

$a = 0.5,\ b = 2,\ K = -1,\ P = 0$

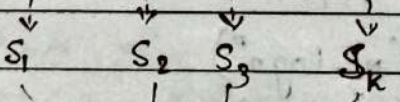
Here $a < 1$ & $K < 0$ So not possible.

Divide & Conqueror

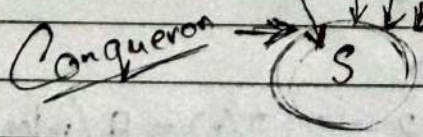
- Let us consider a large problem P and size of n.
- And the problem is divided into sub problem according to your choice, let say k.



- After that we find sol^n of each sub problem.



- Then finally after finding the sol^n of each sub problem, we can combine all sub solution.



URBAN EDGE

Generalise Algo.

```
DAC (P)
{
  if (small(P))
  {
    S(P);
  }
  else
  {
    Divide P into P1, P2, ...., Pn.
    Apply DAC to DAC(P1), DAC(P2), ... DAC(Pn)
    Combine DAC(P1), DAC(P2), ..... DAC(Pn)
  }
}
```

What are the problem that can be solved using DAC meth

→ Binary Search, ~~Bubble~~ Sort, Quick Sort, finding Min/Max
Merge

H.W

Q. Write an Algo. of Max/Min Problem?

URBAN EDGE

Binary Search

3	6	8	12	14	17	25	29	Key = 17
1	2	3	4	5	6	7	8	

S1: l = 1
h = 8
mid = (l+h)/2
= (1+8)/2
= 9/2
= 4.5 ≃ 5

S2: Case 1
arr[mid] = key
arr[5] = 17
14 = 17 ×

Case 2
arr[mid] > key
arr[5] > key
14 > 17 ×
l = mid - 1

Case 3
arr[mid] < key
arr[5] < key
14 < 17 ✓
h = mid + 1
= 5+6

mid = (6+8)/2
= 14/2
= 7

arr[7] == 17
25 == 17
×

25 > 17
h = 7 - 1
= 6

mid = (6+6)/2
= 12/2
= 6

arr[6] == 17
14 == 17

- If you want to find the no. of comparison can be done in binary search :-

  No. of comparison = $\log_2(n+1)$ $\quad \because n \rightarrow$ no. of elements

- Time taken by Binary Search is :- $\log(n)$

- If we are searching for an element which are not present in an array. Then also no. of comparison = $\log_2(n+1)$

- Best Case = $O(1)$
- Worst Case = $O(\log n)$

Q) Write an algo of recursive Binary Search?

(Imp)

* Merge Sort → Merge Sort

1	2	3	4	5	6	7	8
1	5	7	8	2	4	6	9
p			q				r

p = first index = 1
q = mid ⇒ (p+r)/2
r = last index = 8

A_1

1	5	7	8	∞
i=1	2	3	4	5

A_2

2	4	6	9	∞
j=1	2	3	4	5

Case 1: $A_1[i] == A_2[j]$: Terminate the loop

Case 2: $A_1[i] > A_2[j]$

⇒ 5 > 2 $j = j+1$

Case 3: $A_1[i] < A_2[j]$

⇒ $q = i+1$

1	2	4	5	6	7	8	9

▸ Algorithm

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

MERGE (A, P, q, r)
{
     n1 = q-p+1;
     n2 = r-q;
   Let L[1, ...., n1+1] and R[1 to n2+1]
   be new arrays.
        for (i=1 to n2)
            L[i] = A[p+i-1]
```

```
for (j=1 to n2)
    R[j] = A[q+j]
L[n1+1] = ∞
R[n2+1] = ∞

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

Q) Why merge sort is known as out of place algorithm?
→ Bcz we need another array to store the sorted element.

▸ Space Complexity
- Due to merge sort algo. $S(n) = \log n$.
- Due to merge procedure $S(n) = n$

$\therefore$ $\boxed{S(n) = O(n)}$

▸ Time Complexity

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

Case 2 (ii) $T(n) = O(n^{\log_b a} \cdot \log^{p+1} n)$

$= O(n \log n)$

Quick Sort

- Explanation

(i=-1) 9 6 5 0 8 2 4 7
∴ j=0

- pivot element = last element = 7
- i = initially does n't point anything.
- j = first index.

Case 1: j > P
9 > P
j++

Case 2: 6 ≤ P
6 ≤ 7
swap i & j
i++

6 9 5 0 8 2 4 7
i=1
j=1 j=2

6 5 9 0 8 2 4 7
i=2
j=3

6 5 0 9 8 2 4 7
i=3
j=4 j=5

6 5 0 2 8 9 4 7
i=4 j=5 j=6

6 5 0 2 4 9 8 7
i=5 j=7

6 5 0 2 4 7 8 9
i=6 j=7

Smaller value. Larger value.

STEPS:-

- Let us consider last element as pivot element (P=7)
- Let us consider 2 pointers i & j, then j will scan the element from left to right one by one.
- Case 1: If j > pivot then increment j by 1.

Case 2: If j ≤ pivot then swap the value of i & j & i++.

URBAN EDGE

Algorithm of Quick Sort

```
Quick_sort (A, p, r)                    (q → pivot element.)
{
   if (p < r)
   {
      q = partition (A, p, r);  ──→ O(n)
      Quick_sort (A, p, q-1);   ──→ T(n/2)
      Quick_sort (A, q+1, r);   ──→ T(n/2)
   }
}
Partition (A, p, r)
{
   x = A[r];
   i = p-1;
   for (j=p to r-1)
   {
      if (A[j] ≤ x)
      {  i++;
         exchange (A[i] with A[j]);
      }
   }
   exchange (A[i+1] with A[r]);
   return i+1;
}
```

URBAN EDGE

Analysis of Time Complexity

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

$a = 2,\ b = 2,\ k = 1,\ P = 0$

Master's theorm

$a = b^k,\ P > -1$ ✓

Case 2:

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

$$\boxed{T(n) = \theta(n \cdot \log n)} \rightarrow \text{Best / Avg. Case}$$

Worst Case

$T(n) = T(n-1) + n$ — (i)

By substitution method

$T(n-1) = T(n-2) + n - 1$ — (ii)

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

$= T(n-2) + 2n - 1$ — (iii)

$T(n-2) = T(n-3) + n - 2$ — (iv)

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

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

⋮

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

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

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

i) If the inputs are in ascending order then $T(n) = O(n^2)$ (fixed pivot)

ii) If the inputs are in descending order then $T(n) = O(n^2)$

iii) If the elements are like 2 2 2 2 then $T(n) = O(n^2)$

* Space Complexity of Quick Sort

Case 1: If array is balanced then $S(N) = O(\log n)$ Best / Avg. Case

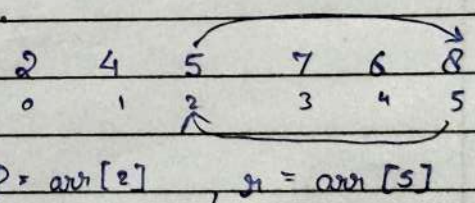
Case 2: If array is unbalanced then $S(N) = O(n)$ Worst Case

Randomised Quick Sort

Step 1: Initially array is sorted & by swapping we have to make an unsorted array.

Step 2: We can swap any random value b/w p & r.

Step 3: Let us consider index 2 is a random number swap with the last element (8).



p = arr[2] , r = arr[5]

S4: Then choose last element as a pivot & apply quick-sort partition algorithm.

Randomized Quick Sort Algo:-

```
Randomized QS(A, P, r)
{
  if (p<r)
    {
      i ← random (p, r);
      swap (A[p], A[r]);
      q ← partition (A, p, r);
      Randomised QS (A, p, q-1);
      Randomised QS(A, q+1, r);
    }
}
  Partition (A, p, r)
  {
    Same as QS;
  }
```

Time Complexity : T(n)

GREEDY METHOD

URBAN EDGE

It is an approach or design for solving a problem which is of similar type.

Step1: Let say a problem $P: A \rightarrow B$

P: bbs travel to home

A ⟶ B

Step2: So, there are many solution to travel from $\{A \rightarrow B\}$

S3: There is some constraints to reach the desired result such as.

- feasible
- Min. Cost & Time

- **Algo. of Greedy**

```
Algorithm Greedy (a,n)
{
  for i=1 to n do
  {
     x = select(a);
     if feasible(x) then
        solution = solution + x;
```

Knapsack Problem (Storage Related Problem in a Bag)

Steps to solve this problem are-

1. Find the ratio of profit & weight & then select the item having maximum ratio.
2. We can solve this type of problem by using 3 approach.
   i) Maximum Profit
   ii) Min. Weight
   iii) Max. Profit/Weight (Knapsack problem).

P/ω : 5 3.33 3 1.75 8 3 2

Q1) Object (O): 1 2 3 4 5 6 7
Profit (P): 5 10 15 7 8 9 4 (Given wt = 15)
Weight (w): 1 3 5 4 1 3 2

i) Maximum Profit

n = 7, max. wt = 5 (Given wt = 15)
max. profit = 15

No. of objects	Max. Profit	Weight	Remaining Wt.
3	15	5	15-5 = 10
2	10	3	10-3 = 7
6	9	3	7-3 = 4
5	8	1	4-1 = 3
4	$7 \times \frac{3}{4} = 5.25$	3	3-3 = 0

Max. Profit = 15 + 10 + 9 + 8 + 5.25 = 47.25 Ans

URBAN EDGE

i) Maximum Weight

No. of objects	Profit	Max. Weight	Remaining Wt. (kg)
3	15	5	15 - 5 = 10
4	7	4	10 - 4 = 6
2	10	3	6 - 3 = 3
6	9	3	3 - 3 = 0

Max. Profit = 15 + 7 + 10 + 9 = 41

ii) Minimum Weight

No. of Objects	Profit	Min. Weight	Remaining Wt. (kg)
5	8	1	15 - 1 = 14
1	5	1	14 - 1 = 13
7	4	2	11
2	10	3	8
6	9	3	5
4	7	4	1
3	$\frac{15 \times 1}{5} = 3$	1	0

Max. Profit = 8 + 5 + 4 + 10 + 9 + 7 + 3 ⇒ 46

iii) Max. P/w.

