Exam Prep

ADA

* Asymptotic Notation

Big oh (O) notation : Upper bound

$f(x) = O(g(x))$ iff. two positive constants c & x_0.

s.t. $x >= x_0$, $0 <= f(x) <= c * g(x)$

Properties:

Transitivity : $f(x) = O(g(x))$ & $g(x) = O(h(x))$

$\Rightarrow f(x) = O(h(x))$

Reflexivity : $f(x) = O(f(x))$

$\therefore O(1)$ is used to denote constants.

Eg: $f(n) = 3n^2+4n+7$
$g(n) = n^2$, prove that $f(n) = O(g(n))$

Proof: let us choose c & n_0 values as 14 & 1 resp. then we can have,

$f(x) <= c*g(n)$, $n >= n_0$ as,
$3n^2+4n+7 <= 14 \times n^2$. for all $n >= 1$
The above inequality is trivally true.
Hence, $f(n) = O(g(n))$

Big Omega (Ω): lower bound,
$f(x) = \Omega(g(x))$ iff 2 positive constants c & x_0 s.t.
$x >= x_0$, $0 <= c*g(x) <= f(x)$

Properties:
Transitivity: $f(x) = \Theta(g(x))$ & $g(x) = \Omega(h(x))$
$\Rightarrow f(x) = \Omega(h(x))$

Reflexivity: $f(x) = \Omega(f(x))$

Eg: $f(n) = 3n^2+4n+7$
$g(n) = n^2$, p.t. $f(n) = \Omega(g(n))$

Proof: let us choose c & n_0 values as 1 & 1 resp. then we can've,
$c*g(n) <= f(n)$, $n >= n_0$ as,
$1 \times n^2 <= 3n^2+4n+7$, for all $n >= 1$
The above inequality is trivally true.
Hence, $f(n) = \Omega(g(n))$

Big Theta (Θ) notation: Tight bound
$f(x) = \Theta(g(x))$ iff. 3 positive constants c_1 c_2 & x_0,
s.t for all $x >= x_0$, $c_1 * g(x) <= f(x) <= c_2 * g(x)$

Properties:
Transitivity: $f(x) = \Theta(g(x))$ & $g(x) = \Theta(h(x)) \Rightarrow f(x) = \Theta(h(x))$
Reflexivity: $f(x) = \Theta(f(x))$
Symmetry: $f(x) = \Theta(g(x))$ iff $g(x) = \Theta(f(x))$

Eg: $f(n) = 3n^2+4n+7$

$g(n) = n^2$

To prove: $f(n) = \Theta(g(n))$

Proof:

let us choose c_1, c_2 & n_0 valued as 14, 1 & 1 resp. Then we can

$f(n) <= c_1 * g(n)$, $n >= n_0$ as $3n^2+4n+7 <= 14 * n^2$ ✓

$f(n) >= c_2 * g(n)$, $n >= n_0$ as $3n^2+4n+7 >= 1 * n^2$

So,

$c_2 \times g(n) <= f(n) <= c_1 * g(n)$, $n >= n_0$

$1 \times n^2 <= f(n) <= 3n^2+4n+7 <= 14 * n^2$

The above inequality is trivally true

Hence, $f(n) = \Theta(g(n))$

Chapter-3 : Divide & Conquer :

Bubble Sort :

```
BubbleSort(A,n)
{   for(i=0; i < n-1; i++)
    {   for (j=0; j < n-1-i; j++)
        {   if ( A[j] > A[j+1])
            {
                temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
            }
        }
    }
}
```

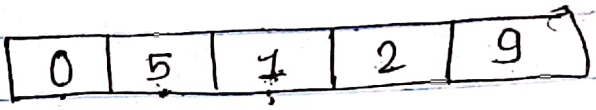
10	20	9	8	3	1
10	9	20	8	3	1
10	9	8	20	3	1
10	9	8	3	20	1
10	9	8	3	1	20
9	10	8	3	1	20

Time Complexity: Inner loop executes (n-1) times when i=0, (n-2) times i=1 & so on.

Time complexity = $(n-1)+(n-2)+(n-3)+\dots+2+1$

$= O(n^2)$

Space complexity $= O(1)$



Selection Sort:

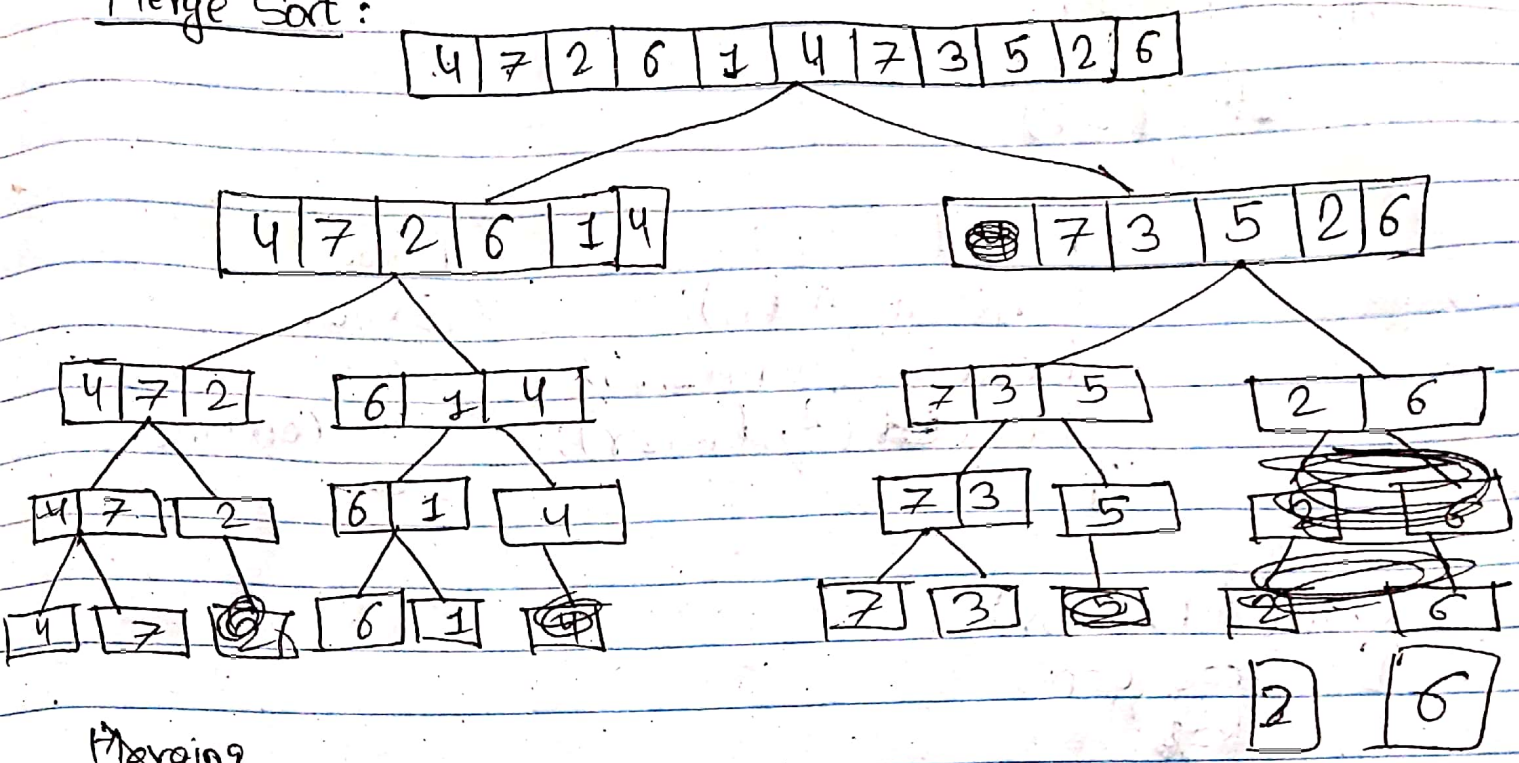
```
SelectionSort(A){
    for(i=0; i<n; i++)
    {   least = A[i];
        p = i;
        for(j=i+1; j<n; j++)
        {
            if(A[j] < least)
            {   least = A[j];
                p = j;
            }
        }
        swap(A[i], A[p]);
    }
```

Time complexity $= O(n^2)$

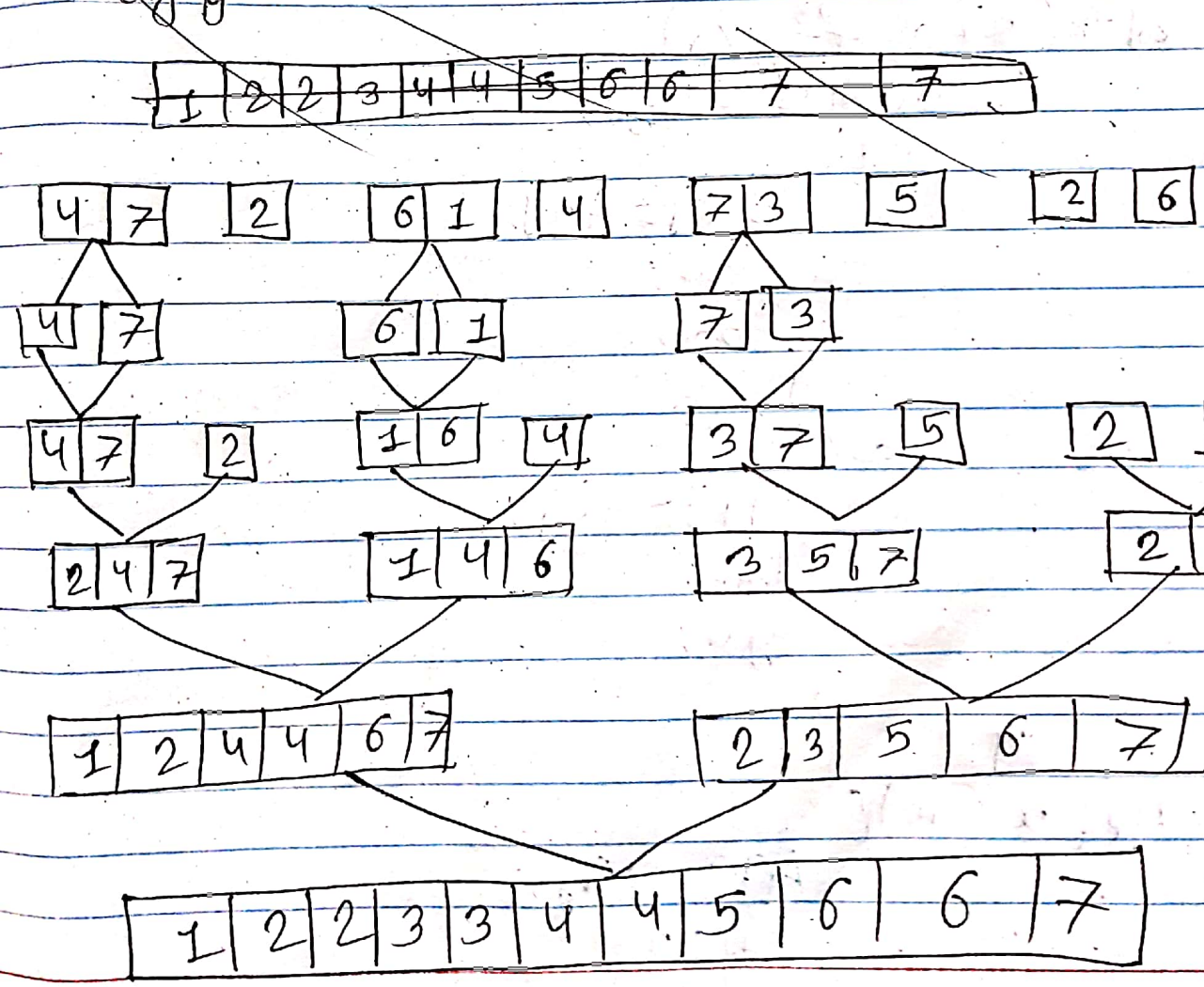
Space " $= O(1)$

```
InsertionSort(A){
    int n = arr.length;
    for(i=1; i<n; i++)
        { int j;
          int key = A[i];
          for(j=i-1; j>=0 && A[j]>key; j--){
                A[j+1] = A[j];
          }
          A[j+1] = key;
    }
}
```

Merge Sort:



