Roll No :- 160140221139
Name :- Gayudh Khatavkar

* Insertion Sort

Algorithm Insertion_sort (A, n)
// The algorithm takes as parameter an array A[1...n] and length n of the array.
// The array A is sorted in place: the numbers are rearranged within the array
// n is integer

```
FOR   j = 2 To length[A]
      key = A[j]
      {
      i = j-1
WHILE i>0 and A[i] >key
      {
          A[i+1] ← A[i]
i = i-1
A[i+1] = key
```

Time complexity for worst case Is :- $O(n^2)$;
Time complexity for best case Is :- $O(n)$

* Worst case for Insertion Sort

Consider an Array

15	20	10	30	50	18	5	45

Step 1 :- j=2 To length [A] i.e

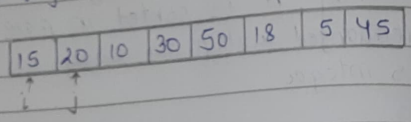
15	20	10	30	50	18	5	45

j=2 ... j=length of A

Step2 :- Key = A[j] i.e

Key = (j=2)
Key = 20

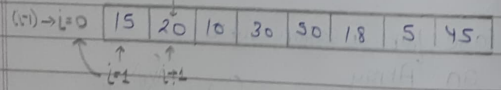
Step 3 :- i = j-1 i.e



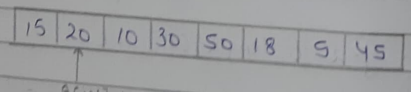
Step 4 :- While i>0 and A[i]>key
Since i=1 and A[i]=40 and key=20
For conditions is i>0
1>0

2nd condition is A[i]>key
40>20
Since both condition are true while loop will run

Step 5 :- A[i+1] = A[i]
i = i-1 i.e



Step 6 :- A[i+1] = key

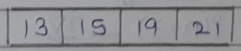


Step 7 :- Similarly j will increment and array will get sorted

Step 8 :- Here FOR loop will run n time and WHILE will lop n^2 times

∴ Time complexity for worst case is $O(n^2)$

* Best case for Insertion Sort
Consider an array :-



Since the array is already sorted only FOR loop will run n times and no WHILE loop will run
∴ Time complexity for best case is $O(n)$

* Methods of Solving various Problem
1) There are 3 methods to solve the problem
   i) Iterative Method
   ii) Binary Tree Method
   iii) Master Method

* Solving Recursion Problems By Iterative Method
I) Ex 1 :-

```
Void Test (int n)
{
  for(i=0;i<n;i++)      i will run time
  {
  Statement
  }
  Test (n-1) .... f(n-1)
}
```

→ $f(n) = n + f(n-1)$ — I
Delay eqn I by 1

$f(n-1) = (n-1) + f(n-2)$ — II

Putting II in I

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

$\therefore f(n) = n + n - 1 + f(n-2)$

$\therefore f(n) = 2n - 1 + f(n-2)$

$\therefore f(n) = 2n + f(n-2)$ III ... $2n-1 \approx 2n$ [-1 is constant]

Delay eqn II by 1

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

$f(n-2) = (n-2) + f(n-3)$ — IV

Put eqn IV in III

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

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

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

⋮

$f(n) = kn + f(n-k)$

When $k = n$

$f(n) = n^2 + f(0)$

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

$\therefore$ Time complexity is $O(n^2)$

2) Ex:-2 :- Void Test (n)

```
for(i=0; i<n; i=i*2)   — i will run log2 n
{
  Statement
  ;
  Test (n-1)   — f(n-1)
  ;
}
```