No. of object	Max. Profit	Weight	Remaining
5	8	1	15 - 1 = 14
1	5	1	14 - 1 = 13
2	10	3	13 - 3 = 10
3	15	5	10 - 5 = 5
6	9	3	5 - 3 = 2
7	4	2	2 - 2 = 0

Σ profit ⇒ 51

Activity Selection Problem (Job Scheduling)

Steps to solve this problem:

1. Sort the activities as per finishing time in asc. order.
2. Select the first activity.
3. Selection of new activity if its starting time is greater or equal to the previous selected activity's finish time.
4. Repeat Step 3 till all activities are checked.

Algo. of Activity Selection Problem

```
Greedy_Activity_Selection (s, f)
{
    n ← length [s]
    A = {1}
    j = 1
    for i=2 to n
        do if si ≥ fi
            then A = A ∪ {i}
                j = i
            return A
    }
```

Time Complexity

- If the given activity are in sorted order acc. to finish time then $T(n) = O(n)$.
- If not in sorted order then $T(n) = O(n \log n)$

Q)

Activity	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
Start Time	1	0	1	4	2	5	3	4
Finish Time	3	4	2	6	9	8	5	5

Solve this using activity selection problem?

Ans Step 1: Sort in Asc. wrt finish Time (If same then smaller St. Time)

Activity	a_3	a_1	a_2	a_7	a_8	a_4	a_6	a_5
Start Time	1	1	0	3	4	4	5	2
finish Time	2	3	4	5	5	6	8	9

Step 2: Select the first activity from above table & check for ($s_i \geq f_i$)

Activity	a_3	a_7	a_6
s_i	1	3	5
f_i	2	5	8

$1 \geq 2$ ×
$0 \geq 2$ ×
$3 \geq 2$ ✓
$4 \geq 5$ ×
$4 \geq 5$ ×
$5 \geq 5$ ✓

Huffman Coding

- It is a compression technique.
- It is used for reducing the size of data & msg.

Q) Message : BCCABBDDAECCBBAEDDCC
Cal. the cost of this msg. to store in a file.
length = 20.

		ASCII	Binary	Size/
1st Method ⇒	A	65	01000001	Cost of Message
	B	66	01000010	⇒ length × bits
	C	67	01000011	= 20 × 8
	D	68	01000100	= 160 bits
	E	69	01000101	

2nd Method

Message	Freq.	Name	
A	3	000	0
B	5	001	1
C	6	010	2
D	4	011	3
E	2	100	4
	20		

∵ Here we have 5 chars. So, we use 3 bit to represent.

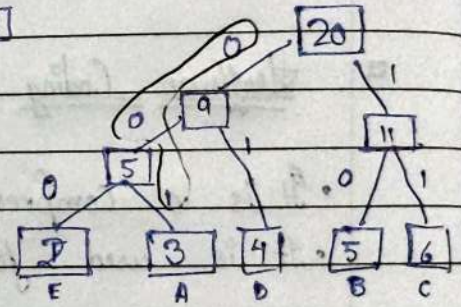
∵ Size of the message = length x bits
= 20 x 3
= 60

∵ Max. size of the character ⇒ 5 x 8 = 40 bits (Character)
Min. size " " " ⇒ 5 x 3 = 15 bits (Code)

Cost of the message = 60 + 40 + 15
= 115 bits.

Huffman Coding

Method 3rd ⇒ E(2) A(3) D(4) B(5) C(6)



- Sort character acc. to their frequency.
- Merge the smaller two each time. (optimal merge)
- Mark left edge as 0 & right as 1.
- follow the code from top to bottom.

Msg.	freq.	Code	freq. of code
A	3	001	3 x 3 = 9
B	5	10	2 x 5 = 10
C	6	11	2 x 6 = 12
D	4	01	2 x 4 = 8
E	2	000	3 x 2 = 6
	20	(add each bit = 12)	Σ → 45 bits

Size of msg. = 45 bits
Max size of char = 5 x 8 = 40
Max size of code = 12
97 bits

$\sum d_i * f_i$ → from tree

3x2 + 3x3 + 2x4 + 2x5 + 2x6 =
6 + 9 + 8 + 10 + 12 ⇒ 45

Algo. Huffman Coding

```
Huffman (c)
{
   n = |c|
   Q = c
   for i=1 to n-1
     do
        z = allocate_Node ()
        x = left(z) = Extract_Min(Q)
        y = right(z) = Extract_Min(Q)
          f(z) = f(x) + f(y)
            Insert (Q, z)
        return Extract_Min(Q);
}
```

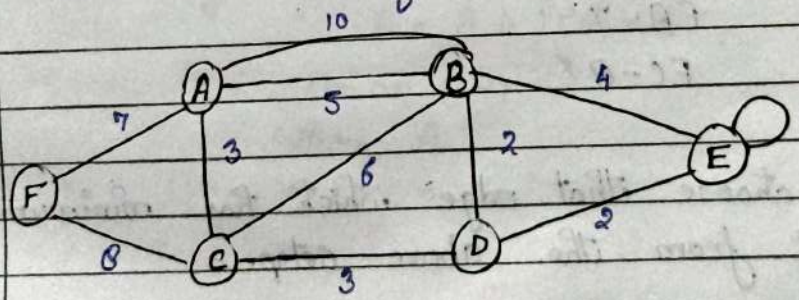
Therefore, time complexity ⇒ O(nlogn) {Best, Worst, Avg}

GRAPH

Kruskal's Algo.

It is used to find minimum spanning tree. (MST)

Q.)

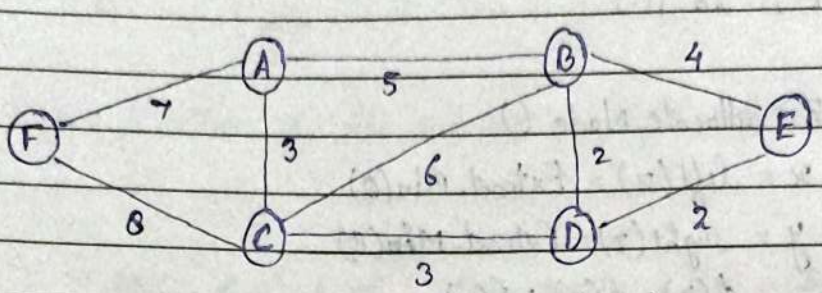


Find the MST using Kruskal's Algo?

URBAN EDGE

Step1: Remove all the loops & parallel edges of the graph.

Step2: Remove that edge, who has max. edge weight & keep the min. edge weight.

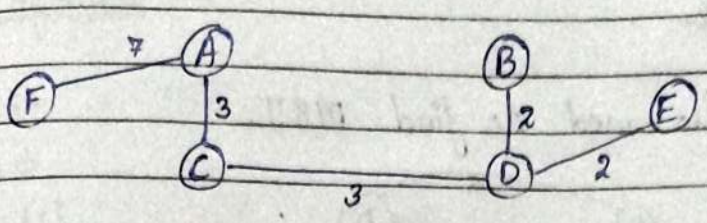


Step3: In this algo. we will arrange all the edges according to their edge weight (ascending order).

BD - 2 ✓
DE - 2 ✓
AC - 3 ✓
CD - 3 ✓
BE - 4 ✗ (form cycle)
AB - 5 ✗ (")
CB - 6 ✗ (")
FA - 7 ✓
FC - 8 ✗

Step4: We will choose that edge which has minimum edge weight from the above setup.

Step5: By construct this graph you should maintain that this MST does not contain any cycle.



Step 1: If graph contains n no. of vertices then MST of that graph contain n no. of vertices and (n-1) no. of edges.

MST { n = 6 (Vertices), n-1 = 6-1 = 5 (Edges) } MST = 17

Time Complexity ⇒ $O(E \log E)$ or $O(V \log V)$ ∵ E/V = vertices

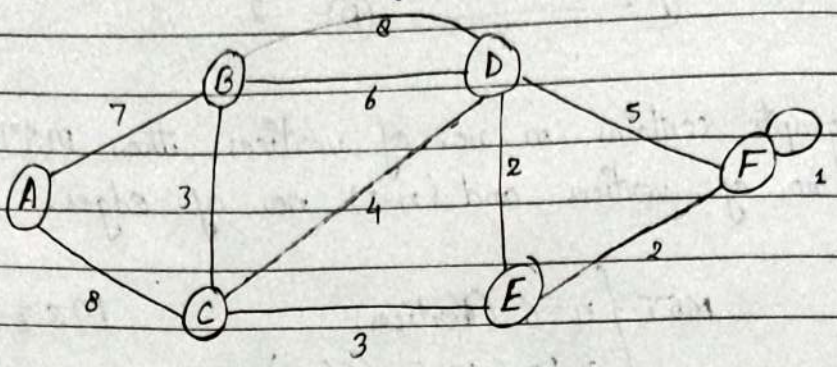
- Kruskal's Algo.

```
Kruskal (G)
{
   A = ϕ
   for each vertex u ∈ G.V;
      Make_Set (V)
   for each edge (u, v) ∈ G.E ordered
      if find_set(u) ≠ find_set(v):
            A = A ∪ {(u, v)}
         Union (u, v)
         Return A
}
```

Prim's Algo.

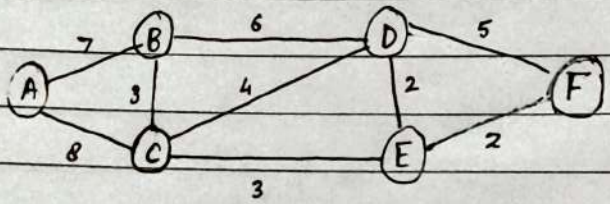
It is also used to find MST.