Merging



```
public class MergeSort {
    void merge(int arr[], int l, int m, int r)
    {
        int n1 = m - l + 1;
        int n2 = r - m;

        int L[] = new int[n1];
        int R[] = new int[n2];

        for (int i=0; i < n1; ++i)
            L[i] = arr[l + i];
        for (int j=0; j < n2; ++j)
            R[j] = arr[m + 1 + j];

        int i=0, j=0;

        int k=l;
```

```
while (i < n1 && j < n2){
        if( L[i] <= R[j])
            { arr[k] = L[i];
              i++;
            }
        else{
              arr[k] = R[j];
              j++;
            }
        k++;
    }
while (i < n1){
        arr[k] = L[i];
        i++;
        k++;
    }
while ( j < n2){
        arr[k] = R[j];
        j++;
        k++;
    }
}

void sort(int arr[] , int l, int r){
        if(l < r){
            int m = l + (r-l)-/2;
            sort(arr, l, m);
            sort(arr, m+1 , r);
```

```
        merge(arr, l, m, r);
    }
}
static void printArray(int arr[])
{
    int n = arr.length;
    for(int i=0; i<n; ++i)
        System.out.println(arr[i] + " ");
    System.out.println();
}
public static void main(String args[])
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    System.out.println("Given Array");
    printArray(arr);
    Mergesort a = new Mergesort();
    a.sort(arr, 0, arr.length-1);
    System.out.println("\n Sorted Array");
    printArray(arr);
```

Output

Given Array

12 11 13 5 6 7

Sorted Array

5 6 7 11 12 13

Quick Sort

```
public class QuickSort {
    static void swap(int[] arr, int i, int j)
    {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    static int partition(int[] arr, int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1);

        for(int j=low; j<=high-1; j++)
        {
            if (arr[j] < pivot)
            {   i++;
                swap(arr, i, j);
            }
        }
        swap(arr, i+1, high);
        return (i+1);
    }

    static void quickSort(int[] arr, int low, int high)
    {
        if( low < high)
        {
```

```
        int pi = partition (arr, low, high);

        quickSort(arr, low, pi-1);
        quickSort (arr, ~~low~~ pi+1, high);
    }
}
static void printArray (int[] arr, int size)
    {
        for(int i=0; i<size; i++)
            System.out.print~~ln~~(arr[i]+" ");

        System.out.println();
    }

public static void main(String[] args)
    {
        int[] arr = {10,7,8,9,1,5};
        int n = arr.length;

        quickSort(arr, 0, n-1);
        System.out.println("Sorted Array:");
        printArray (arr, n);
    }
}
```

Output

Sorted Array:

1 5 7 8 9 10

Algorithm

```
Algorithm MERGESORT (low, high)
// a (low : high) is a global array to be sorted.
{
        if (low < high)
        {
                mid := |(low  + high)/2|          //finds where to split the set
                MERGESORT(low,  mid)              //sort one subset
                MERGESORT(mid+1, high)  //sort the other subset
                MERGE(low,  mid,  high)           // combine the results
        }
}
```

```
Algorithm MERGE (low, mid, high)
// a (low : high) is a global array containing two sorted subsets
// in a (low : mid) and in a (mid + 1 : high).
// The objective is to merge these sorted sets into single sorted
// set residing in a (low : high). An auxiliary array B is used.
{
        h :=low; i := low; j:= mid + 1;
        while ((h ≤ mid) and (J ≤ high)) do
        {
                if (a[h] ≤ a[j]) then
                {
                        b[i] := a[h]; h := h + 1;
                }
                else
                {
                        b[i] :=a[j]; j := j + 1;
                }
                i := i + 1;
        }
        if (h > mid) then
                for k := j to high do
                {
                        b[i] := a[k]; i := i + 1;
                }
        else
                for k := h to mid do
                {
                        b[i]  := a[K];  i := i + l;
                }
        for k := low to high do
                a[k] := b[k];
}
```

ADA

Strassen's Matrix Multiplication

Usual: $n \times n$ mul

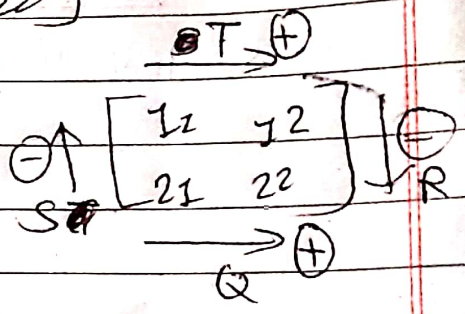
```
for i:=1 to n do
    for j:=1 to n do
        c[i,j] := 0;
        for k:=1 to n do
            c[i,j] := c[i,j] + a[i,k] * b[k,j];
```

Now

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

Formula:

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



B A A B
11 11 22 22

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

Job Sequencing Deadline

let, $n=4$

$(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$

$(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$

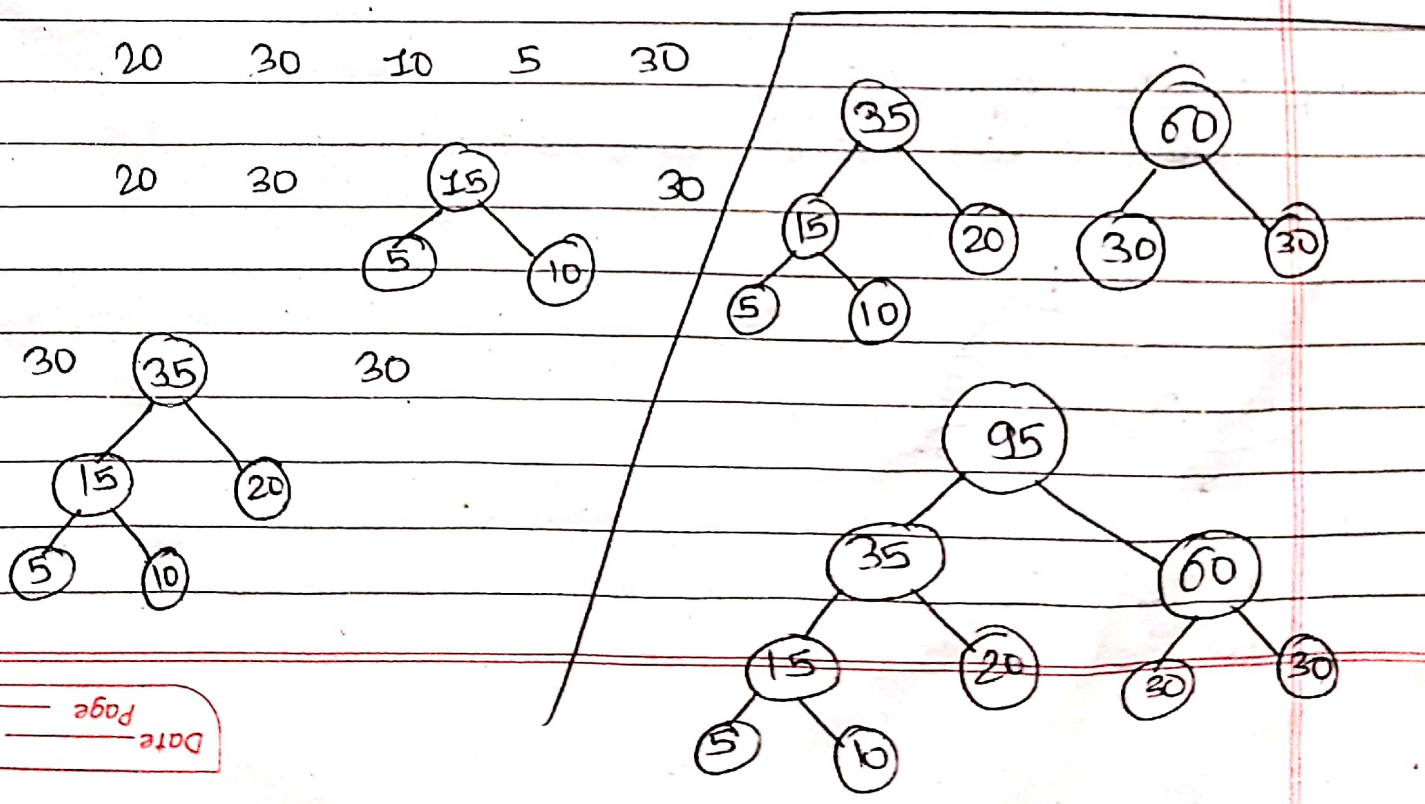
POC

P_2	P_4	P_1	P_3
1	1	2	2
10	15	100	27
~~P_3, P~~		P_4, P_1	

S.N.	Feasible soln	Processing seq	Value	Remarks
1	1, 2	2, 1	110	
2	1, 3	1, 3 or 3, 1	115	
3	1, 4	4, 1	127	(optimal)
4	2, 3	2, 3	25	
5	~~3~~ 3, 4	4, 3	42	
6	1	1	100	
7	2	2	10	
8	3	3	15	
9	4	4	27	

Optimal Merge Pattern

$(x_1, x_2, x_3, x_4, x_5) = (20, 30, 10, 5, 30)$



Date Page

Huffmann Coding

Using Traditional approach

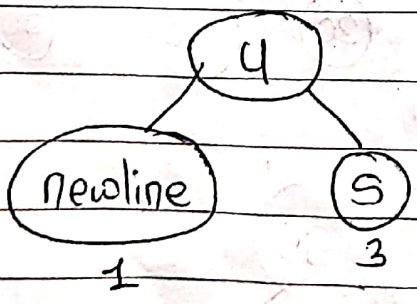
character	a	e	i	s	t	space	new line
frequency	10	15	12	3	4	13	1

Character	Code	Frequency	Total bits
a	000	10	10×3 = 30
e	001	15	15×3 = 45
i	010	12	12×3 = 36
s	011	3	3×3 = 9
t	100	4	4×3 = 12
space	101	13	13×3 = 39
newline	110	1	1×3 = 3

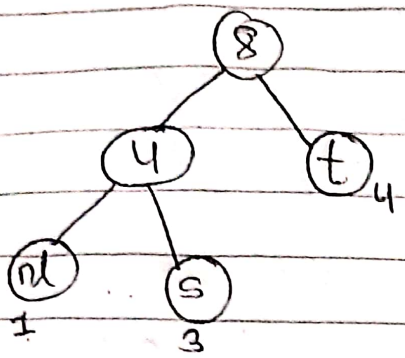
∴ Total bits = 30+45+36+9+12+39+3
= 174

Using Huffmann approach:

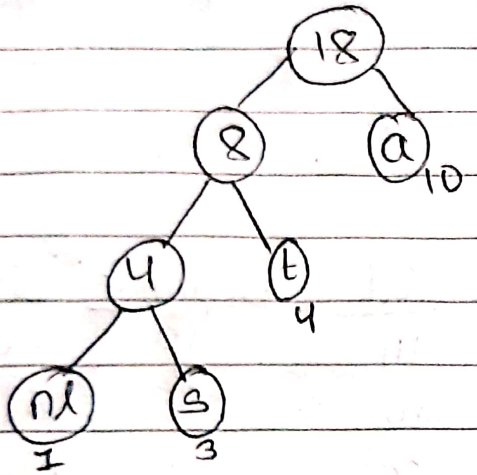
Select 2 least element with least frequency.