⟶ $f(n) = \log_2 n + f(n-1)$ — I

Delay eqn I by 1

$f(n-1) = \log_2 (n-1) + f(n-2)$ — II

Put eqn II in I

$f(n) = \log_2 n + \log_2 (n-1) + f(n-2)$

$f(n) = \log_2 n + \log_2 (n-1) + f(n-2)$ — III

Delay eqn II by 1

$f(n-2) = \log_2 (n-2) + f(n-3)$ — IV

Put eqn IV in III

$f(n) = \log_2 n(n-1) + \log_2 (n-2) + f(n-3)$

⋮

$f(n) = \log_2 [n-(k-1)] + \ldots + \log_2 (n-2) + \log_2 (n-1) + \log_2 n + f(n-k)$

When $k = n$

$f(n) = \log_2 1 + \ldots + \log_2 (n-2) + \log_2 (n-1) + \log_2 n + f(0)$

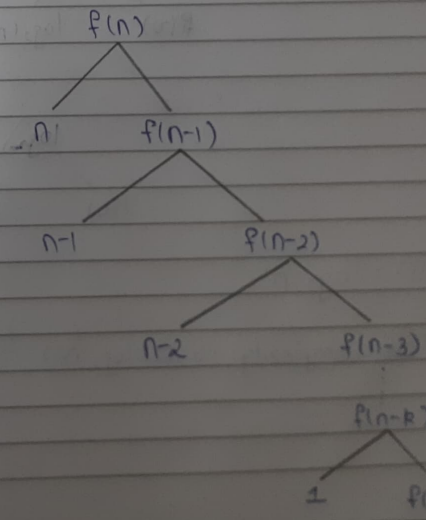
$f(n) = \log_2 n! + 1$

$f(n) = \log_2 n!$

$\therefore$ Time complexity is $O(\log_2 n!)$

* Solving Recursion by Binary Tree Method

Ex 1:- Same as Iterative Method example 1



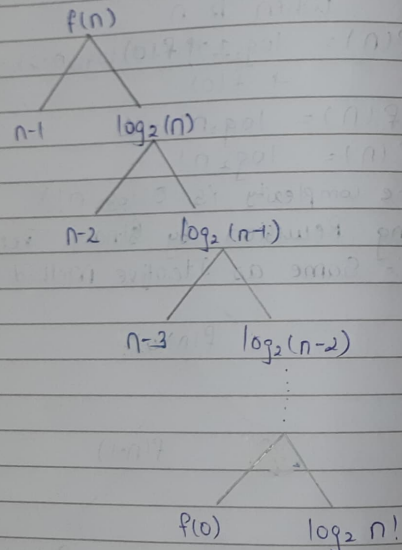
$f(n) = 1 + \dots + (n-2) + (n-1) + n$

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

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

∴ Time complexity is $O(n^2)$

2) Ex 2:- Same as Iterative method example 2



$f(n) = \log_2 n!$

∴ Time complexity is $O(\log_2 n!)$

* Master Method

1) Uses principle of Divide and Conquer method
2) It is calculated using:-

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$, $b > 1$ and $f(n) = \Theta(n^c)$

$$\therefore T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^c)$$

where a = no of size problem in recursion

$\frac{n}{b}$ = size of each sub problem

n = size of problem

$f(n) = \Theta(n^c)$ = work done outside recursion problem

* Cases

Since $f(n) = \Theta(n^c)$

Case 1:- $c < \log_b a$

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

Case 2:- $c = \log_b a$

$$T(n) = \Theta(n^c \log_2 n)$$

Case 3:- $c > \log_b a$

$$T(n) = \Theta[f(n)]$$

* Examples

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

→ Hae, $a = 2$

$b = 2$

$c = 2$

$f(n) = n^2$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_2 2 = \frac{\log_2 2}{\log_2 2} = 1$

$\therefore c > \log_b a$

i.e $2 > 1$

$\therefore T(n) = \Theta(n^2)$

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

⟶ Here, $a = 8$

$b = 4$

$c = 2$

$f(n) = n^2$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_4 8 = \frac{\log_2 8}{\log_2 4}$