Q.)



Step1 : Remove all the loops & parallel edges

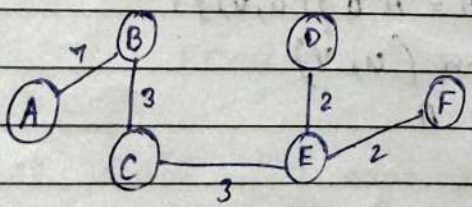
Step2: Remove that parallel edges which has max edge weight & keep the minimum



Step3: Choose any arbitrary node i.e. consider as root node.

Step4: Check all the outgoing edges from the root node and then choose the minimum edge weight.

Step5: Repeat this S4 until all the vertices are traversed.



MST = 7+3+3+2+2 = 17

URBAN EDGE

Prim's Algorithm

```
Prim(G)
{ T = ϕ
  u = {1}
  while (u ≠ v)
  let (u,v) be the lowest cost edge such that
    u ∈ v and  v = v - u
      T = T ∪ {(u,v)}.
      u = uv {v}
      return T
}
```

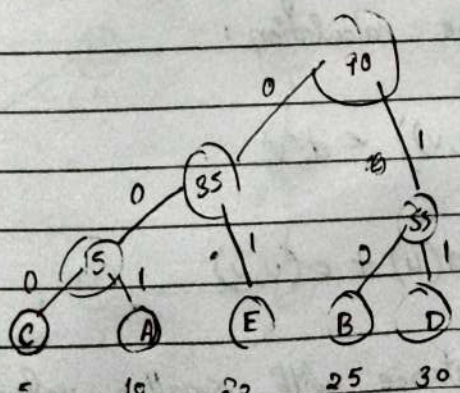
Therefore, Time Complexity : $O(E \log V)$

(∵ E → Edge, ∵ V → Verifices)

Q) Huffman Coding

Character	Count	Code	Freq. of Code
A	10	001	3×10 = 30
B	25	10	2×25 = 50
C	5	000	3×5 = 15
D	30	11	2×30 = 60
E	20	01	2×20 = 40
	90	12	195

1 bit = 4 unit



- Size of msg = ~~90×8 = 196~~
- Max. char = 100 × 8 = 8000
- Size of code = 12

8207

Average code length = $\frac{\text{Size of the msg.}}{\text{Count}}$

$= \frac{195}{90} = 2.167$

Total bits in Huffman Coding of 1000 char = 2.167×1000

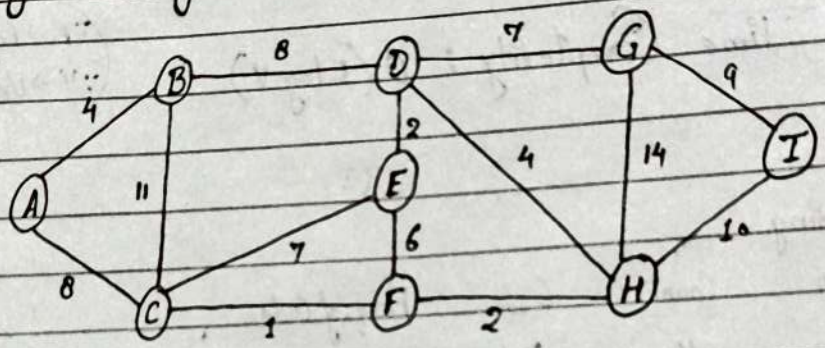
$= 2167$

Now,

Cost of 4 bits = 2167×4

$= 8668$

Dijkstra Algorithm



→ It is also known as single source shortest path.

→ Remove loops & parallel paths

Step 1: Consider A as a source node.

Step 2: Initialise with zero as a source node.

Step 3: Put the initial value as ∞ of all other nodes.

Step 4: formula used for distance calculation:

$$\text{if } (d(u) + c(u,v)) < d(v)$$

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

Step 5: Update the vertices distance with smaller value.

URBAN EDGE

Limitations of Dijkstra

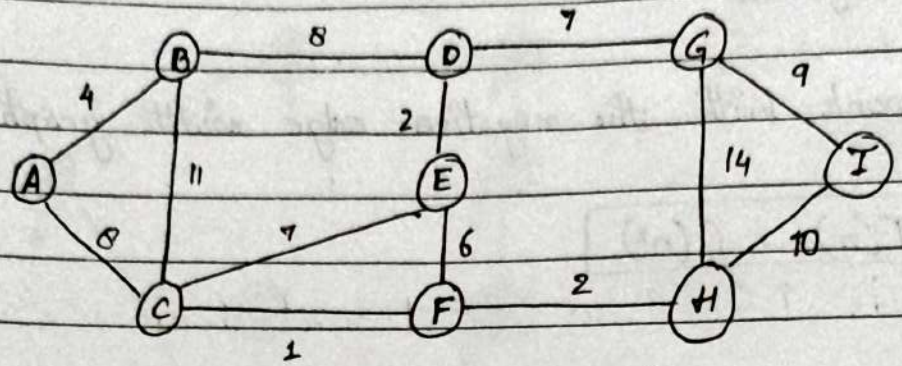
→ It does not work with the negative edge width graph having cycle.

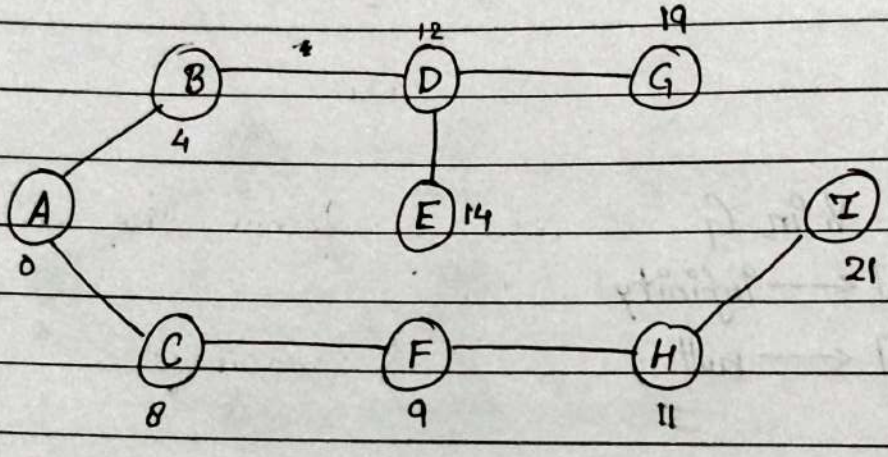
$$T(n) = O(n^2)$$

Algorithm

```
Dijkstra (G, s)
for each vertex V in G
    distance [v] ← infinity
    previous [v] ← null
    if v != s
      distance [s] ← 0
while Q is not empty
    u ← extract Min from Q
for each unvisited neighbour
    v of u
tempDistance ← dis[u] + cost [u,v]
if  dist [v] ← tempDistance
    previous [v] ← u
return distance[], previous[].
```

Correct Dijkstra Algo.





Dynamic Programming

URBAN EDGE 27 Aug 24

→ It is mainly an optimization over recursion.
→ This idea is to satisfy simply store the result of subproblem so that we don't have to recompute them when needed later.

→ Ex. Matrix Chain Multiplication & Longest Common Subsequence

Q: Diff. b/w Divide & Conqu & Dynamic Prog.

Divide & Conquere	Dynamic Programming
• Partition of problem into independent smaller subproblem	• Partition of problem into overlapping of subproblems.
• Doesn't store sol. of subproblem	• Store sol^n of subproblem.
• Top-Down Algorithm.	• Bottom-Up Algorithm
• Ex→ Merge, Quick, Binary Search.	• Ex → Matrix CM, 0-1 Knapsack.

Matrix Chain Multiplication

Let A & B are 2 matrix and want to multiply them
A no. of column $\neq$ B no of rows

$$A[\ ]_{4\times3} = B[\ ]_{3\times2}$$

Q: A_1, A_2, A_3 and A_4 are 4 matrices solve using the concept of MCM
$A_1 \rightarrow 5\times4$, $A_2 = 4\times6$, $A_3 = 6\times2$, $A_4 = 2\times7$

Sol: Draw

P.T.O →

$A_1 \rightarrow 5 \times 4$ (d_0 d_1) $A_3 \rightarrow 6 \times 2$ (d_2 d_3)

$A_2 \rightarrow 4 \times 6$ (d_1 d_2) $A_4 \rightarrow 2 \times 7$ (d_3 d_4)

URBAN EDGE

S1: Draw the table

M:

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

S:

	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

S2: formula Used

$$m[i,j] = \min \{ m[i,k] + m[k+1,j] + d_{i-1} * d_k * d_j$$

Given: $d_0 = 5$

$d_1 = 4$

$d_2 = 6$

$d_3 = 2$

$d_4 = 7$

S3: $m[1,2]$ (k value starts from i to $j-1$)

$i=1, j=1, k=2$

$m[1,2] = m[1,1] + m[2,2] + d_0 \times d_1 \times d_2$

$= 0 + 0 + 5 \times 4 \times 6 = 120$

$m[2,3] = m[2,2] + m[3,3] + d_1 \times d_2 \times d_3$

$= 48$

URBAN EDGE

- $m[1,3]$ $(i=1, k=1, j=3) \Rightarrow m[1,1] + m[2,3] + d_0 \times d_1 \times d_3 = 88$

  $(i=1, k=2, j=3) \Rightarrow m[1,2] + m[3,3] + d_0 \times d_2 \times d_3 = 180$

- $m[2,4] = m[2,2] + m[3,4] + d_1 \times d_2 \times d_4$

  $= 0 + 84 + 4 \times 6 \times 7 = 252$

$(i=2, k=2, j=4)$

$(i=2, k=3, j=4) \Rightarrow m[2,3] + [4,4] + d_1 \times d_3 \times d_4$

$= 104$

- $m[1,4] = m[1,1] + m[2,4] + d_0 \times d_1 \times d_4$

$i=1$	$\Rightarrow 0 + 104 + 5 \times 4 \times 7$
$j=4$	$\Rightarrow 244$
$k=1$	

$i=1$	$m[1,2] + m[3,4] + d_0 \times d_2 \times d_4$
$k=2$	$\Rightarrow 120 + 84 + 5 \times 6 \times 7$
$j=4$	$\Rightarrow 414$