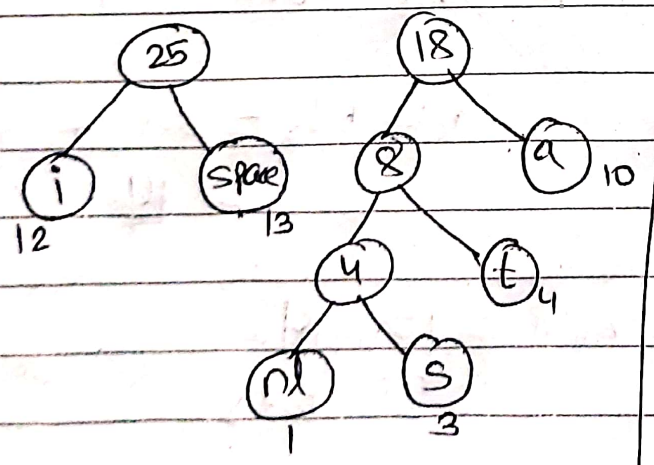
a	e	i	4	t	space
10	15	12	4	4	13



a	e	i	8	space
10	15	12	8	3

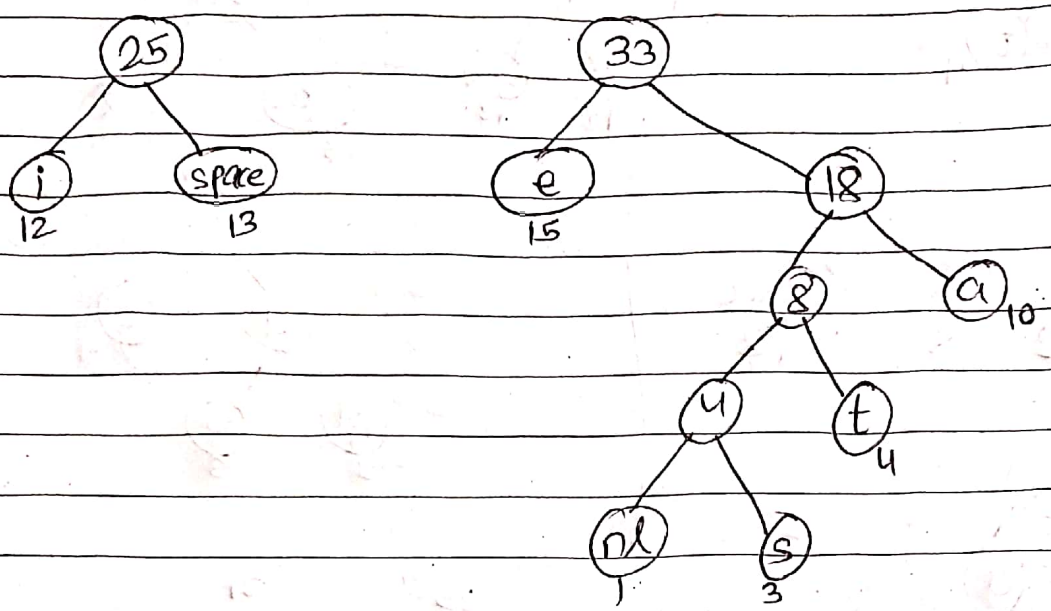


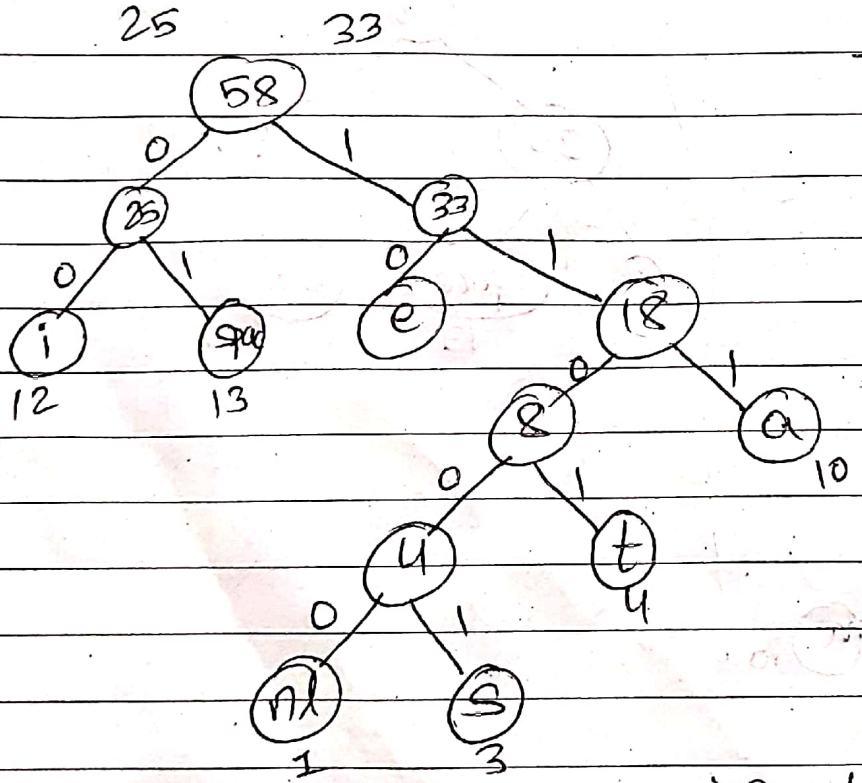
- e i 18 space
  15 12 18 13



Date
Page

e
15 18 25





chr	Code	frequency	Bits
a	111	10	10×3=30
e	10	15	15×2=30
i	00	12	12×2=24
s	11001	3	3×5=15
t	1101	4	4×4=16
sp	01	13	13×2=26
nl	11000	1	1×5=5

∴ Total bits = 30+30+24+15+16+26+5
= 146

∴ Saved = 174-146
= 28 bits

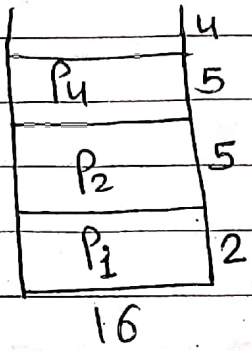
0/1 Knapsack Problem (Greedy approach)

n=4, W=16

Given,

i	P_i	W_i	P_i / W_i
1	40	2	20 ✓
2	30	5	6 ✓✓
3	50	10	5
4	10	5	2 ✓✓✓

Sol?



W_i, ((10 > (16-5-2)), so, it can't fit

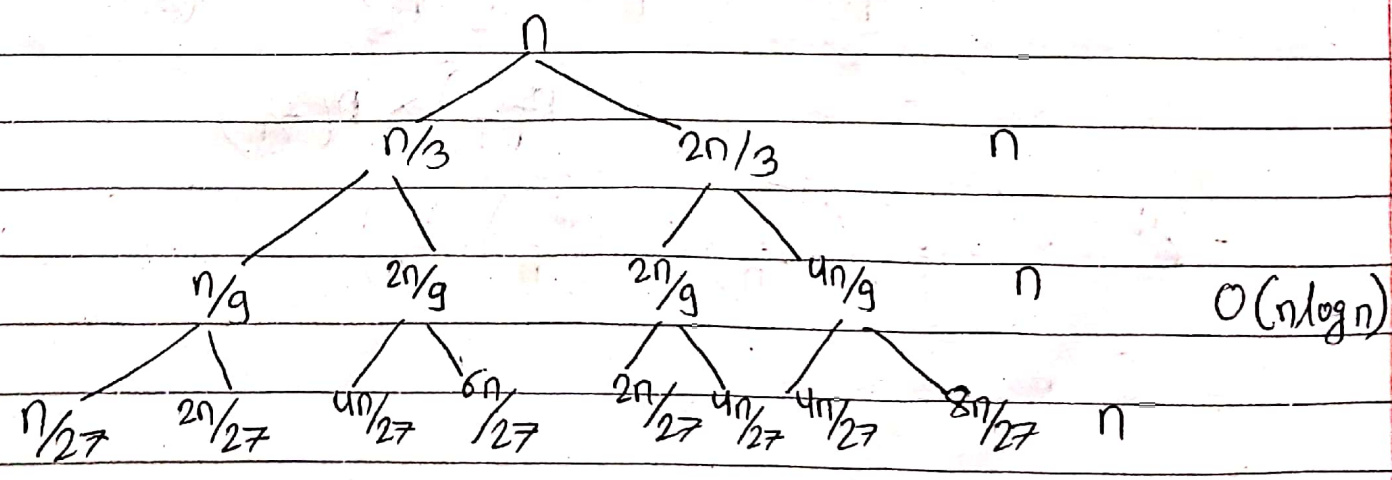
Now, move to next highest profit rate

$P = 40+30+10 = 80$

Recurrence Relation:

Recursive Method

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



Substitution Method

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

Now,

$T(n) = n * T(n-1)$ --- (i)

$T(n-1) = (n-1) * T(n-2)$ --- (ii)

$T(n-2) = (n-2) * T(n-3)$ --- (iii)

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

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

$= n(n-1)\{(n-2) * T(n-3)\}$

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

$= n(n-1)(n-2)(n-3) \ldots\ldots T(n-(n-1))$

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

$= n(n-1)(n-2)(n-3) \ldots T(1)$

$= n(n-1)(n-2)(n-3) \ldots 3,2,1$

$= n \times n\left(1-\frac{1}{n}\right) \times n\left(1-\frac{2}{n}\right) * n\left(1-\frac{3}{n}\right) \ldots\ldots \times n\left(\frac{3}{n}\right) \times n\left(\frac{2}{n}\right) \times n\left(\frac{1}{n}\right)$

$= n^n \cdot \left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right) \ldots\ldots \left(\frac{2}{n}\right)\left(\frac{1}{n}\right)$

$= O(n^n)$

Substitution Method

Q) $T(n) = \begin{cases} 1 & , n=1 \\ 2T(n/2)+n & , \text{otherwise} \end{cases}$

Soln,

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

$T(n/2) = 2T(n/4) + n/2$ --- (ii)

$T(n/4) = 2T(n/8) + n/4$ --- (iii)

Substituting,

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

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

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

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

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

$= 8T(n/8) + n + 2n$

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

$= \cancel{2^3 T(n/8) + kn}$ $2^3 T(n/2^3) + 3n$

$\vdots$

$= 2^k T(n/2^k) + kn$

When, $n = 2^k$

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

$\log n = k \cdot 1 = k$

$\therefore 2^k T(n/2^k) + kn$

$= n \cdot T(n/n) + n \log n$

$= nT(1) + n\log n$

$= n + n\log n \quad \therefore O(n \log n)$

Substitution Method

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

Soln

$$T(n) = T(n-1) + \log n \quad \text{---(i)}$$
$$T(n-1) = T(n-2) + \log(n-1) \quad \text{---(ii)}$$
$$T(n-2) = T(n-3) + \log(n-2) \quad \text{---(iii)}$$

Substituting,

$$T(n) = T(n-1) + \log n$$
$$= T(n-2) + \log(n-1) + \log n$$
$$= T(n-3) + \log(n-2) + \log(n-1) + \log n$$
$$= T(n-4) + \log(n-3) + \log(n-2) + \log(n-1) + \log n$$
$$\vdots$$
$$= T(n-k) + \log[n-(k-1)] + \log[n-(k-2)] + \dots + \log n$$

Now,

let, $n-k=1 \Rightarrow k = 1+n \approx n$

Then,

$$T(n) = T(1) + \log[n-((n-1)-1)] + \log[n-((n-1)-2)] + \dots + \log n$$
$$= 1 + \log(n-n+1) + \log(n-n+2) + \dots + \log n$$
$$= 1 + \log(1) + \log(2) + \dots + \log n$$
$$= 1 + \log(1 \cdot 2 \cdot 3 \cdot \dots \cdot n)$$
$$= 1 + \log(n!)$$
$$= 1 + \log(n^n)$$
$$= 1 + n \cdot \log n$$
$$= n \log n$$

$\therefore O(n \log n)$

Master Method

→ $T(n) = aT(n/b) + f(n)$, where $a \geq 1$, $b > 1$

→ Soln is,

$$T(n) = n^{\log_b a}[U(n)]$$

→ $U(n)$ depends on $h(n)$

→ $h(n) = \dfrac{f(n)}{n^{\log_b a}}$

→ Relation bet^n $h(n)$ & $U(n)$ is;

if $h(n)$	$U(n)$
$n^r, r > 0$	$O(n^r)$
$n^r, r < 0$	$O(1)$
$(\log_2 n)^i, i \geq 0$	$\dfrac{(\log_2 n)^{(i+1)}}{(i+1)}$

Eg:. $T(n) = 8T(n/2) + n^2$

where, $a = 8$, $b = 2$, $f(n) = n^2$

$\therefore T(n) = n^{\log_b a}[U(n)]$

$= n^{\log_2 8}.[U(n)]$

$$h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{n^2}{n^{\log_2 8}} = \frac{n^2}{n^3} = n^{-1}$$

Since, $h(n) = n^{-1}$ where, $-1 < 0$.