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

$= \frac{3 \log_2 2}{2 \log_2 2}$

$= \frac{3}{2}$

$\therefore \log_4 8 = 1.5$

$\therefore c > \log_b a$

i.e $2 > 1.5$

$T(n) = \Theta(n^2)$

3) $T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$

⟶ Here, $a = 8$

$b = 2$

$c = 2$

$f(n) = n^2$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_2 8 = \frac{\log_2 8}{\log_2 2}$

$= \frac{\log_2 2^3}{\log_2 2^1}$

$= \frac{3 \log 2}{\log_2 2}$

$\therefore \log_2 8 = 3$

$\therefore c < \log_b a$

i.e $2 < 3$

$\therefore T(n) = O(n^{\log_a b})$

$\therefore T(n) = \Theta(n^3)$

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

⟶ Here, $a = 1$

$b = 10/9$

$c = 1$

$f(n) = n$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_{10/9} 1 = \frac{\log_2 1}{\log_2 \frac{10}{9}}$

$\therefore \log_{base} 1 = 0$ — $\log 1 = 0$

$\therefore c > \log_b a$

i.e $1 > 0$

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

5) $T(n) = 4T\left(\frac{n}{2}\right) + n^3$

→ Here, $a = 4$

$b = 2$

$c = 3$

$f(n) = n^3$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_2 4 = \frac{\log_2 4}{\log_2 2}$

$= \frac{\log_2 2^2}{\log_2 2}$

$= \frac{2 \log_2 2}{\log_2 2}$

$\log_2 4 = 2$

$\therefore c > \log_b a$

i.e $3 > 2$

$\therefore T(n) = \Theta(n3)$

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

→ Here, $a = 4$

$b = 2$

$c = 1$

$f(n) = n$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_2 4 = \frac{\log_2 4}{\log_2 2}$

$= \frac{\log_2 2^2}{\log_2 2}$

$= \frac{2 \log_2 2}{\log_2 2}$

$\log_2 4 = 2$

$\therefore c < \log_b a$

i.e $1 < 2$

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

$\therefore T(n) = \Theta(n^2)$

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

→ Here, $a = 2$

$b = 2$

$c = 1$

$f(n) = n$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_2 2 = \frac{\log_2 2}{\log_2 2}$

$\log_2 2 = 1$

$c = \log_a b$

i.e $1 = 1$

$\therefore T(n) = \Theta(n^c \log n)$

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

8) $T(n) = 3T\left(\frac{2n}{6}\right) + n$

→ Here, $a = 3$

$b = 6/2 = 3$

$c = 1$

$f(n) = n$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_3 3 = \frac{\log_2 3}{\log_2 3}$

$\log_3 3 = 1$

$\because c = \log_b a$

i.e $1 = 1$

$\therefore T(n) = \Theta(n^c \log n)$

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

9) $T(n) = 16T\left(\frac{n}{8}\right) + n^2$

→ Here, $a = 16$

$b = 8$

$c = 2$

$f(n) = n^2$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_8 16 = \frac{\log_2 16}{\log_2 8}$

$= \frac{\log_2 2^4}{\log_2 2^3}$

$= \frac{4 \log_2 2}{3 \log_2 2}$

$\therefore \log_8 16 = \frac{4}{3} = 1.33$

$\because c > \log_b a$

i.e $2 > 1.33$

$\therefore T(n) = n^2$

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

→ Here, $a = 1$

$b = 2$

$c = 0$

$f(n) = n^0$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\log_2 1 = \frac{\log_2 1}{\log_2 2}$

$\log_2 1 = 0$

$\because c = \log_b a$

i.e $0 = 0$

$\therefore T(n) = \Theta(n^c \log n)$

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

MODULE 2

ALGORITH DESIGN TECHNIQUE

2.1 Divide and Conquer Technique (DAC)

The divide and conquer strategy used in algorithm works only if there is a way to club together the solution of the sub-problems.