$i=1$	$m[1,3] + m[4,4] + d_0 \times d_3 \times d_4$
$k=3$	$\Rightarrow 158$
$j=4$	

⇒ Priority Distribution

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

MCM Algorithm

```
Main ()
{
  int n=5
  int p[] = {5, 4, 6, 2, 7}
  int m[5][5] = {0}
  int j, min, q;
  int s[5][5] = {0}
for (int d=1; d<n-1; d++)
  {
    for (int i=1; i<n-d; i++)
    {
      j = i+d;
      min = 0;
      for (int k=i; k<=j-1; k++)
      {
        q = m[i][k] + m[k+1][j] + P[i-1] x P[k] x P[j];
        if (q<min)
        {
          min = q;
          s[i][j] = k;
        }
      }
    }
  }
}
cout << m[i][n-1];
}
```

Therefore, $T(n) = O(n^3)$

URBAN EDGE

Lowest Common Subsequence

Let us we have 2 String:-

Case 1:
String 1 - a, b, c, d, e f g h i
String 2 - c d g i
and then find String 2 characters is common to String 1 in same sequence or not.

LCS of above String ⇒ cdgi
Size = 4

Case 2:
Str 1 = a b c d e f g h i
Str 2 = ~~e~~ c d g i

∵ → egi is subsequence but cegi is a LCS of this String.
→ d is not considered as LCS.

Q.) Solve the problem by using LCS.

Str 1: b d
Str 2: a b c d

Same X ⇒ Dia.

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

So, LCS = b d
& Size = 2

[* Longest char ⇒ columns
* Smallest char ⇒ rows]

URBAN EDGE

Q2.) String 1: Stone
String 2: LONGEST

		L	O	N	G	E	S	T
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
S 1	0	0	0	0	0	0	1	1
T 2	0	0	0	0	0	0	1	2
O 3	0	0	1	1	1	1	1	2
N 4	0	0	1	2	2	2	2	2
E 5	0	0	1	2	2	3	3	3

LCS ⇒ ONE
Size ⇒ 3

- Time Complexity = $O(m \times n)$

Q3.) String 1: abcdefghi
String 2: ecdgi

		a	b	c	d	e	f	g	h	i
	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
e 1	0	0	0	0	0	0	1	1	1	1
c 2	0	0	0	1	1	1	1	1	1	1
d 3	0	0	0	1	2	2	2	2	2	2
g 4	0	0	0	1	2	2	2	3	3	3
i 5	0	0	0	1	2	2	2	3	3	4

LCS ⇒ cdgi
Size ⇒ 4

URBAN EDGE

Previous Year

Q1. $T(n) = 4T(n/2) + cn$

$a = 4, \quad b = 2, \quad K = 1$

Master Theorem

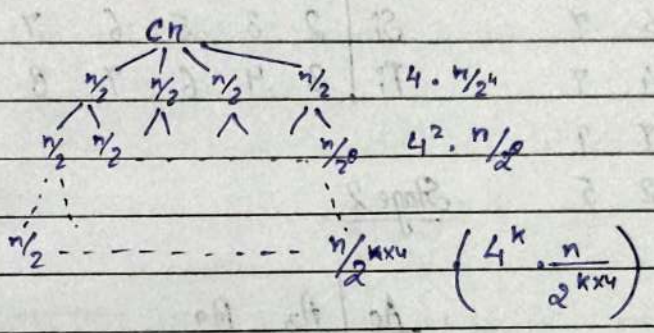
$$T(n) = aT(n/b) + \theta(n^K \cdot \log^P n)$$

$a > b^K$

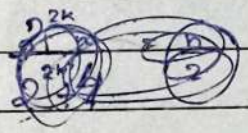
Case 1:

$$T(n) = \theta(n^{\log_b^a}) = \underline{\theta(n^2)}$$

Recurrence Tree



$$\rightarrow \frac{4^k \cdot n}{2^{4k}} \qquad \therefore T(n) = \theta(n^2)$$



Q2. $T(n) = T(n-1) + 2^n$ —(i) (Using Iterative)

$T(n-1) = T(n-2) + 2^{n-1}$ —(ii)

$T(n) = T(n-2) + 2^{n-1} + 2^n$ —(iii)

$T(n-2) = T(n-3) + 2^{n-2}$ —(iv)

$T(n) = T(n-3) + 2^{n-2} + 2^{n-1} + 2^n$ —(v)

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

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

$T(n) = 2^{n-3} + 2^{n-2} + 2^n + \cdots + 2^1$

$= 2^n$

$T(n) = \theta(2^n) = \underline{O(2^n)}$

- Merge
- Quick
- Insertion

URBAN EDGE

Q) Activity Selection

Acti	Si	fi
A_1	2	3
A_2	3	4
A_3	2	5
A_4	8	9
A_5	5	6
A_6	7	8
A_7	6	7
A_8	4	7
A_9	7	9
A_{10}	2	5

→

Ac	A_1	A_2	A_3	A_{10}	A_5	A_8	A_7	A_6	A_9	A_4
Si	2	3	2	2	5	4	6	7	7	8
fi	3	4	5	5	6	7	7	8	9	9

Stage 1

Ac	A_1	A_2	A_5	A_7	A_6	A_4
Si	2	3	5	6	7	8
fi	3	4	6	7	8	9

Stage 2

Ac	A_3	A_9
Si	2	7
fi	5	9

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

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

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

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

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

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

URBAN EDGE

Q: $T(n) = T(n-2) + 2^n$ by using Master Theorm

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

$= T(2^{n-2}) + 2^n$

Let $2^n = m$

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

$a=1, b=4, k=1, P=0$

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

10/Sep/24

- Algorithm of LCS:-

```
int Longest (LCS[][], i, j)
{
    for (int i=0 ; i<=m; i++)
    {
        for (int j=0;  j<=n; j++)
        {
            if (A[i] == B[j])
                LCS[i,j] = 1 + LCS[i-1, j-1]
        }
    }
}
```

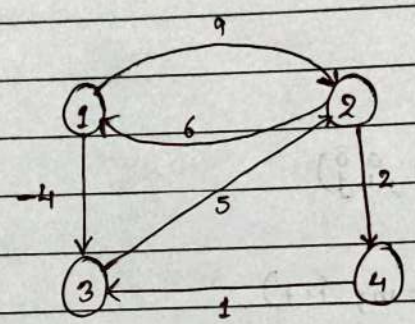
- $T(n) = O(m \times n)$

FLOYD WARSHALL

- It is used to find all pair shortest path.
- In this algo we use to find shortest path of all vertices from root node.
- By using this algo. we can work with negative edge weight having cycles.

~~Steps~~

Q.) Solve this graph using floyd warshall.



Step1: Consider one node as starting node (ie. 1)

Step2: Draw the distant matrix.

		1	2	3	4
	1	0	9	-4	∞
$D^0 \rightarrow$	2	6	0	∞	2
	3	∞	5	0	∞
	4	∞	∞	1	0

URBAN EDGE

Step 3: Put the diagonal element as 0 bcz at the node there is no self-loop.

Step 4: If you don't have any direct edge from the source node to the destination node then put (∞).

Step 5: Consider one as a middle vertex & then calculate the shortest distance b/w each and every node via vertex one.

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

D^1 =

	1	2	3	4
1	0	9	-4	∞
2	6	0	2	2
3	∞	5	0	∞
4	∞	∞	1	0

- first check ∞ value
- Freeze 1st R&C

⇒ $D^1[2,3] = \min\left(D^0[2,3],\ D^0[2,1] + D^0[1,3]\right)$

$i=2 \quad = \min(\infty,\ 6 + (-4))$

$k=1 \quad = \min(\infty, 2)$

$j=3 \quad = 2$

$D^1[3,4] = D^0[3,1] + D^0[1,4] = \infty + \infty = \infty$

$D^1[4,2] = D^0[4,1] + D^0[1,2] = \infty + 9 = \infty$

⇒ $D^2[1,4] = D^2[1,2] + D^1[2,4] = 9 + 2 = 11$

$D^2[3,1] = D^1[3,2] + D^1[2,1] = 5 + 6 = 11$

$D^2[3,4] = D^1[3,2] + D^1[2,4] = 5 + 2 = 7$

$D^2[4,1] = D^1[4,2] + D^1[2,1] = \infty + 6 = \infty$

$D^2 =$

	1	2	3	4
1	0	9	-4	11
2	6	0	2	2
3	11	5	0	7
4	∞	∞	1	0

$\Rightarrow D^3[4,1] \Rightarrow D^2[4,3] + D^2[3,1] = 1 + 11 = 12$

$i = 4$

$j = 1$

$k = 3$

$D^3[4,2] \Rightarrow D^2[4,3] + D^2[3,2] = 1 + 5 = 6$

$D^3 =$

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

$D^4 \Rightarrow$

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

Therefore, $T(n) = O(n^3)$

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

Algorithm for floyd warshall

n = no. of vertices
A = Matrix of dimension n×n

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

0-1 Knapsack Method

Q) P = {1, 2, 5, 6}
w = {2, 3, 4, 5}
Max. wt. = 8
No. of object = 4

Pr.	wt.		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	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	6	7	8

max wt. →
↑ No. of objects

Step 1: Initially assign 0 in first row & column
Step 2: If weight match with capacity of the bag then assign the value of profit on that weight.
Step 3: Formula used:

$$V[i, w] = \max \{ V[i-1, w], V[i-1, w - w(i)] + P[i] \}$$

$$V[3, 5] = \max \{ V[2, 5], V[2, 5-4] + 5 \}$$
$$\{ 7[3], 0+5 \} \Rightarrow 5$$