$\therefore U(n) = O(1)$

$\therefore T(n) = n^{\log_2 8}.1$

$= n^3 \cdot 1 = n^3$

Eg: $T(n) = T(n/2) + c$

Here, $a = 1,\ b = 2,\ f(n) = c$

$\therefore T(n) = n^{\log_b a} \cdot U(n)$

$= n^{\log_2 1} \cdot U(n)$

$= n^{\log_2 2^0} \cdot U(n)$

$= n^{0 \cdot \log_2 2} \cdot U(n)$

$= 1 \cdot U(n)$

$\therefore h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{c}{1} = c = (\log_2 n)^0 \cdot c$

$\therefore U(n) = \frac{(\log_2 n)^{(0+1)}}{(0+1)} \cdot c = \log_2 n \cdot c$

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

$= O(\log_2 n)$

Eg: $T(n) = \begin{cases} T(\sqrt{n}) + \log n & , n \geq 2 \\ O(1) & , \text{else} \end{cases}$

Soln,

$T(n) = T(\sqrt{n}) + \log n$ $\quad \left[\because T(n) = aT(n/b) + n^k \log^p n\right]$

Let, $n = 2^m \Rightarrow \log n = m \cdot \log 2$

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

$= T(2^{m/2}) + m \cdot \log_2 2 \quad = T(2^{m/2}) + m$

Here,

$a = 1,\ b = 2,\ f(n) = m \cdot \log 2$

$T(2^m) = S(m)$

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

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

Eg: $T(n) = T(n/2) + c$

Here, $a = 1,\ b = 2,\ f(n) = c$

$\therefore T(n) = n^{\log_b a} \cdot U(n)$

$= n^{\log_2 1} \cdot U(n)$

$= n^{\log_2 2^0} \cdot U(n)$

$= n^{0 \cdot \log_2 2} \cdot U(n)$

$= 1 \cdot U(n)$

$\therefore h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{c}{1} = c = (\log_2 n)^0 \cdot c$

$\therefore U(n) = \frac{(\log_2 n)^{(0+1)}}{(0+1)} \cdot c = \log_2 n \cdot c$

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

$= O(\log_2 n)$

Eg: $T(n) = \begin{cases} T(\sqrt{n}) + \log n, & n \geq 2 \\ O(1), & \text{else} \end{cases}$

Soln,

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

$\left[\because T(n) = aT(n/b) + n^k \log^p n\right]$

let, $n = 2^m \Rightarrow \log n = m \cdot \log 2$

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

$= T(2^{m/2}) + m \cdot \log_2 2 \quad = T(2^{m/2}) + m$

Here,

$a = 1,\ b = 2,\ f(n) = m \log 2$

$T(2^m) = S(m)$

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

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

$a < b^k$

$1 < 2$,

$T(n) = n^k \log^p n$

$T(n) = n^1 \cdot 1$

or, $T(n) = n$

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

Eg: $T(n) = 28T(n/3) + cn^3$

Soln,

Here, $a = 28$, $b = 3$, $f(n) = cn^3$

$\therefore T(n) = n^{\log_b a} \cdot U(n)$

$= n^{\log_3 28} \cdot U(n)$

$\therefore h(n) = \frac{f(n)}{n^{\log_3 28}} = \frac{cn^3}{n^{\log_3 28}} = c \cdot n^{3-\log_3 28} = cn^r$

$r = 3 - \log_3 28 < 0$

$\therefore h(n) = O(n^r)$, $r < 0$, $U(n) = O(1)$

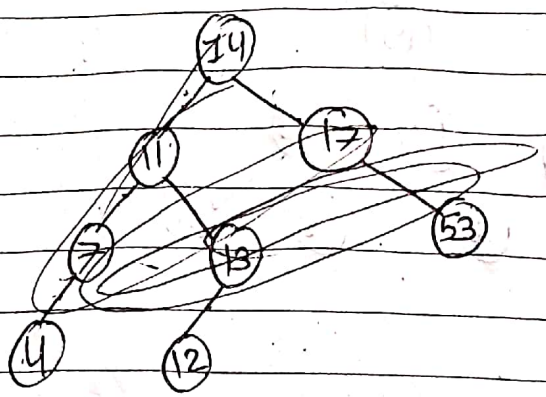
$\therefore T(n) = n^{\log_3 28} [T(1) + h(n)]$

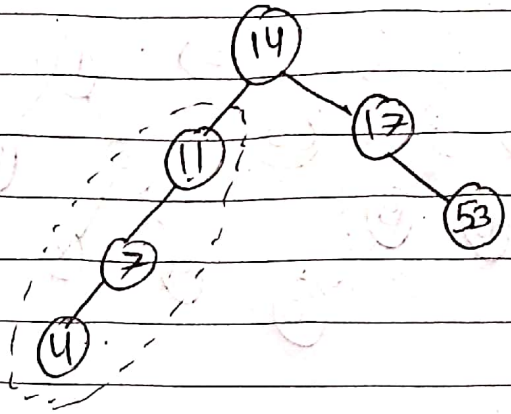
$= n^{\log_3 28} [T(1) + O(1)]$

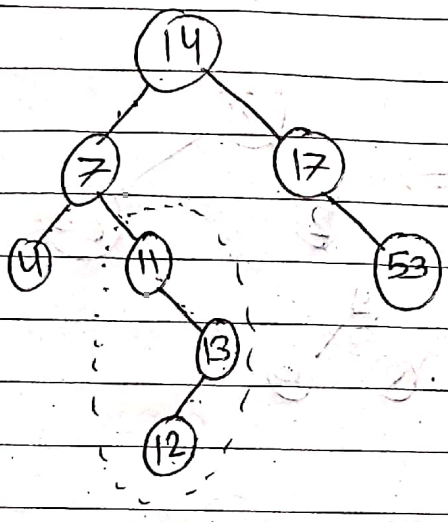
$= \Theta(n^{\log_3 28})$

AVL

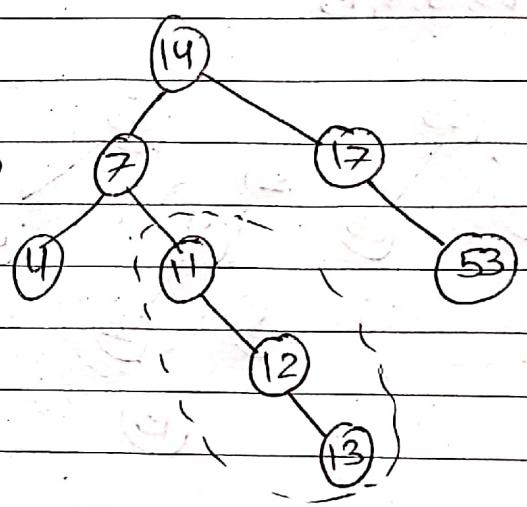
Q) 14, 17, 11, 7, 53, 4, 13, 12, 8, 60, 19, 16, 20