Step 1:- Divide the domain into SMALL (atomic level), where SMALL is the basic unit

Step 2:- Solve individual sub-problem using the solution of SMALL

Step 3:- Combine the sub-solutions to get the final answer

* General Algorithm for Divide and Conquer Technique

```
Algorithm DAnd C (P)
{
if Small (P) then return S(P)
else
{
divide P into smaller instances P1, P2, ... Pk, k>=1;
Apply DA and C to each of these subproblems;
return Combine (DAnd C(P1), DA and C(P2), .... DAnd
  C(Pk));
}
}
```

* Maximum / Minimum Algorithm without using DAQC (Iterative Method)

```
VOID Straight Max Min (Type a[], int n, Type &max,
                       Type &min)
// Set max to the maximum and min to the
minimum of a[1:n]
// Array is starting from index 1
{
max = min = a[1];
FOR (int i=2; i<=n; i++)
 {
IF (a[i] > max) then
    max = a[i];
IF (a[i] < min)
    min = a[i]
 }
}
```

* Maximum / Minimum Algorithm Using DAQC (Recursive Method)

```
VOID MaxMin (int i, int j, Type &max, Type &min)
// A[i:n] is a global array
// Parameters i, j are integers, 1<=i<=j<=n
// The effect is to set max and min to largest
and min to the smallest values in a[i:j]
{
IF (i==j)
max = min = a[i];
ELSE IF (i=j-1)  // for 2 elements
 {
IF (a[i] < a[j])
  {
```

```
max=a[j]
min = a[i]
}
ELSE
{
max = a[i]
min = a[j]
}
ELSE                    // for n bit
{
Type max 1, min 1
```

```
// If P is not small divide P into sub problem.
Find where to split the set
int mid = (i+j)/2

// Solve the sub problems
MaxMin(i, mid, max, min);
MaxMin(mid+1, j, max 1, min 1);

// Combine the solutions
IF (max < max 1) max = max 1;
IF (min > min 1) min = min 1;
}
```

* Master method on max min Algorithm (DAC method)

Here,

No. of time calling MaxMin is 2

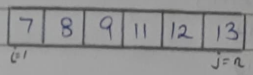
$\therefore a = 2$

No of time array is breaking into 2 parts

$\therefore b = 2$

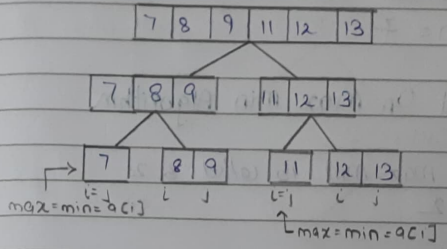
* Time Complexity For Max Min Algorithm

Consider an array



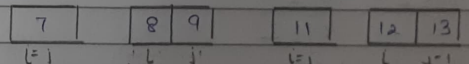
Step 1 :- IF (i = j)

max = min = a[i];



Step 2 :- ELSE If (i = j-1)

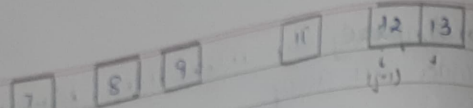
```
{
IF (a[i] < a[j-1])
max = a[j]
min = a[i]
}
ELSE
{
max = a[i]
min = a[j]
ELSE
{
Type max 1, min 1
```



Since 8 < 9

max = 9

min = 8



Since 12<13
max = 13
min = 12

maximum = 13
& Minimum = 7

* Master Method On Max Min Algorithm

Here,

No of time max-min is called is 2
$\therefore a = 2$

No of time array is breaking is 2
$\therefore b = 2$

Work Done Outside recursive is
$f(n) = 2n^0$

$\therefore c = 0$

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

$\therefore T(n) = 2T\left(\frac{n}{2}\right) + 2n^0$

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_2 2 = \frac{\log_2 2}{\log_2 2}$