Step 4: If we get -ve weight set same previous value.
Step 5: Repeat S3 & S4 until all vertices are traversed.

x_1	x_2	x_3	x_4
↑	↑	↑	↑
0	1	0	1

Rem. max profit = 8 − 6 = 2
max. profit = 2 − 2 = 0

Ans ⇒ {0101}

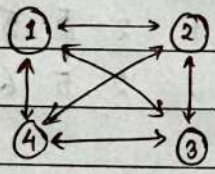
Algorithm

Dynamic 0-1 Knapsack (v, w, n, W)

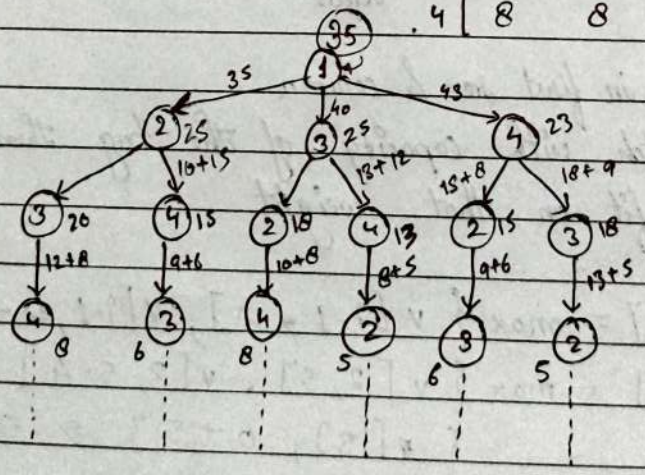
```
for w = 0 to W do
    c[0, N] = 0
for i = 1 to n do
    c[i, 0] = 0
for w = 1 to W do
    if wi ≤ w then
    if vi + c[i-1, w-wi] then
        c[i, w] = vi + c[i-1, w-wi]
    else
        c[i, w] = c[i-1, w]
```

Time Complexity → O(n.w)

Travelling Salesman Problem



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



So, min cost = 35

URBAN EDGE

Step 1: We have to find the starting vertex and through that we have to traverse all vertex and return to the starting vertex.

Step 2: Let say 1 as a starting node

Step 3: We will solve this problem from bottom-up approach that's why it is known as brute force method.

Step 4: formula : $\min \{ C_{ik} + g(k, \{2,3,4\} - \{k\}) \}$

where . $k = \{2, 3, 4\}$

Step 5: Repeat Step 3 & 4 until all vertices are traversed.

$g(2, \phi) = 5$
$g(3, \phi) = 6$
$g(4, \phi) = 8$

$g(2, \{3\}) = 15$
$g(2, \{4\}) = 18$
$g(3, \{2\}) = 18$
$g(3, \{4\}) = 20$
$g(4, \{2\}) = 13$
$g(4, \{3\}) = 15$

$g(2, \{3,4\}) = 25$
$g(3, \{2,4\}) = 25$
$g(4, \{2,3\}) = 23$

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

URBAN EDGE

Algorithm for Travelling Salesman

```
TSP()
{
  c({1}, 1) = 0
  for s = 2 to n do
    for all subsets S ∈ {1, 2, 3, ... n} of size S and
    containing 1.
      c(S, 1) = ∞
  for (S, j) = min { c(S - {j}, i) + d(i, j)  for
      i ∈ S and i ≠ j
```

Return min $c(\{1,2,3,\ldots n\}, j) + d(j,i)$

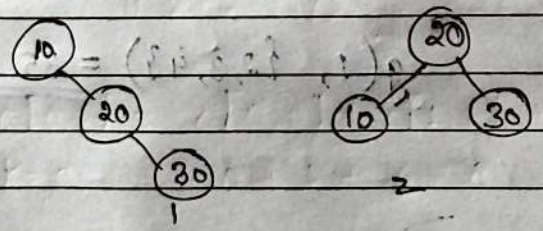
- Time Complexity :: $T(n) = O(2^n \cdot n^2)$

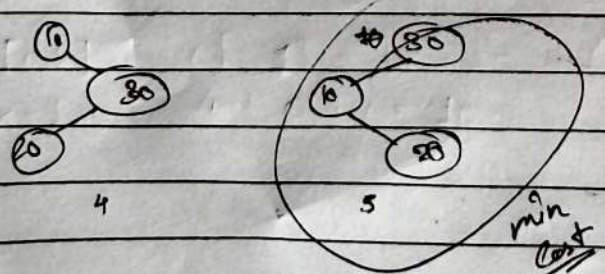
05/Oct/2024

Optimal Binary Search (Rough)

Keys = 10, 20, 30

freq = 5 3 2





$\frac{2^n C_n}{n+1}$

URBAN EDGE

Optimal Binary Search Tree

Formula: $\frac{^{2n}C_n}{n+1}$

→ For n keys how many binary search tree possible:

$^nC_r = \frac{n!}{r!(n-r)!}$

$\frac{^{2n}C_n}{n+1} = \frac{^6C_3}{(3+1)} \Rightarrow \frac{6\times5\times4\times3\times2\times1}{3!\,3!\times4} = 5$ (n=3)

Q.) Activitys 1 2 3 4

Keys	10	20	30	40
Frequency	4	2	6	3
index →	0	1	2	3

Solve above problem by using optimal binary search tree:

→ $\frac{^{2n}C_n}{n+1} \Rightarrow n=4 = \frac{^8C_4}{5} = \frac{8!}{4!\times4!\times5} = \frac{8\times7\times6\times5}{4\times3\times2\times1\times5} = 14$

i/j	0	1	2	3	4
0	0	4^1	8^2	20^2	26^3
1		0	2^2	10^3	16^3
2			0	6^3	12^3
3				0	3^4
4					0

Step 1: $j-i=0$

$0-0=0$

$1-1=0$

$2-2=0$

$3-3=0$

$4-4=0$

Generalised formula

$$c[i,j] = \min\left\{ c[i,k] + c[k+1,j] + w(i,j) \right\}$$

$$w(i,j) = \sum_{i=0}^{n-1} f(i)$$ (f → freq)

k = i to j-1

URBAN EDGE

Step 2: $j-i=1$ (i,j)

$1-0=1$ $(0,1)$

$2-1=1$ $(1,2)$

$3-2=1$ $(2,3)$

$4-3=1$ $(3,4)$

- $c[0,1] = c[0,0] + c[1,1] + w(i,j)$

  $k=0 \Rightarrow 0+0+4 \Rightarrow 4$

- $c[1,2] = c[1,1] + c[2,2] + w(1,2)$

  $k=1 = 0+0+2=2$

- $c[2,3] = c[2,2] + c[3,3] + w[2,3]$

  $k=2 = 0+0+6 \Rightarrow 6$

- $c[3,4] = c[3,3] + c[4,4] + w(3,4)$

  $k=3 = 0+0+3 \Rightarrow 3$

Step 3: $j-i=2$ (i,j)

$2-0=2$ $(0,2)$

$3-1=2$ $(1,3)$

$4-2=2$ $(2,4)$

- $c[0,2] = c[0,0] + c[1,2] + w(0,2)$

  $k=0 = 0+2+6=8$

$c[0,2] \Rightarrow c[0,1] + [2,2] + w(0,2)$

$k=1 \Rightarrow 4+0+6=10$

URBAN EDGE

- $c[0,3] = c[1,1] + c[2,3] + w(1,3)$

  $k=1 \quad = \quad 0 + 6 + 8 = 14$

$c[1,3] = c[1,2] + c[3,3] + w(1,3)$

$k=2 \quad = 2 + 0 + 8 = 10$

- $c[2,4] = c[2,2] + c[3,4] + w(2,4)$

  $k=2 \Rightarrow 0 + 3 + 9 = 12$

  $c[2,4] = c[2,3] + c[4,4] + w(2,4)$

  $k=3 \quad = 6 + 0 + 9 = 15$

Step 4: $j - i = 3 \quad (i,j)$

$3-0 = 3 \quad (0,3)$

$4-1 = 3 \quad (1,4)$

- $c[0,3] = c[0,0] + c[1,3] + w(0,3)$

  $k=0 \Rightarrow 0 + 10 + 12 = 22$

  $c[0,3] \Rightarrow c[0,1] + c[2,3] + w(0,3)$

  $k=1 \Rightarrow 4 + 6 + 12 \Rightarrow 22$

  $c[0,3] \Rightarrow c[0,2] + c[3,3] + w(0,3)$

  $k=2 \Rightarrow 8 + 0 + 12 \Rightarrow 20$

- $c[1,4] \Rightarrow c[1,1] + c[2,4] + w(1,4)$

  $k=1 \quad = 0 + 12 + 12 = 24$

  $c[1,4] \Rightarrow c[1,2] + c[3,4] + w(1,4)$

  $k=2 \Rightarrow 2 + 3 + 12 \Rightarrow 17$

  $c[1,4] \Rightarrow c[1,3] + c[4,4] + w(1,4)$

  $k=3 \Rightarrow 10 + 0 + 12 = 22$

Step 5: $j-i = 4$ (i,j)

$4-0 = 4$ $(0,4)$

$c[0,4] = c[0,0] + c[1,4] + w(0,4)$

$k=0 \Rightarrow 0+16+15 = 31$

$c[0,4] = c[0,1] + c[2,4] + w(0,4)$

$k=1 \Rightarrow 4+6+15 = 31$

$c[0,4] = c[0,2] + c[3,4] + w(0,4)$

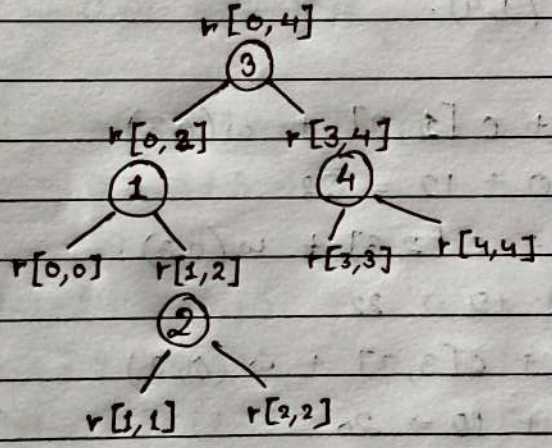
$k=2 = 8+3+15 = 26$

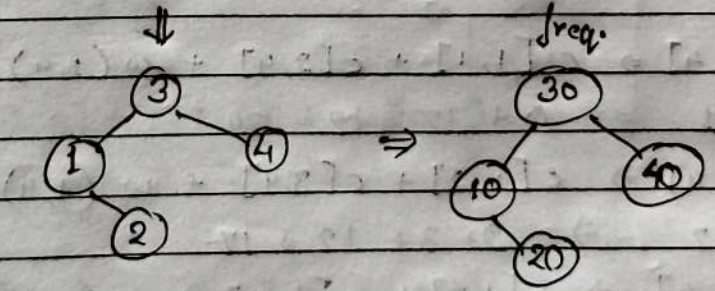
$c[0,4] = c[0,3] + c[4,4] + w(0,4)$

$k=3 \Rightarrow 20+0+15 = 35$

(root node)

$c(0,4)$ is a minimum cost of binary tree which is 26, here tree is constructed by using the value of $k+1$.