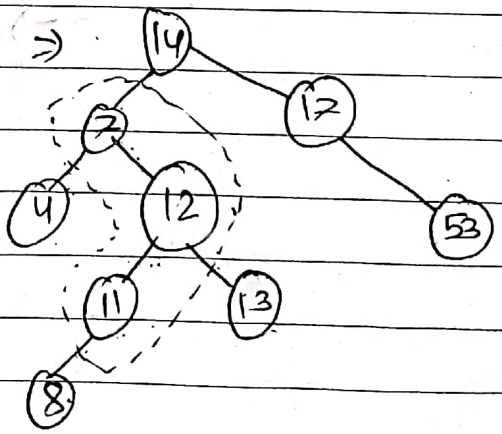




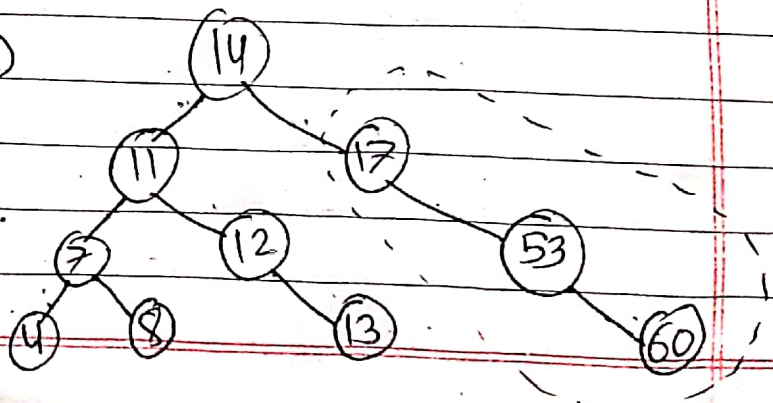
⇒



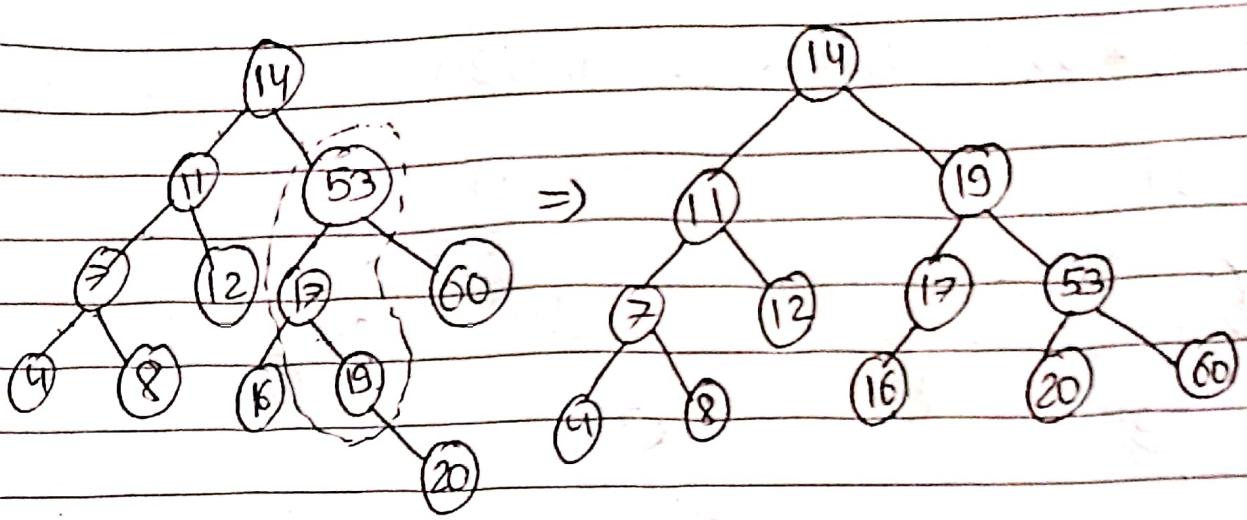
⇒



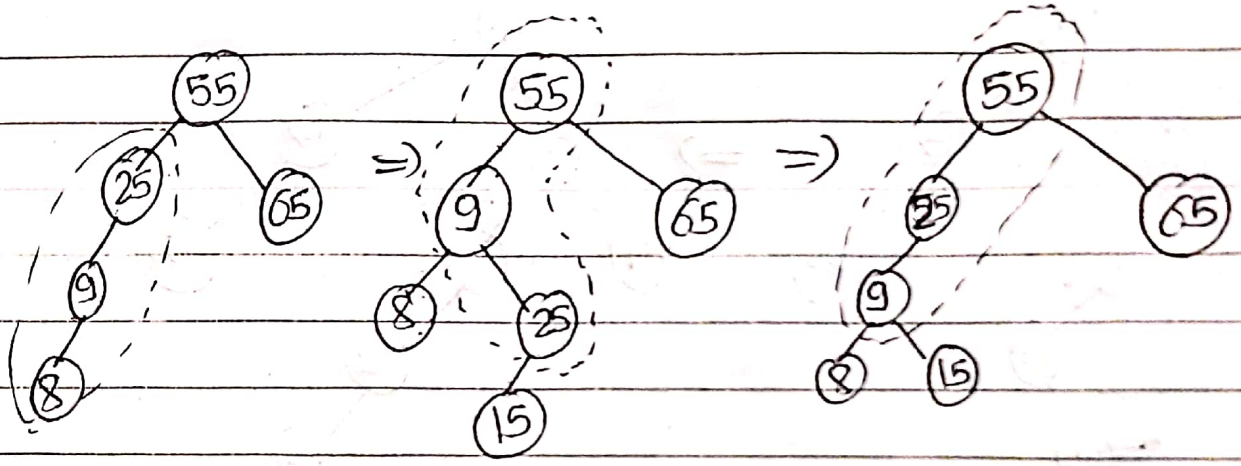
⇒



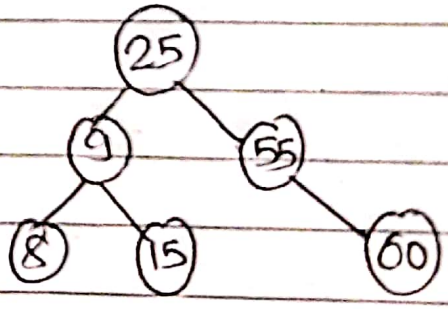
Date Page



(1) 55, 25, 65, 9, 8, 15



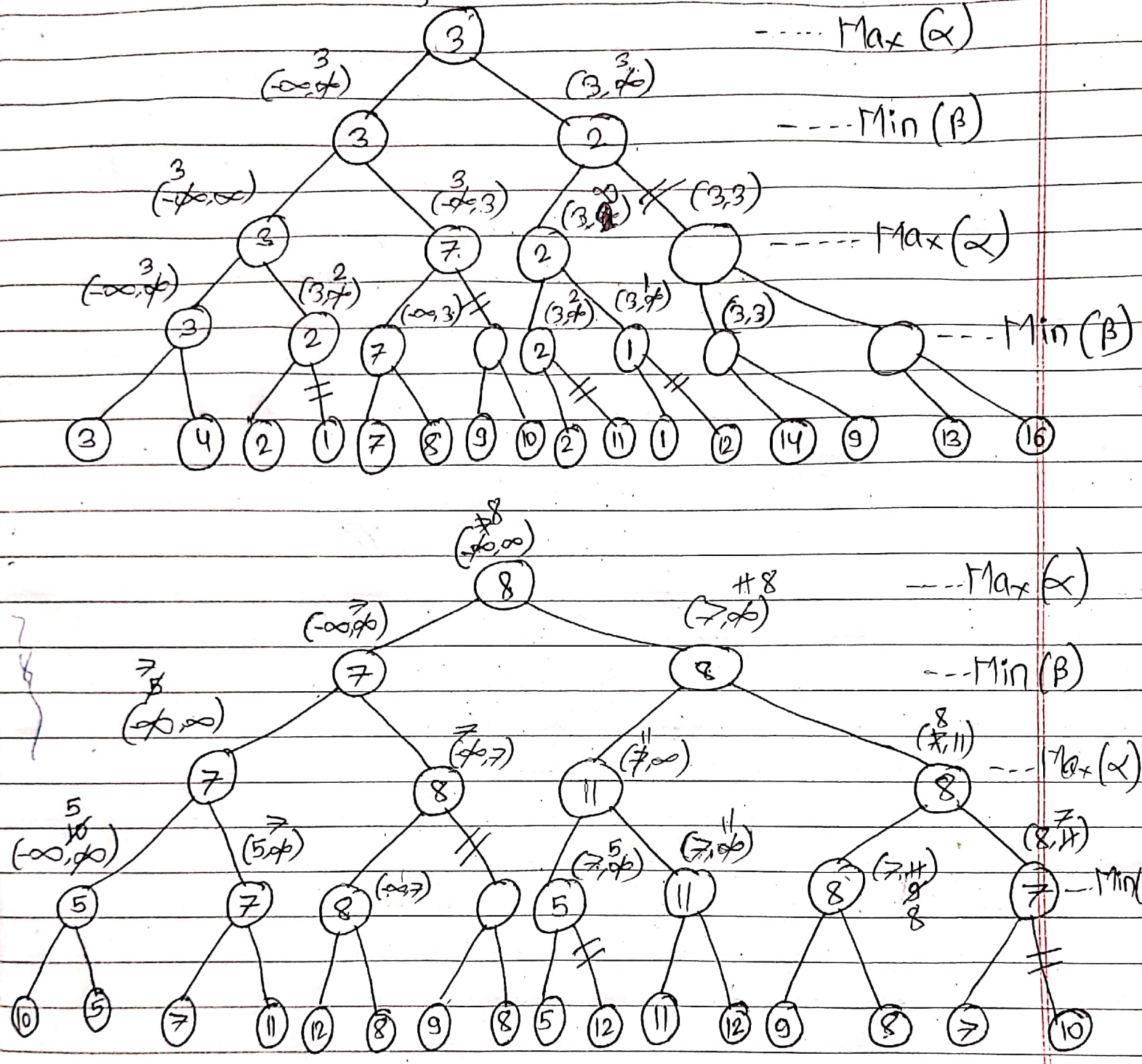
=>



MINIMAX SEARCH (Do yourself)

Alpha beta Pruning ($\alpha - \beta$) → Rules To Prune: $\alpha \geq \beta$

Max = α
Min = β



Scanned with CamScanner

※ Travelling Sales Person :

Dynamic Prog :

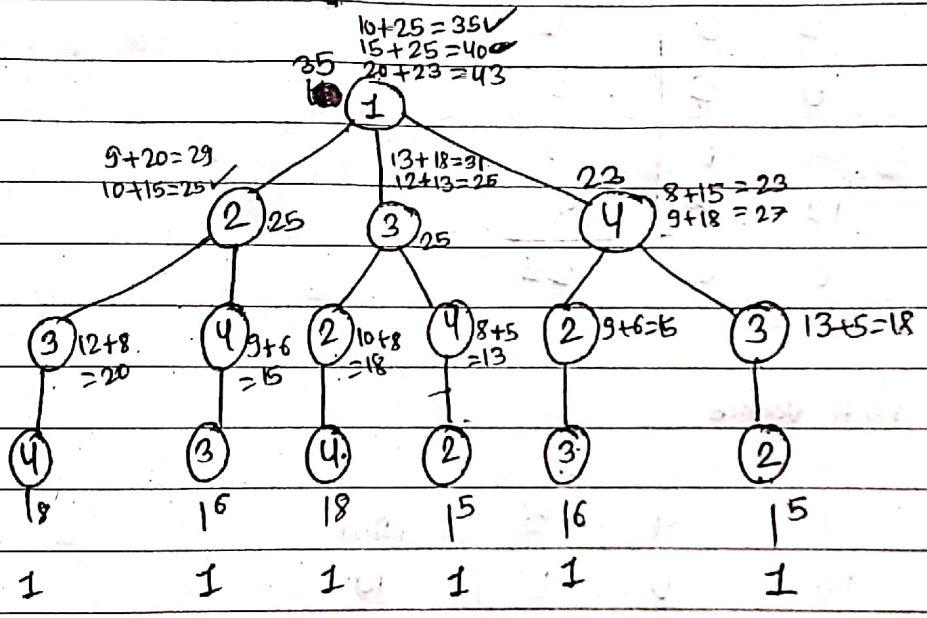
$$g(1, V-\{1\}) = \min_{2 \le k \le n} \{ c_{1k} + g(k, V-\{1,k\}) \} \quad \text{---(i)}$$

Generalizing eqn (i),

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

Q) A =

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



Now, C_{i1}

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

$g(2,\{3\}) = 9+6 = 15$
$g(2,\{4\}) = 10+8 = 18$
$g(3,\{2\}) = 13+6 = 19$
$g(3,\{4\}) = 12+8 = 20$
$g(4,\{2\}) = 8+5 = 13$
$g(4,\{3\}) = 9+6 = 15$

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

$g(1,\{2,3,4\}) =$
$\min\{ C_{12} + g(2,\{3,4\}),$ $C_{13} + g(3,\{2,4$
$C_{14} + g(4,\{2,3\}) \}$
$= \min\{10+25, 15+25, 20+23\}$
$= \min\{35, 40, 43\} = 35$

Date Page

Travelling Salesperson

Branch & Bound :

	1	2	3	4	5	min
1	∞	20	30	10	11	10
2	15	∞	16	4	2	2
3	3	5	∞	2	4	2
4	14	6	18	∞	3	3
5	16	4	7	16	∞	4

Subtract rows with resp. min value

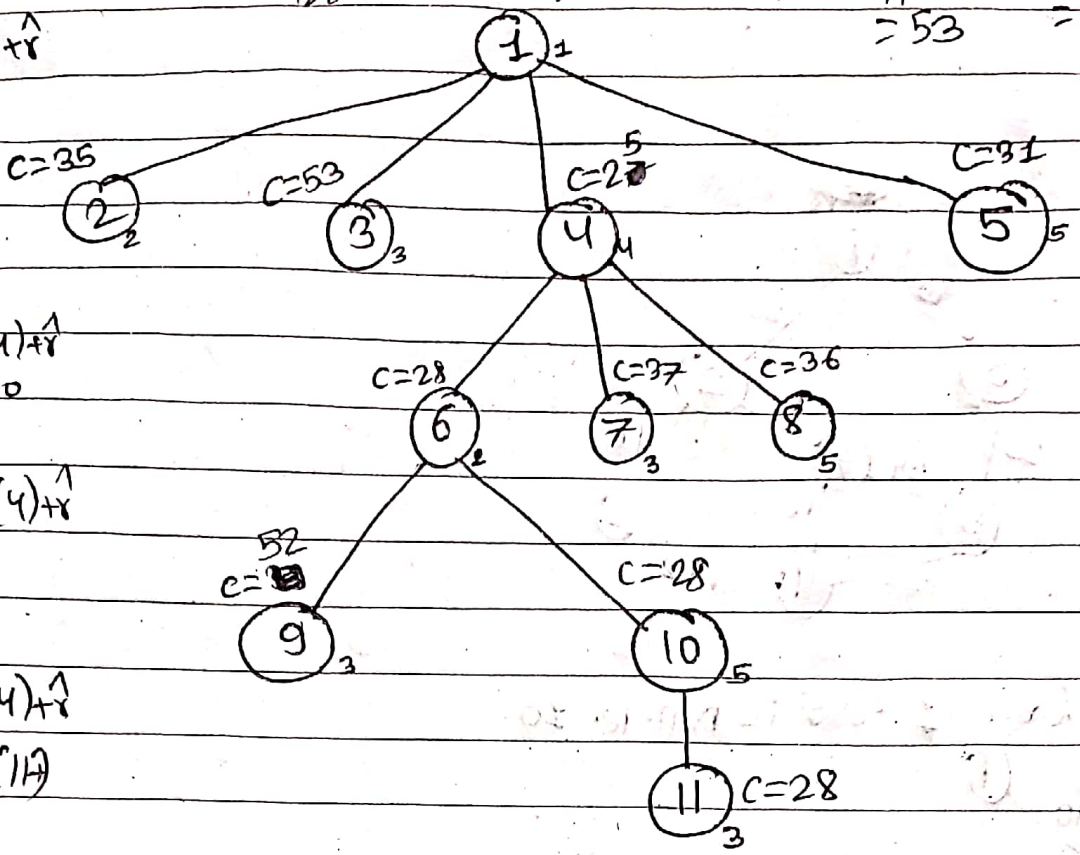
	1	2	3	4	5	min
1	∞	10	20	0	1	10
2	13	∞	14	2	0	2
3	1	3	∞	0	2	2
4	11	3	15	∞	0	3
5	12	0	3	12	∞	+4
min	1	0	3	0	0	21

Subtract col. with resp min value

	1	2	3	4	5	min
1	∞	10	17	0	1	10
2	12	∞	11	2	0	2
3	0	3	∞	0	2	2
4	10	3	12	∞	0	3
5	11	0	0	12	∞	+4
min:	1	0	3	0	0	21+4 = 25

reduced cost = 21+4
= 25

upper = ~~∞~~ 28



$C(1,2)+r+\hat{r}$

$10+25+0$

$= 35$

$C(1,3)+r+\hat{r}$

$17+25+11$

$= 53$

$C(1,4)+r+\hat{r}$

$= 0+25+0$

$= 25$

$C(1,5)+r$

$= 1+25+($

$= 31$

$C(4,2)+c(4)+\hat{r}$

$= 3+25+0$

$= 28$

$C(4,3)+c(4)+\hat{r}$

$= 12+25+0$

$= 37$

$C(4,5)+c(4)+\hat{r}$

$= 0+25+(11)$

$= 36$

$C(2,3)+c(6)+\hat{r}$

$= 11+28+13$

$= 52$

$C(5,3)+c(10)+\hat{r}$

$= 0+28+0$

$= 28$

$C(2,5)+c(6)+\hat{r}$

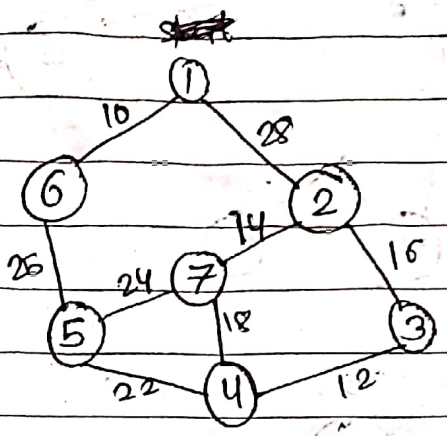
$= 0+28+11+$

✳ Graph

* MST

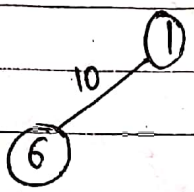
1) Prism's :

Eg:

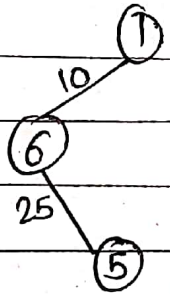


Soln

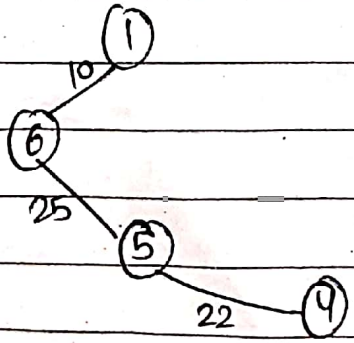
cost between 1 to 6 is min. ie. 10.