$\therefore \log_2 2 = 1$

$\therefore c < \log ab$

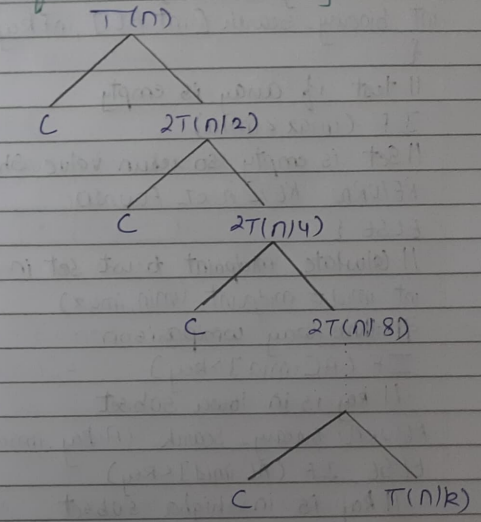
ie $1 > 0$

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

$T(n) = \theta(n)$

Time complexity is $O(n)$

* Tree Builder for max min Algorithm



$\therefore f(n) = O(n)$

$\therefore$ Time complexity is $O(n)$.

* Bineary Search Using (OA&C) Recursive Technique

// The algorithm takes as parameter an array A[1... n], the search key and lowa index pair of array
// output The algorithm returns index of the search key in the given array

```
int bineary_search (int A[], int key, int imin, int imax
{
// test if array is empty
IF (imax < imin)
// set is empty so return value showing not found
RETURN KEY_NOT_FOUND
ELSE {
// calculate midpoint to cut set in the half
int imid = midpoint (imin, imax)
// three way comparison
IF (A[imid] > key)
 // key is in lower subset
RETURN bineary_search (A, key, imid-1, imin);
ELSE IF (A[imid] < key)
 // key is in higha subset
RETURN bineary_search (A, key, imid+1, imax);
 ELSE
 // key has been found
 RETURN imid;
 }
}
```

* Time complexity for Bineary Search Technique

Consider an array

14	2	15	17	27	

Step 1 :- int bineary_search (int A[], int key, int imin, int imax)

{

IF (imax < imin)

RETURN KEY_NOT_FOUND

14	2	15	17	27	34

imin ↑ imax ↑

∴ imax > imin

Step 2 :- ELSE {

int imid = midpoint (imin, imax)

14	2	15	17	27	34

imin ↑ imid ↑ imax ↑

For Best Case :- Element to be search is 15 = key

Since A[imid] = key

∴ There will be further comparisons and

∴ Time complexity is O(1)

For Worst Case :- Element to be search is 14 = key

Step 3 :- IF (A[imid] > key)

RETURN bineary_search (A, key, imid-1, imin);

∵ 15 > 14

∴ key is in lower subset

with further iteration Key will be found at location 0

∴ Time complexity is O(log n)

* Master Method For Binary Search

Hae

No of time binary-search gets call is 1

$\therefore a = 1$

No of time away breaking is 2

$\therefore b = 2$

Work done outside recursive is 1

$\therefore f(n) = 1n^0$

$\therefore c = 0$

$\therefore T(n) = aT\left(\frac{n}{b}\right) + \theta(n^c)$

$\therefore T(n) = 1T\left(\frac{n}{2}\right) + 1n^0$

$\because \log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_2 1 = \frac{\log_2 1}{\log_2 2}$