Time Complexity ⇒ $O(n^3)$

Algorithm for Optimal BST

```
OBST (p, q, n)
for size in 1 .... n loop
  for i in 1 .....(n-size+1) loop
      j = i + size - 1
      e(i,j) = float max;
      for r in i .... j loop
      t = e(i, r-1) + e(r+1, j) + w(i,j)
        if (t < e(i,j)) then
                    (i,j) = t
                  root (i,j) = r
```

- Time Complexity ⇒ $O(n^3)$

- Algorithm for Optimal BST

```
OBST (p, q, n)
for size in 1 .... n loop
  for i in 1 .... (n-size+1) loop
     j = i + size - 1
     e(i,j) = float max;
     for r in i .... j loop
     t = e(i, r-1) + e(r+1, j) + w(i,j)
       if (t < e(i,j)) then
                 (i,j) = t
               root (i,j) = r
```

Amortized Analysis

- It means finding average running time per operation over worst case sequence of operations.

- It does not allow random choices.
  Eg. Making a cup of coffee for 'n' person.

There are several techniques :-

i) Aggregate Method : It means we have to calculate Time Comp. by using the concept of average.
   Eg. We have to calculate T.C. of Quick Sort
   $Q.S. = O(n \log n)$
   But here,
   $$Q.S. = \frac{O(n \log n)}{n} = O(\log n)$$

ii) Accounting Method: Executing the op. in sequence & manage in this way so that execution time is same as actual time.

Eg.	(1)	(2)	(3)	(4)	(5)	
Let time taken:	2	2	4	3	2	= 13
Actual time:	2	4	2	2	3	= 13

iii) Potential Method: We can manage the op. by using potential energy of ∀ col. T.C.

Q) Define amortized analysis. Explain the amortized complexity for 4 Bit binary increment from 0 to 8 & also write algo.

A)

	b_1	b_2	b_3	b_4	Cycle
0	0	0	0	0	0
1	1	0	0	0	1
2	0	1	0	0	2
3	1	1	0	0	1
4	0	0	1	0	(3)
5	1	0	1	0	1
6	0	1	1	0	2
7	1	1	1	0	1
8	0	0	0	1	(4)

Potential Method

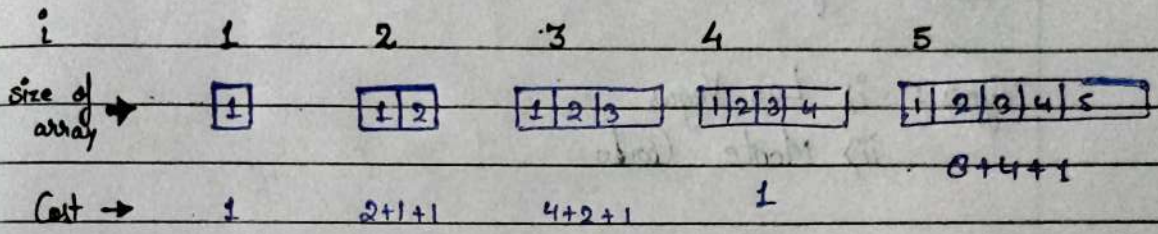
Total no. of cycle:
⇒ 4 × 8 (Bit ← , → Highest no.)
= 32

TC ⇒ O(1)
Worst Case ⇒ O(n)

URBAN EDGE

Worst Case Analysis

Array Insertion Cost
- Array is not full = 1.
- Array is full
  - Size should be double
  - \+
  - Copy All element
  - \+
  - Element Insertion



$$\text{Avg. Case} = \frac{(1) + (2+1+1) + (4+2+1) + (1) + (8+4+1) + \cdots}{n}$$

$$= \frac{(1+2+4+8+\cdots) + (1+2+4+\cdots) + (1+1+1+\cdots)}{n}$$

$$= \frac{\log_2 n + \log_2 n + n}{n}$$

$$= \frac{2\log_2 n + n}{n} = \frac{n}{n} = O(n1)$$

P.T.O →

URBAN EDGE

Randomized Algorithm

- An algo. that employs a degree of randomness as parts of its logic.

- It uses some random no. as an auxiliary input to guide its behaviour in the hope of achieving good performance in the average case over all possible choices of random bits.

Types: i) Las Vegas
ii) Monte Carlo

Step1: Given an array with 'n' elements (n is even)

Eg.

Array = [0, 1, 0, 1, 1, 1, 0, 1, ..., n]

Step2: Half of the array contains 0, other half contains 1.

Step3: Find an index that contains 1.

Step4: CASE 1: Repeat:

```
K = RandInt(n)
if A[K] = 1
return n;
```

Las Vegas

CASE 2: Repeat 300 times:

```
K = RandInt(n)
if A[K] = 1
return K
return "failed
```

Monte Carlo

Ans5: CASE 1:

In this case, we will deal with the Result & not with execution time.

Eg.

It will traverse all element until we get 1.

Probability [failure] = 0

Worst Case running time = Not bounded

Expected Running Time = O(1) [2 iteration]

This is LAS VEGAS algo..

Eg. Randomized Quick Sort.

CASE 2:

Probability [failure] = $\frac{1}{2^{300}}$

Worst Case Running time = O(1)

Expected running time.

This is called MONTE CARLO Algo.

P.T.O→

22/10/24

URBAN EDGE

Q) Why Las Vegas is known as probably fast but deterministic accurate.

→ It always produce the correct answer, its running time is a random variable whose expectations is bounded by a polynomial that's why it is prob. fast but deterministic accurate.

Min-Cut Graph Algorithm

It is used to select the edge with minimum weight and according to this minimum edge weight is considered in a network graph.

For its better working, following points must be taken into consideration-

A cut of connected graph is obtained by dividing vertex set (v) of graph G into two sets v_1 & v_2

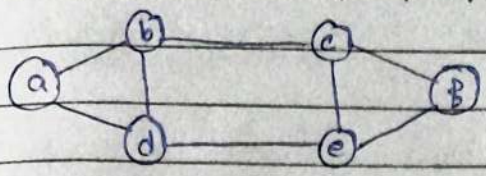
1. There are no common vertices in v_1 & v_2 i.e. two sets are disjoint.
2. $v_1 \cup v_2 = V$

Algorithm for Min-Cut Graph

Random Min_Cut (G)

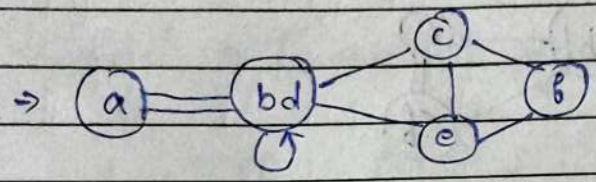
1. Repeat $s \to 2$ (step) to 4 until only 2 vertices are left.
2. Pick an edge e(v,u) at random.
3. Merge u & v
4. Remove self loop from E
5. Return E

Q.

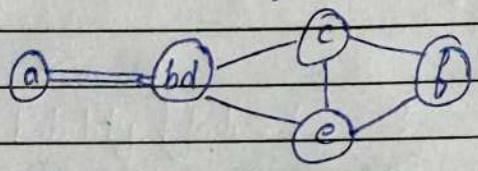


1. G(V, E)
2. Edges = {a, b, c, d, e, f}

3. V == 2
   6 > 2
   Merge random (b, d)

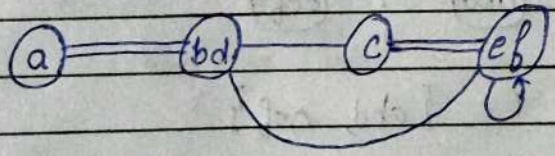


Remove self loop

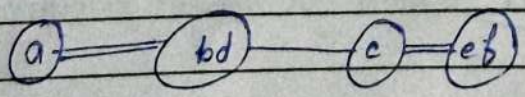


E = {a, bd, c, e, f}

4. V == 2
   5 > 2



Remove self loop



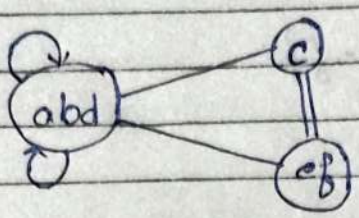
E = {a, bd, c, ef}

URBAN EDGE

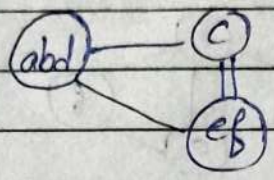
5. $v == 2$

$4 > 2$

merge (a, bd)



Remove self loop

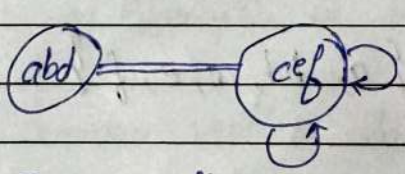


$E \Rightarrow \{abd, c, ef\}$

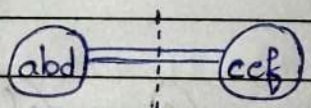
6. $v == 2$

$3 > 2$

merge (c, ef)



Remove self loop



$E = \{abd, cef\}$

$v == 2$

$2 == 2$

Complexity classes

Types of algorithm based on time complexity:

1. Polynomial Time Algo. (TC is very less)

   Ex.
   - Linear Search ($O(n)$)
   - Binary Search ($O(\log n)$)
   - Merge Sort : $O(n \log n)$

2. Non-Polynomial or Exponential Algo. (TC is high)

   Ex.
   - TSP :- $O(n^2 \cdot 2^n)$
   - Knapsack :- $O(2^{n/2})$

Based on Results:-

1. Decision Problem :- Result is Yes or No

   Ex.
   - Linear Search

2. Optimization Problem :- Result is a no. representing an objective value. Ex.
   - Min/Max Value

P-class Problem:-

It is a set of problem that can be solved in less time (Polynomial time).