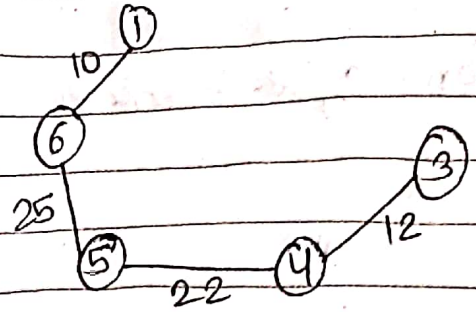
Next min is, 25 from 6 to 5 which is only adjacent of 6.



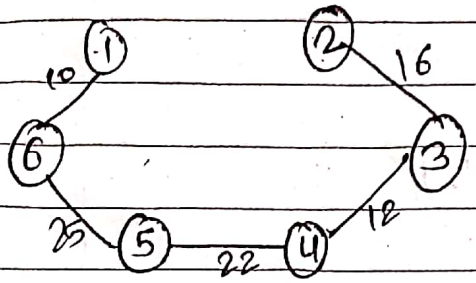
Now, adjacent vertices of 5 are 4, 7. 4 is with least cost. Now add 4.



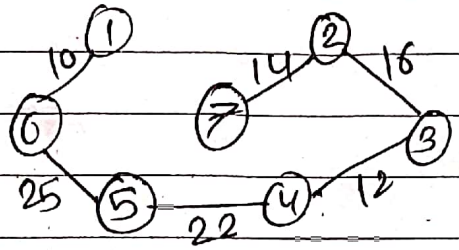
Now, adjacent vertices of 4 are 3, 7. 3 is with least cost. Add 3.



Now, adjacent of 3 are 2 with cost '16'. so add 2.



Now, adjacent of 2 is 7, 1 & 7 is with least cost. So add 7 to graph.



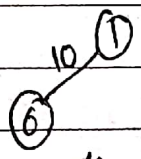
Since, all vertices are added to graph. So, it MST. with cost

$= 10 + 25 + 22 + 12 + 16 + 14$

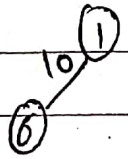
$= 99$

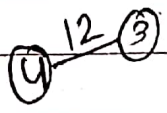
2) Kruskal's

select edges with least cost.

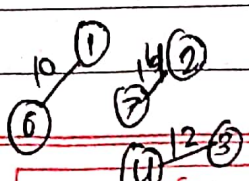


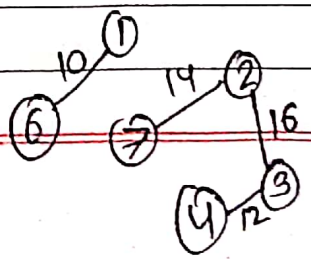
Now, add another edge with smallest cost

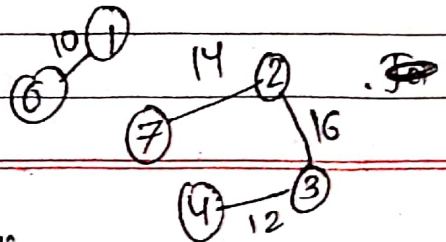




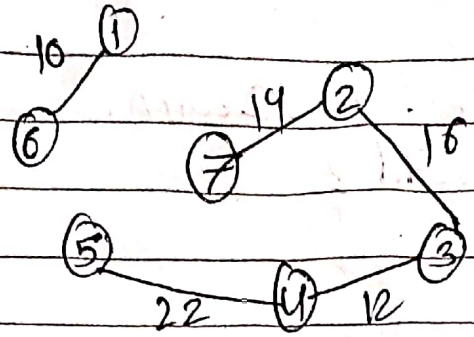
Now, add another edge with least cost



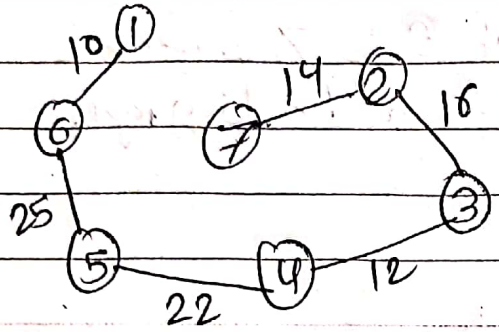




Now, another edge is 4,7 but it forms cycle so exclude it.

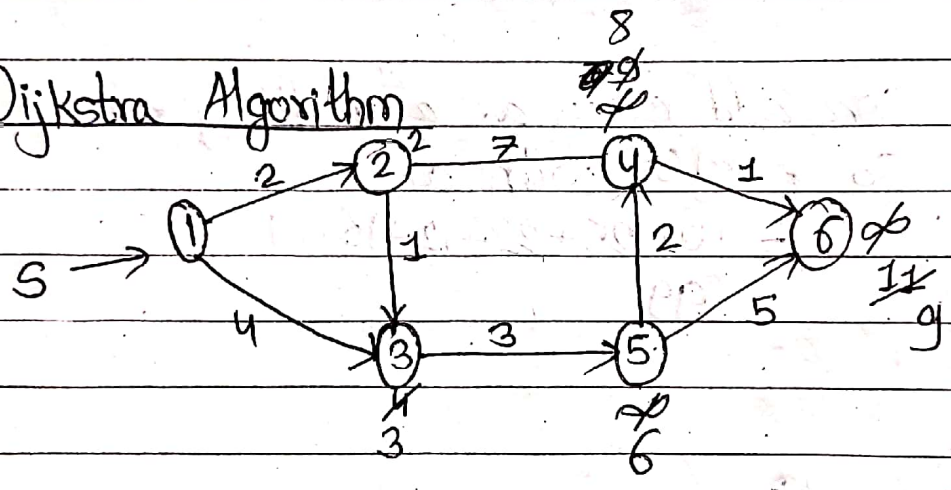


Again, edge 5,7 form cycle - So exclude



∴ Total cost = 99

* Dijkstra Algorithm



Path : 1, 2, 3, 5, 4, 6

Cost : 9

* Multistage Graph - Using Dynamic Programming :

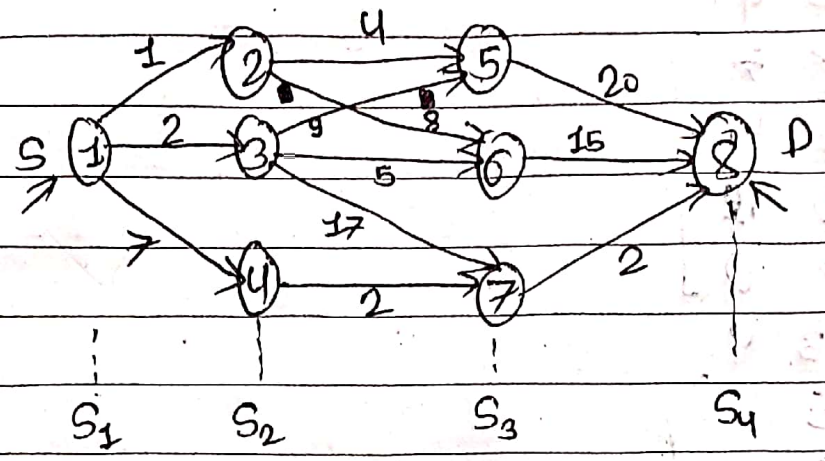
$$cost(i,j) = \min \{ c(j,l) + cost(i+1,l) \} \quad \text{if } S_i = S_{i-1}$$

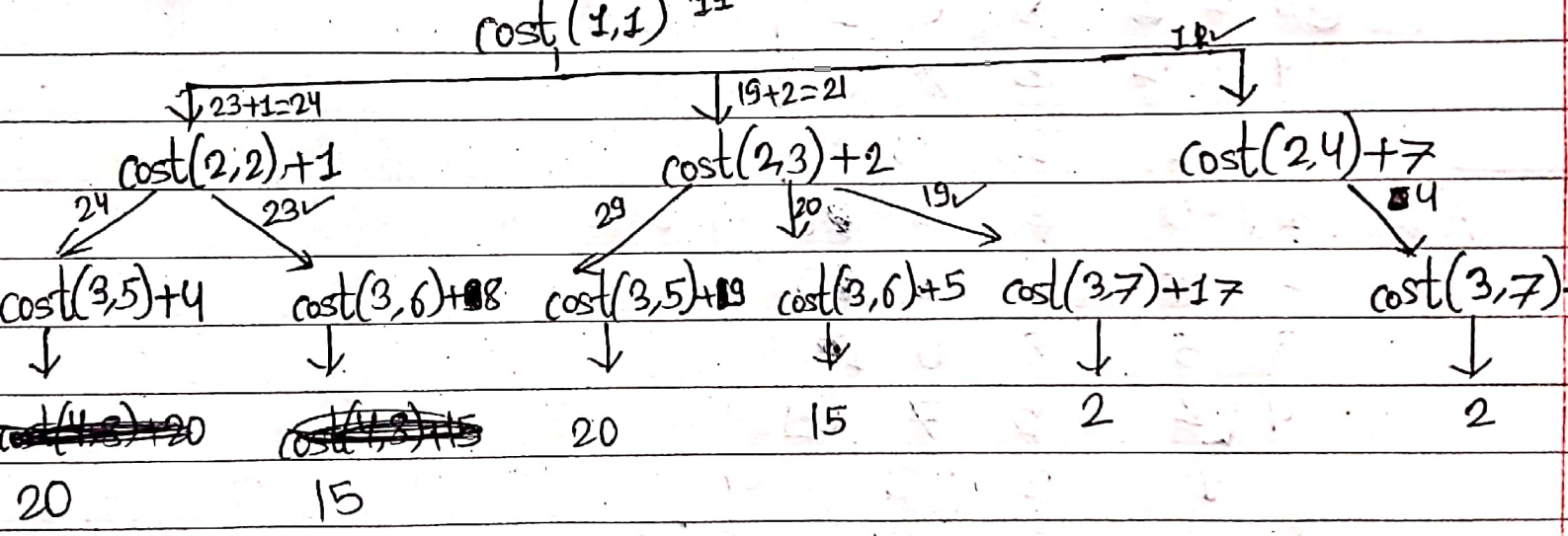
where,

$l \in V_{i+1}$ — V = vertices

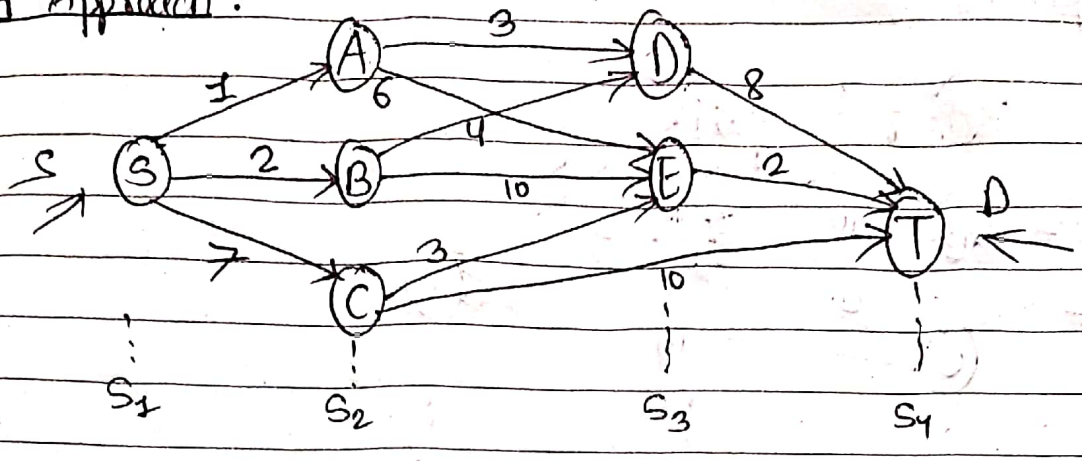
$\langle j,l \rangle \in E$, E = edge

Eg:





Forward Approach:



Sol?