$\therefore \log_2 1 = 0$

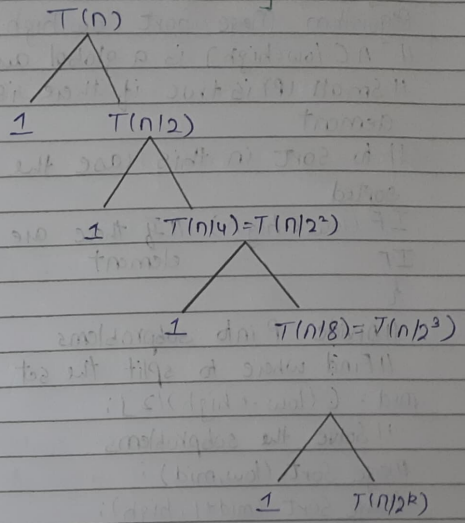
$\because c = \log_b a$

i.e $0 = 0$

$\therefore T(n) = \theta(n^c \log_2 n)$

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

* Tree Builda Method For Binary Search



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

$n = 2^k$

$\log n = k \log 2$

$k = \log_2 n$

$\therefore$ Time complexity is $O(\log_2 n)$

Merge Sort

```
Algorithm Merge Sort (low, high)
// A[low:high] is a global array to be sorted
// Small(P) is true if there is only one
element
// to sort in this case the list is already
sorted
IF (low < high): // If there are more than one
                    element
{
// Divide P into subproblems
// Find where to split the set
mid = [(low + high)/2];
// Solve the subproblems
Merge Sort (low, mid);
Merge Sort (mid+1, high);
// Combine the solutions
Merge (low, mid, high)
}
}
```

Algorithm for Merge Sort Function

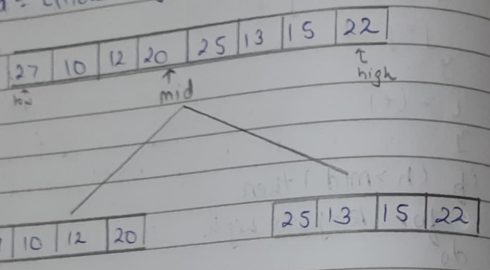
```
Algorithm Merge (low, mid, high)
// A [low:high] is a global array containining
two sorted
// subset in a [low:mid] and in a[mid+1:high]
// The goal is to merge these two sets into a
single set residing
// in A[low:high]. B[ ] is an auxillary
global array
h = high; i = low; j = mid+1;
while ((h ≤ 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+1
}
for k = low to high
  do
a[k] = B[k];
```

* Time complexity of Merge Sort

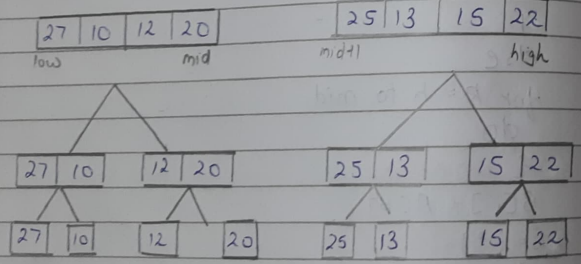
Consider an array

27	10	12	20	25	13	15	22

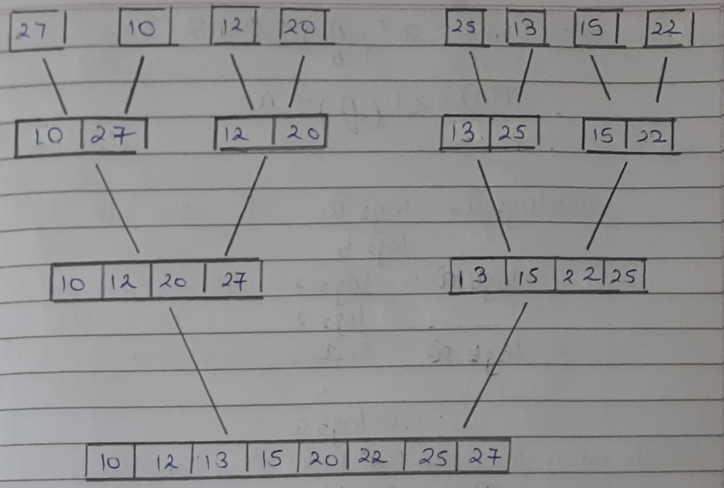
Step 1 :- If (low < high)
{
mid = [(low + high)/2];



Step 2:- Merge Sort (low, mid)
Merge Sort (mid+1, high)