Page Missing

URBAN EDGE

Tractable Problem:-

Ex. Linear, Binary search etc.

SCHOLAR
DATE: __/__/____
PAGE: _____

NP class:-

It is a set of problems that can be solved (non deterministic) in exponential (non-deterministic polynomial type)

But these kind of problem can be verified in polynomial type
ex: TSP

Its time complexity is very high but verification time is very less.

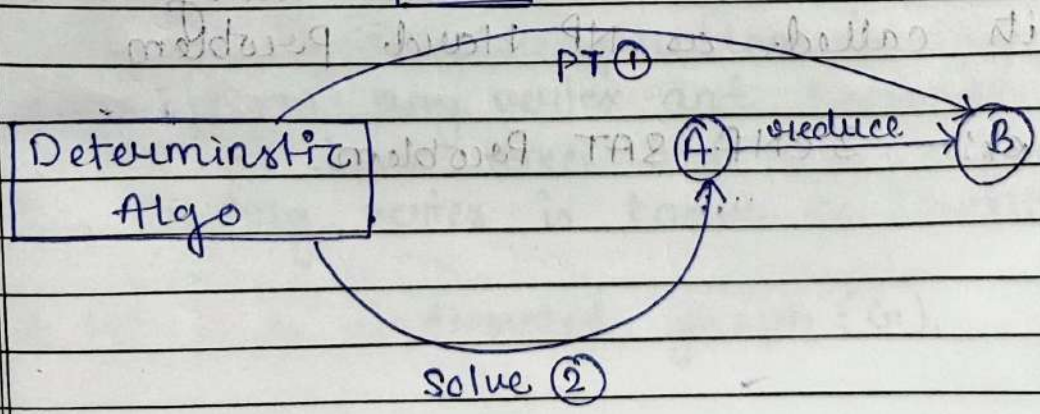
Not tractable problem.

Reduction / Reducibility Problem

let A and B are two problems in NP if prob. A is reduced to prob B, iff there is a way to solve A by deterministic algo that solve B in polynomial time.

Then we denote A is a subset of B.

$A \propto B$



SCHOLAR
DATE : ___/___/____
PAGE : _____

Properties :—

If A is reduced reduceable to B and B is in polynomial time then A also in polynomial time.

examples:—

Suppose we have two decision prob. P_1 and P_2

Problem : P_1 | Problem : P_2
Input : I_1 | Input : I_2
Algo A = ? | Algo B = (exist)

If we can solve P_1 prob. by using algo of prob P_2 that is (B) then there is no need to write an algo of (A). P_1 is a subset of P_2

$\boxed{P_1 \subseteq P_2}$

NP Hard Problem:—

Every problem in (NP) class can be reduced into other sets using PT then its called as NP Hard problem

ex:— 3 CNF (A) SAT Problem

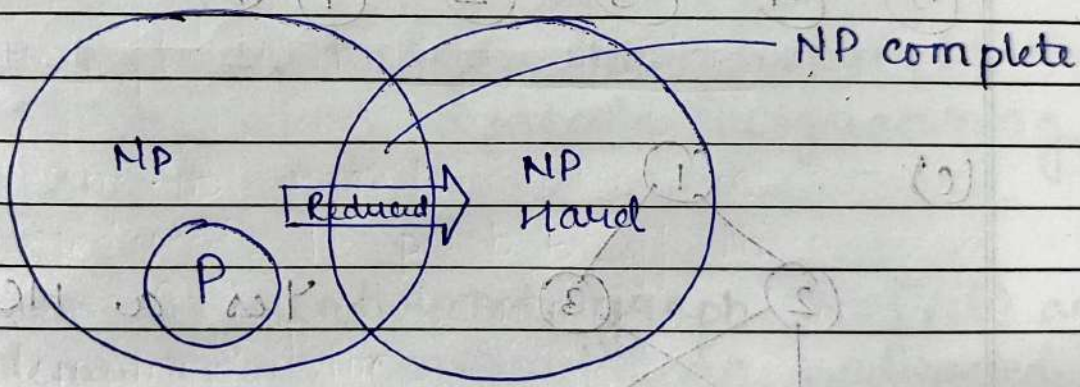
DATE: __/__/____
PAGE: _____

NP Complete Problem:—

The group of problems which are both in NP and NP Hard problem are known as NP complete problem.

All NP complete prob. are NP Hard but vice versa is not true.

ex:— Vertex cover problem
Clique problem
TSP problem
Hamilton Cycle Problem



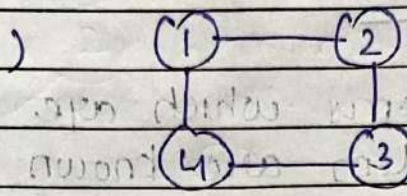
Hamilton Cycle Problem (HC)

It is a NP complete problem in this we will start from any vertex and traverse all the vertex at least once and then return to the starting vertex is known as Hamilton cycle.

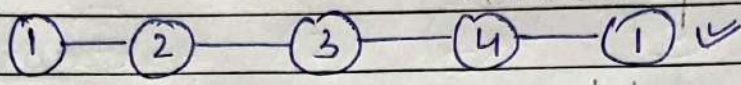
A HC is a undirected graph (G).

example :-

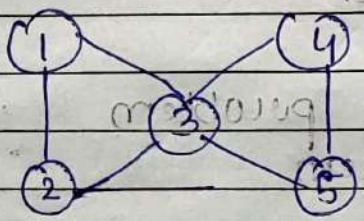
(a)



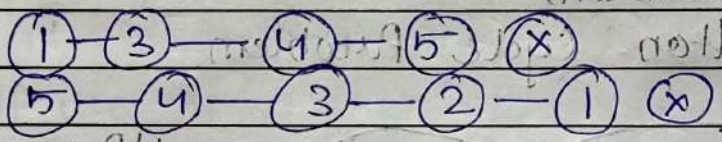
Yes a HC



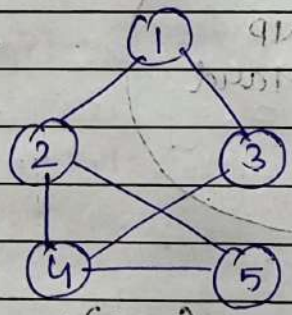
(b)



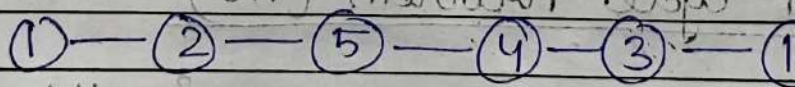
Not a HC

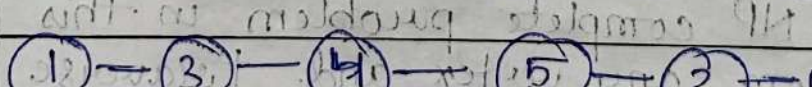


(c)



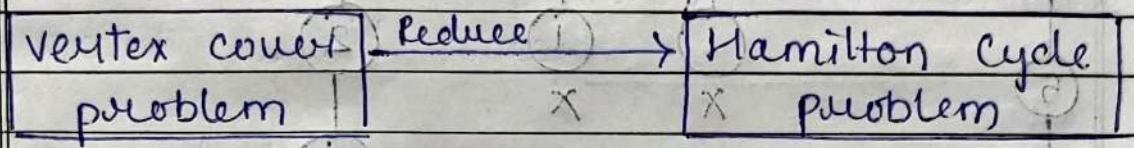
Yes a HC





Que:- Hamilton Cycle belongs to NP.complete

The verification algo checks that this sequence contains each vertex in V exactly once and that with the first vertex repeated at the end, it forms the cycle in G_1.

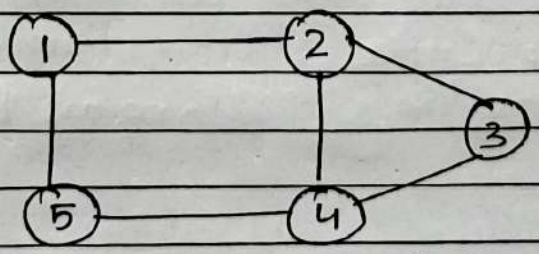


Vertex cover $\leq_p$ Hamiton Cycle.

Note:- $G_1 = (V, E)$ is a simple cycle that contains each vertex in V.
It can be used to solve the problem in time $O(2^n n^2)$ by using dynamic Programming (dp).

Note:-
Given an undirected graph $G_1 = (V, E)$ and an integer K, we constructed an undirected graph $G_2 = (V', G')$ has a hamilton cycle if and only if G_1 has a vertex cover of size K.

Que1:-



Does this graph contain HC.