At S_1:

start vertex = S

Stage 2:

$c(S,A) = 1$, $c(S,B) = 2$, $c(S,C) = 7$

Stage 3:

$c(S,D) = \min\{1 + c(A,D), 2 + c(B,D)\}$

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

$= \min\{4, 6\}$

$= 4$, S-A-D

$c(S,E) = \min\{1 + c(A,E), 2 + c(B,E), 7 + c(C,E)\}$

$= \min\{1+6, 2+10, 7+3\}$

$= \min\{7, 12, 10\}$

$= 7$, S-A-E

Stage 4:

$c(S,T) = \min\{c(S,D) + c(D,T), c(S,E) + c(E,T), c(S,C) + c(C,T)\}$

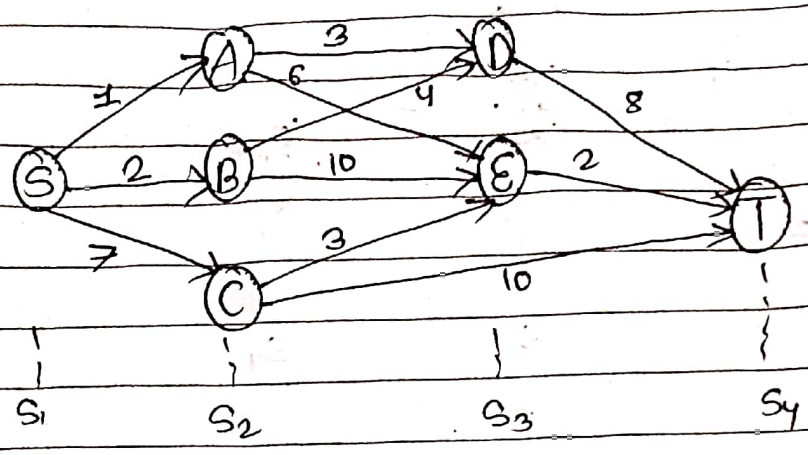
$= \min\{4+8, 7+2, 7+10\}$

$= \min\{12, 9, 17\} = 9$, S-A-E-T

$\therefore$ cost = 9

shortest Path = S-A-E-T

Backward Approach:



Sol?

Stage 4:

Last vertex = T

Stage 3:

$c(D,T) = 8$

$c(E,T) = 2$, shortest path: E-T

Stage 2:

$c(A,T) = \min\{c(A,D)+c(D,T),\ c(A,E)+c(E,T)\}$

$= \min\{3+8,\ 6+2\}$

$= \min(11, 8)$

$= 8$ A-E-T

$c(B,T) = \min\{c(B,D)+c(D,T),\ c(B,E)+c(E,T)\}$

$= \min\{4+8;\ 10+2\}$

$= \min\{12, 12\}$

$= 12$ B-D-T, B-E-T (both)

$c(C,T) = \min\{c(C,E)+c(E,T),\ c(C,T)\}$

$= \min\{3+2, 10\} = 5$ C-E-T

Stage 1:

$$c(S,T) = \min\{c(S,A)+c(A,T),\ c(S,B)+c(B,T),\ c(S,C)+c(C,T)\}$$

$$= \min\{1+8,\ 2+12,\ 7+5\}$$

$$= \min\{9, 14, 12\}$$

$$= 9, \quad S-A-E-T$$

cost = 9

path = S-A-E-T

Warshall Algorithm: $R^{(k)}[i,j] = R^{(k-1)}[i,j]$ OR $[R^{(k-1)}[i,k]$ AND $R^{(k}$

$A^0 =$

	1	2	3	4
1	0	3	∞	7
2	8	0	2	∞
3	5	∞	0	1
4	2	∞	∞	0

$A^1 =$

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	12
4	2	5	∞	0

$A^2 =$

	1	2	3	4
1	0	3	5	7
2	8	0	2	15
3	5	8	0	1
4	2	5	7	0

$A^3 =$

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

$A^4 =$

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

** Algorithms

* Push on stack

1) If (top == max-1)
   Write " Stack is full";
   Go to step 4.
2) top = top+1
3) st[top] = item
4) Stop

* Pop on stack

1) If (top < 0)
   Write " Stack is Empty";
   Go to Step
2) item = st[top];
3) return item;
4) top = top-1;
5) Stop

* Queue :

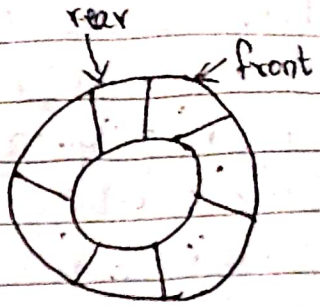
Enqueue (Q[max], Rear, Front, item)

1) If Rear = max-1. Then,
   Display "Queue is Full".
   Go to step 5.
   End If
2) If Rear = front = -1. Then,
   Front ← Front + 1
   End If
3) Rear ← Rear + 1
4) Q[Rear] ← item
5) Stop

Dequeue (Q[max], Rear, Front)

1) If Rear = Front = -1. Then,
   Display "Queue is empty".
   Go to step 5.
   End If
2) item ← Q[front]
3) If Rear = Front [redacted]. Then,
   Front ← -1;
   Rear ← -1;
   Else
   Front = Front + 1;
   End If
4) Return item
5) Stop

※ Circular Queue :



add (Queue[max] , item)

1) If (front == (rear+1) %. max). Then,
   Write "Queue is full".
   Go to step 2
   Else
      If (front == -1). Then,
         front = 0;
         rear = 0;
      Else
         rear = ((rear+1) %. max);
      Queue[rear] = item;
   End If
2) Stop

remove (Queue[max] , item)

1) If (front == -1) . Then,
      Write "Queue is empty".
      Go to step 2.
   Else
      item = Queue[front];
      If (front == rear). Then;
         front = -1;
         Rear = -1;
      Else
         front = ((front+1) %. max);
      End If
      //Item deleted.
   End If
2. Stop

Polynomial Problem (P):

- A problem is said to be Polynomial problem (P) if it can be solved in polynomial time using deterministic algorithm, like $O(n^k)$, where k is constant

  Eg: linear search, binary search, insertion sort, merge sort

Non-Polynomial Problem (NP):

- A problem that can not be solved in ~~program~~ polynomial time but can be verifiable in polynomial time using non-deterministic algorithm is known as ~~Non-Polynomial Problem.~~ Non-Deterministic Polynomial Problem.

  Eg: TSP, Sudoku, Scheduling problem

NP Hard:

- A problem that can not be solved in Polynomial time is called NP Hard problem.

  Eg: Subset sum problem

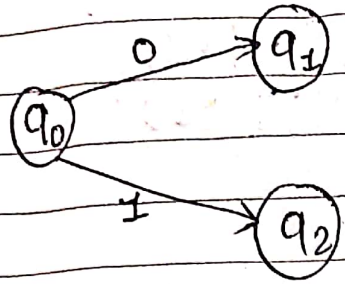
NP-Complete:

- If a problem A can be reduced into NP-problem B in polynomial time then problem A will be NP complete problem.

  Eg: Graph colouring problem, longest path problem, Hamilton path problem, etc.

Deterministic algorithm

1) Given a particular input, the computer will always give same output.

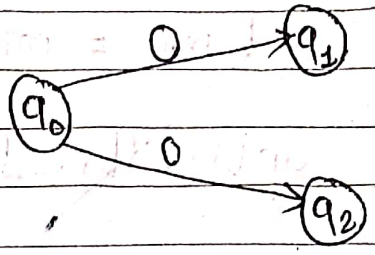
2) 

3) Can solve problem in polynomial time NP.

4) Can ~~not~~ determine what is the next step.

Non-deterministic algorithm

1) In this case, the computer did not understand what is the actual output.

2) 

3) Can not solve the problem in polynomial time NP.

4) Can not determine next step.

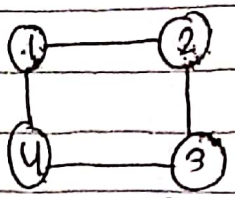
Minimum Edit Distance

convert abcdef to azced : 3 tasks ~~to do~~ can be use :- insert
- delete
- replace

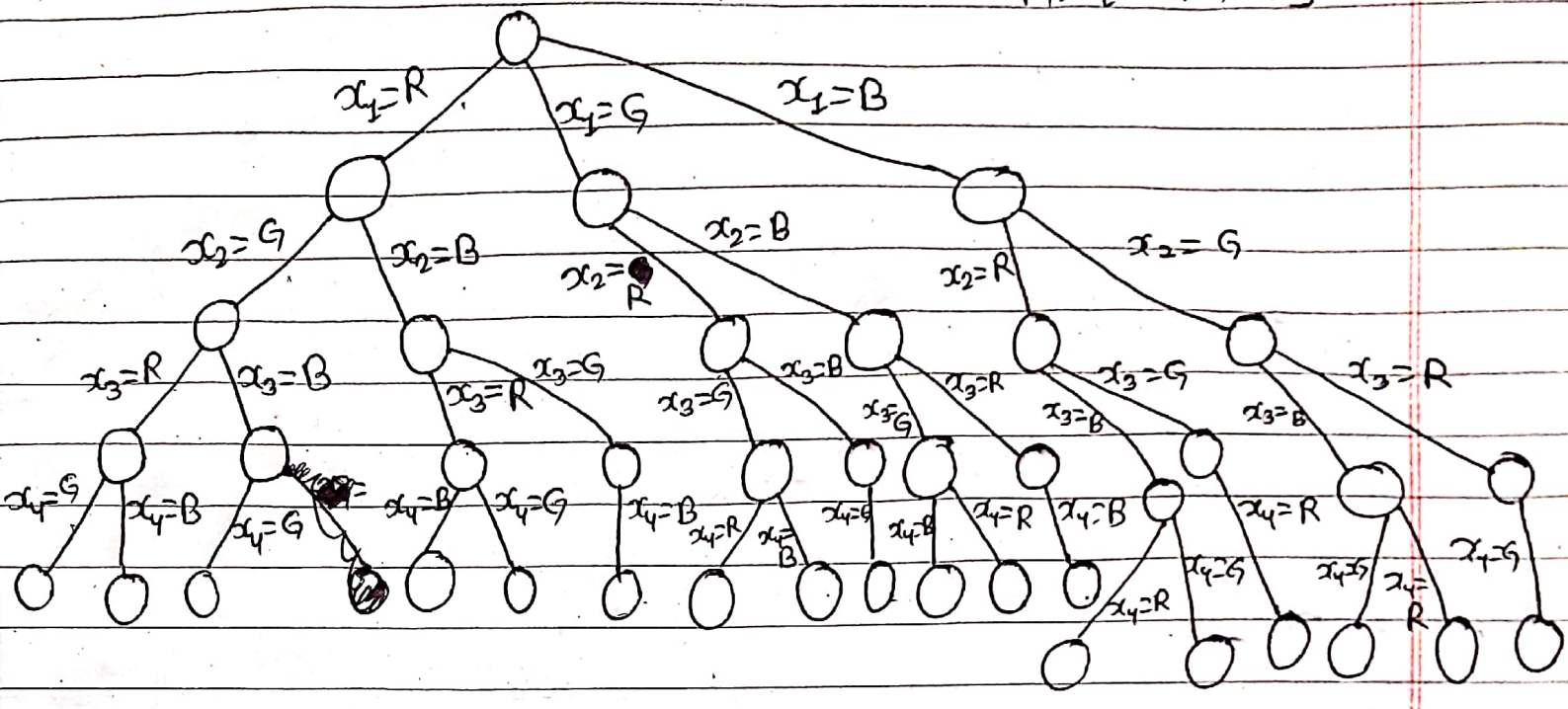
	0	a	b	c	d	e	f
0	0	1	2	3	4	5	6
a	1	0	1	2	3	4	5
z	2	1	1	2	3	4	5
c	3	2	2	1	2	3	4
e	4	3	3	2	2	2	3
d	5	4	4	3	2	3	3

```
int minDistance (String str1, String str2)
{
    int[][] dp = new int[str1.length+1][str2.length
                                        +1];

for (int i=0; i< str1.length; i++)
{
    char c1 = str1.charAt(i);

    for (int j=0; j< str2.length; j++)
    {
        char c2 = str2.charAt(j);

        if (c1 == c2)
            { dp[i+1][j+1] = dp[i][j];
        }
        else{
        dp[i+1][j+1]=min{ dp[i][j]+1,
            dp[i][j+1]+1 , dp[i+1][j]+1};
        }
    }
}
```

Graph Colouring Problem (Backtracking) :



m = { R, G, B }



Solution

1) R, G, R, G
2) R, G, B, G
3) R, G, R, B
4) R, B, R, B
5) R, B, R, G
6) ~~R, B, G, B~~ R, B, G, B
7) G, R, G, R
8) G, R, G, B
9) G, B, G, B
10) G, B, G, R
11) G, B, R, B
12) B, R, B, R
13) B, R, B, G
14) B, R, G, R
15) B, G, B, G
16) B, G, B, R
17) B, G, R, G

Date Page

Graph Coloring (for planar graphs):

Let G be a graph and m be a given positive integer. We want to discover whether the nodes of G can be colored in such a way that no two adjacent nodes have the same color, yet only m colors are used. This is termed the m-colorabiltiy decision problem. The m-colorability optimization problem asks for the smallest integer m for which the graph G can be colored.