29/10/24

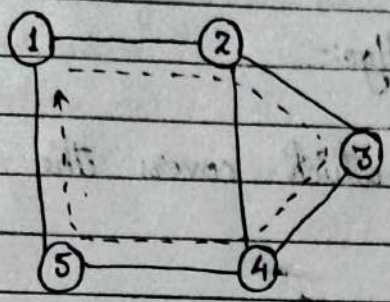
URBAN EDGE

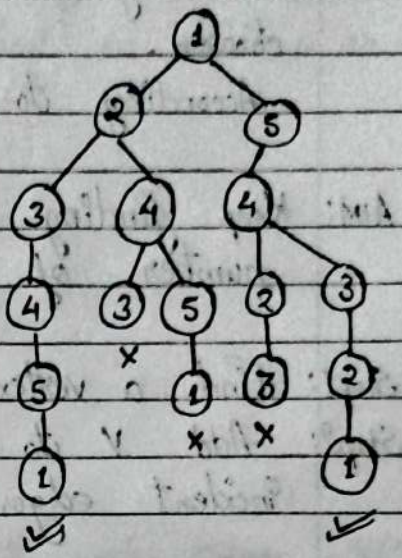
Hamilton Cycle ⟶ start & end on same node.

↳ (NP Complete)

NP ∩ NP Hard

Belongs to Both





(1) → 2 → 3 → 4 → 5 → (1)

(1) → 5 → 4 → 3 → 2 → (1)

VCP Algo → flow.

URBAN EDGE

Vertex Cover Problem

A vertex cover sometimes, node cover of a graph is a subset of vertices, which 'cover every edges.

An edge is covered if one of its end point is choosen.

According to Greedy Algo:-

AIM: Keep finding a vertex which covers the maximum number of edges.

Step1: Find a vertex V with maximum degree.

Step2: Add V to solution & remove V & all its incident edges from the graph.

Step3: Repeat until all the edges are covered.

Acc. to complexity classes :-

- It is a NP complete problem.
- A two approximation PT algo. is as following :-

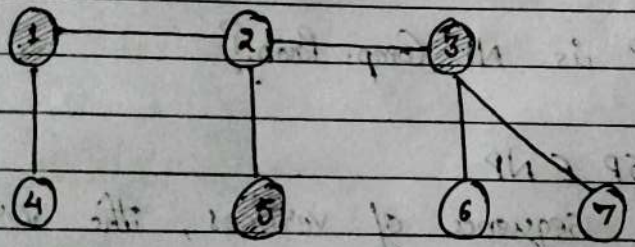
```
Approx - Vertex - Cover (G)
{
C = ϕ
E' = G.E
while (E' ≠ ϕ)
[
Randomly choose a edge (u, v) in E, put u and v into c;
Remove all the edges that covered by u or v from E
}
return c;
}
```

URBAN EDGE

2) Approx - Vertex - Cover

```
{  C ← ϕ
   E' ← E
   while E' ≠ ϕ ; do
   let (u,v) be an arbitrary edge of E'
   C ← C ∪ {(u,v)}
   remove from E' all edges incident on either u or v
   end while
   return C;
}
```

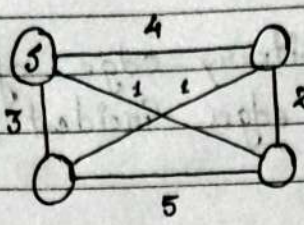
Q.) Given an undirected graph, find the vertex cover with minimum size.



Size of cover = 3

Travelling Salesman Problem

Does a graph G has tour of cost atmost K.



- Travel to all cities & come back to starting point with minimum cost.
- The order doesn't matter.
- Visit city only once.
- It is a NP complete Problem.

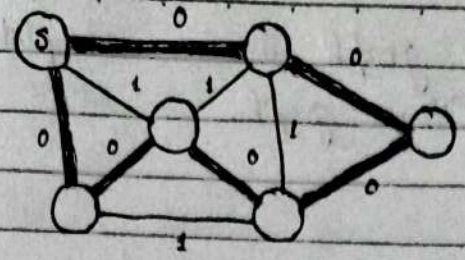
Q:- Why TSP is NP Comp. Prob.?

→ TSP ∈ NP

Given as sequence of vertices, the verification algo. checks that, this sequence contain each vertex exactly once. Sum up the edges, cost and checks whether the sum is at the most K.

Hamilton Cycle → TSP

Algo. VCP



E' ⇒ ∈
E ⇒ –

Hamilton Cycle ≤P TSP

→ Let $G = (V, E)$ is an instance of Hamilton cycle.
An ~~ais~~ instance of TSP is constructed as,
Form a complete graph $G' = (V, E')$ where
E

$$E' = \{(i,j) \in V \text{ and } i \neq j\}$$

$$c(i,j) = \begin{cases} 0, & \text{if } (i,j) \in E \rightarrow \text{hamilton cycle.} \\ 1, & \text{if } (i,j) \notin E \end{cases}$$

The graph G has a hamilton cycle if and only if graph G' has a tour of cost at most 0 (value of K).

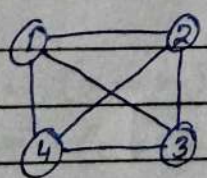
9/10/24

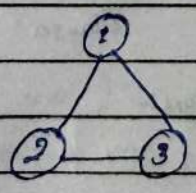
Clique Decision Problem

Complete graph :-
Let $|V| = n$

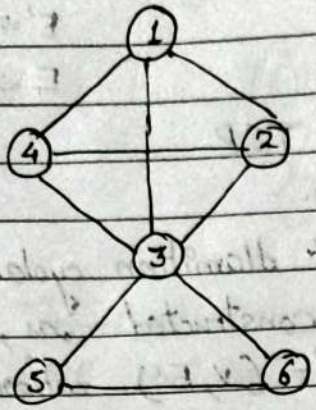
$$|E| = \frac{n(n-1)}{2}$$

Ex:-





⇒ A subgraph of a graph is a complete graph is known a Clique graph.

Q) 

Find the clique of this graph.

K=4 [1, 2, 3, 4]
K=3 [3, 5, 6]
K=2 [5, 6]

It is a NP Hard problem.

Q) Prove that CDP is a NP Hard Problem.

⇒ We have to proof that SAT is reduced to CDP then it is called as CDP Problem.

SAT ∝ CDP

Let take a 3 variable i.e x_1, x_2, x_3.

Properties of SAT:-

$$F = (x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2}) \wedge (x_1 \vee x_3)$$

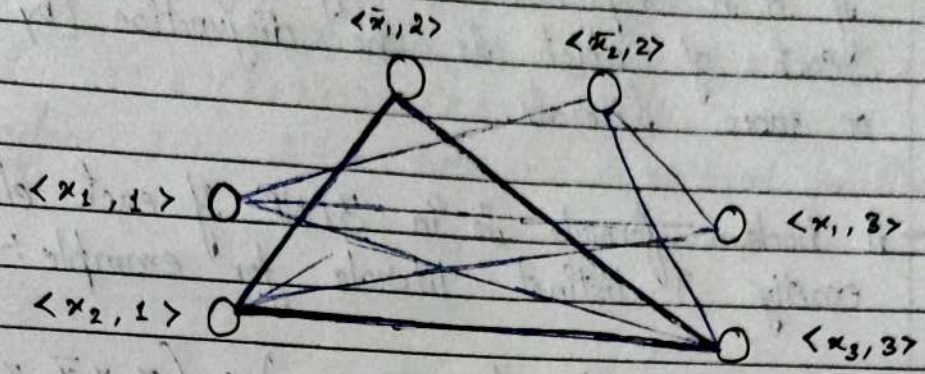
formula — c_1, c_2, c_3 — Clauses

Formula of clauses:-

$$F = \bigwedge_{i=1}^{K} C_i$$

By using these clauses I have to prepare a graph by using two rules:-

1) Same Clauses Vertices are not connected with each other.
2) Negation of same vertices are also not connected with each other.



Clique of a graph:-

$K = 3 \quad [\bar{x}_1, x_2, x_3]$

Vertices of a graph

$$G(v) = \{ \langle a_i, i \rangle \mid a \in c_i \}$$

→ a is a variables that denotes vertices
→ i is no. of clauses.

$\bar{x}_1$	x_2	x_3
0	1	1

Now,

$$F = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee x_3)$$
$$= (0 \vee 1) \wedge (1 \vee 0) \wedge (0 \vee 1)$$
$$= 1 \wedge 1 \wedge 1$$
$$= 1 \Rightarrow \text{True.}$$

Therefore CDP is NP Hard Problem.

3-CNF SAT Problem

- A boolean formula is in conjuctive normal form if it is expressed as conjunction (by AND) of clauses, each of which is the disjunction (by OR) of one or more literals.

- A boolean formula is in 3CNF if each clause has exactly 3 distinct literals for example:-

$(x_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$ is in 3 CNF SAT.

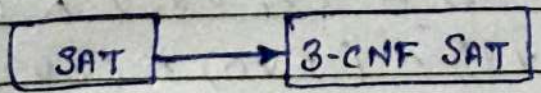
- In 3 CNF SAT, we are asked whether a given boolean formula, in 3CNF SAT is satisfiable.

- It is a ~~entry~~ NP complete problem.

Prove that 3 CNF is NP Complete:

3CNF SAT $\in$ NP

→ It can be verified in polynomial time by using simply replacing each variable in the formula with its corresponding value and then evaluates the expression.



(TSP Approximation)

URBAN EDGE

SAT $\leq_P$ 3CNF SAT

$F = X + YZ$

$= (X+Y)\ (X+Z)$

$= (X + Y + ZZ')\ (X + YY' + Z)$

$= (X+Y+Z)\ (X+Y+Z')\ (X+Y+Z)\ (X+Y'+Z)$

$= (X+Y+Z)\ (X+Y+Z')\ (X+Y'+Z)$

Therefore 3CNF is a both NP & NPhard problem.