Given any map, if the regions are to be colored in such a way that no two adjacent regions have the same color, only four colors are needed.

For many years it was known that five colors were sufficient to color any map, but no map that required more than four colors had ever been found. After several hundred years, this problem was solved by a group of mathematicians with the help of a computer. They showed that in fact four colors are sufficient for planar graphs.

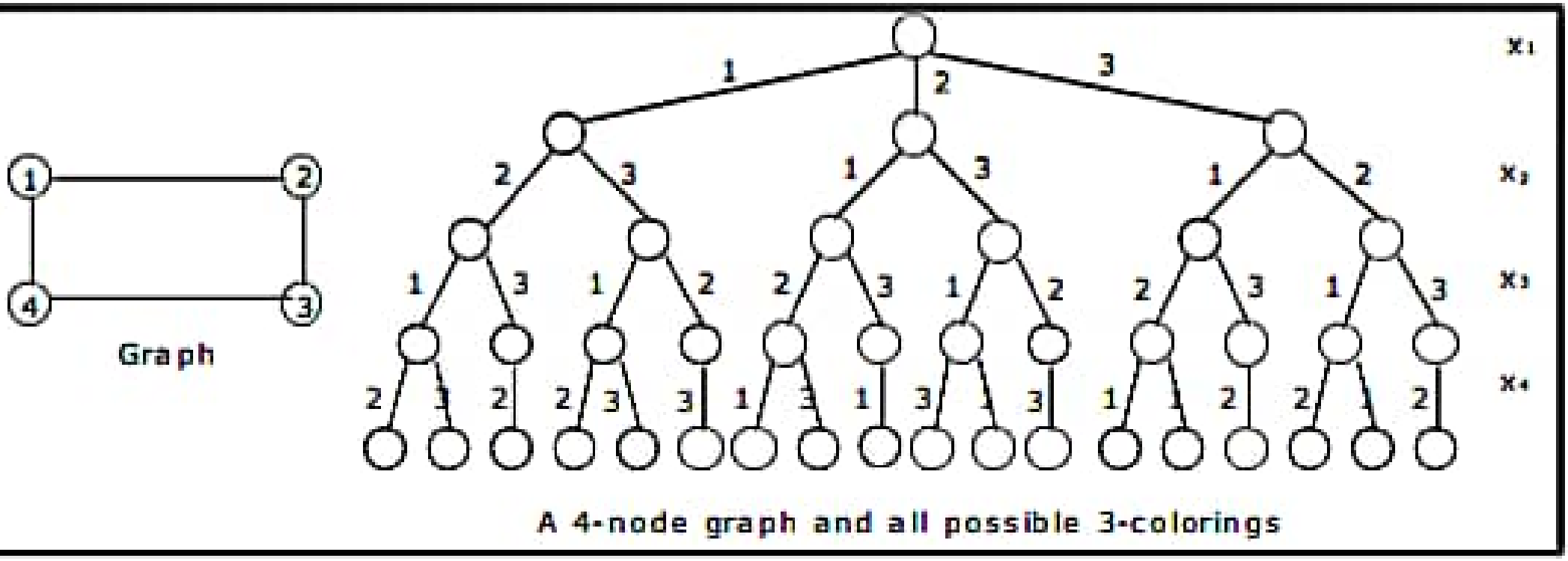
The function m-coloring will begin by first assigning the graph to its adjacency matrix, setting the array x [] to zero. The colors are represented by the integers 1, 2, . . . , m and the solutions are given by the n-tuple $(x_1, x_2, . . ., x_n)$, where x_i is the color of node i.

A recursive backtracking algorithm for graph coloring is carried out by invoking the statement mcoloring(1);

```
Algorithm mcoloring (k)
// This algorithm was formed using the recursive backtracking schema. The graph is
// represented by its Boolean adjacency matrix G [1: n, 1: n]. All assignments of
// 1, 2, . . . . . , m to the vertices of the graph such that adjacent vertices are assigned
// distinct integers are printed. k is the index of the next vertex to color.
{
        repeat
        {                                               // Generate all legal assignments for x[k].
                NextValue (k);          // Assign to x [k] a legal color.
                If (x [k] = 0) then return;     // No new color possible
                If (k = n) then         // at most m colors have been
                                                // used to color the n vertices.
                        write (x [1: n]);
                        else mcoloring (k+1);
                } until (false);
        }

Algorithm NextValue (k)
// x [1] , . . . . x [k-1] have been assigned integer values in the range [1, m] such that
// adjacent vertices have distinct integers. A value for x [k] is determined in the range
// [0, m].x[k] is assigned the next highest numbered color while maintaining distinctness
// from the adjacent vertices of vertex k. If no such color exists, then x [k] is 0.
{
        repeat
        {
                x [k]: = (x [k] +1) mod (m+1)                   // Next highest color.
                If (x [k] = 0) then return;                     // All colors have been used
                for j := 1 to n do
                {       // check if this color is distinct from adjacent colors
                        If ((G [k, j] ≠ 0) and (x [k] = x [j]))
                        // If (k, j) is and edge and if adj. vertices have the same color.
                        then break;
                }
                if (j = n+1) then return;               // New color found
        } until (false);                        // Otherwise try to find another color.
}
```

Example:

Color the graph given below with minimum number of colors by backtracking using state space tree.

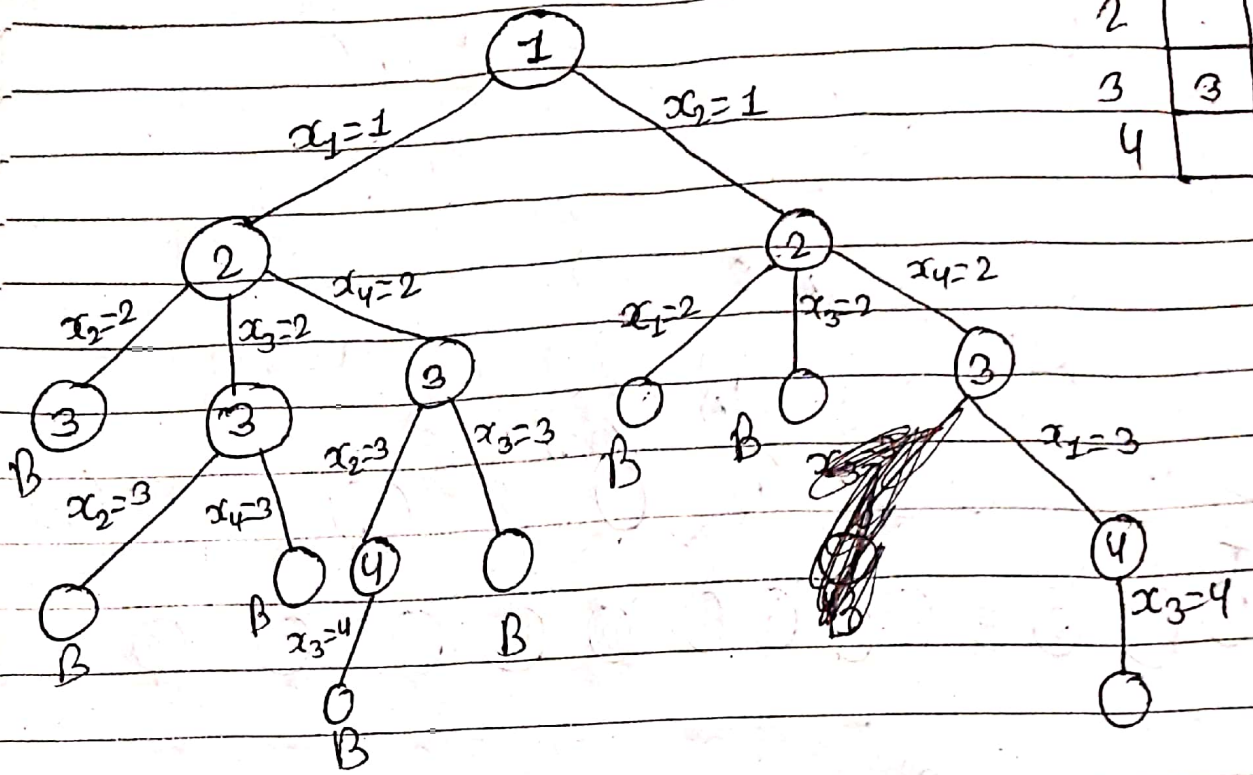


Backtracking

N-Queens

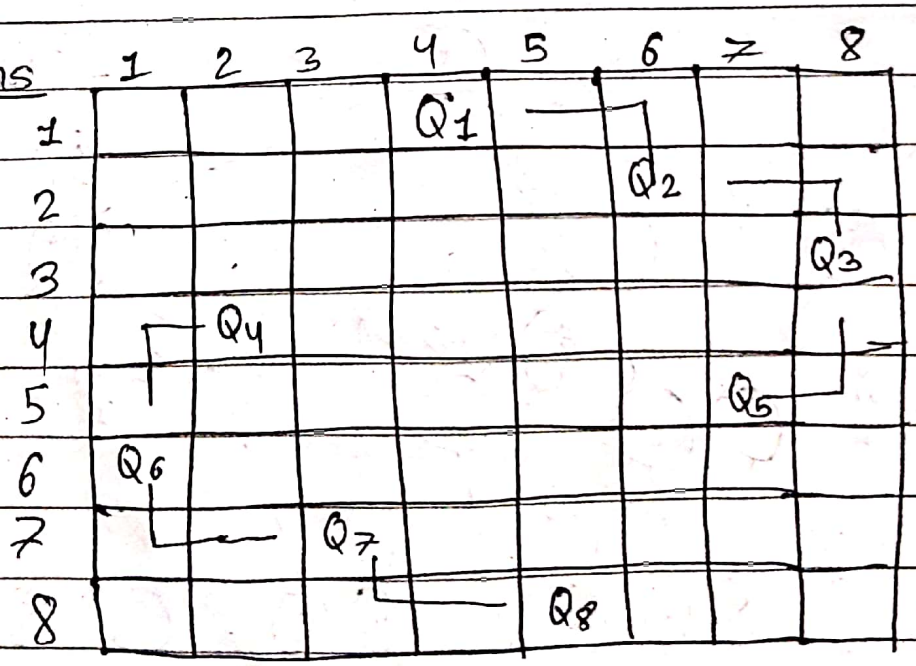
4-Queens

	x_1	x_2	x_3	x_4
1		1		
2				2
3	3			
4			4	



Required place for $\{1,2,3,4\} = \{x_2, x_4, x_1, x_3\}$

8-Queens



Hint

1- 4 col

8- 5 col

8×8

Soln

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

Date Page

Program for N-Queens Problem:

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>

int x[10] = {5, 5, 5, 5, 5, 5, 5, 5, 5, 5};

place (int k)
{
        int i;
        for (i=1; i < k; i++)
        {
                if ((x [i] == x [k]) || (abs (x [i] - x [k]) == abs (i - k)))
                return (0);
        }
        return (1);
}
nqueen (int n)
{
        int m, k, i = 0;
        x [1] = 0;
        k = 1;
        while (k > 0)
        {
                x [k] = x [k] + 1;
                while ((x [k] <= n) && (!place (k)))
                        x [k] = x [k] +1;
                if(x [k] <= n)
                {
                        if (k == n)
                        {
                                i++;
                                printf ("\ncombination; %d\n",i);
                                for (m=1;m<=n; m++)
                                printf("row = %3d\t column=%3d\n", m, x[m])
                                getch();
```

```
                }
                else
                {
                        k++;
                        x [k]=0;
                }
        }

        else
                k--;
    }
    return (0);
```



```
 }
main ()
{
        int n;
        clrscr ();
        printf ("enter value for N: ");
        scanf ("%d", &n);
        nqueen (n);
 }
```

```
}
main ()
{
        int n;
        clrscr ();
        printf ("enter value for N: ");
        scanf ("%d", &n);
        nqueen (n);
}
```

Output:

Enter the value for N: 4

Combination: 1		Combination: 2
Row = 1	column = 2	3
Row = 2	column = 4	1
Row = 3	column = 1	4
Row = 4	column = 3	2

For N = 8, there will be 92 combinations.