Step 3:- Merge (low, mid, high)
}
}



Time complexity is $O(n \log n)$ for worst and best case both.
If the array is sorted still it will divide and compare all the element in the array
Hence the time complexity is same for both case

* Master method for Merge Sort

Here,
no of time mergesort is called is 2
$\therefore a = 2$

No of time array breaking is 2
$\therefore b = 2$

Work done outside recursion is
$f(n) = 1n$
$\therefore c = 1$

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

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

$\log_b a = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_2 2 = \frac{\log_2 2}{\log_2 2}$

$\therefore \log_2 2 = 1$

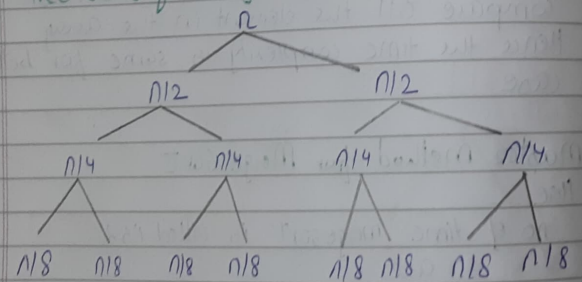
$\therefore c = \log_b a$

$i.e\ 1 = 1$

$\therefore T(n) = \Theta(n^c \log n)$

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

* Tree Build for Merge Sort



$n/2^k$

Let $n/2^k = 1$

$n = 2^k$

$k = \log n$

Since there are $\log_2 n$ levels, complexity is $O(n \log n)$

* Quick Sort

```
Algorithm QuickSort (p,q)
// Sorts the elements a[p], ..., a[q] which
// reside in the global
// array a[1:n] into ascending order; a[n+1] is
  considered to
// be defined and must be ≥ all the element
// in a[1:n]
{
if (p<q) then // If there are more than one element
{
// divide P into two subproblems
j = Partition (a, p, q+1);
// j is the position of the partition element
// Solve the subproblem
QuickSort (p, j-1);
QuickSort (j+1, q);
// There is no need for combining solutions
}
}
```

* Algorithm for Partition Function

```
Algorithm Partition (a,m,p)
// within a[m], a[m+1] ... a[p-1] the elements are
// rearranged in such a manner that if initially t=a[m].
// then after completion a[q]=t for some q
// between m and p-1, a[k]≤t for m≤k<q and
// a[k] ≥ t for q<k<p, q is returned.
// Set a[p] = ∞
```

```
{
v = a[m];
i = m;
j = p;
repeat
{
repeat
i = i+1
until (a[i] ≥ v);
repeat
j = j-1
until (a[j] ≤ v);
if (i<j) then interchange (a,i,j);
}
until (i ≥ j)
a[m] = a[j];
a[j] = v;
return j;
}
```

* Algorithm for Interchange function

```
// Exchange a[i] with a[j]
{
p = a[i];
a[i] = a[j];
a[j] = p;
}
```

* Time complexity for Quick Sort

I) For Best case consider an array,

0	1	2	3	4	5	6	7	8
65	70	75	80	85	60	55	50	45
p	i→						←j	

Step 1 :- i = 1 , j = 8

a[i] = 70 > 65 a[p]

a[j] = 45 ≯ 65 a[p]

i = i++ = 2

j = j-- = 7

Swap

Step 2 :-

65	45	75	80	85	60	55	50	70
p		i					j	

a[i] = 75 > 65

a[j] = 50 ≯ 65

i = i++ = 3

j = j-- = 6

Swap

:

continue till i > j

Step 3 :- When i > j

65	45	50	55	60	85	80	75	70
p			i	j				

Partition the array and swap p with last element j

Step 4:-

60	45	50	55	65
p	i			j

→ V

70	80	75	85
2	i		j

→ U

Step 5:- Keep sorting and final array is

45	50	55	60	65	70	75	80	85

∴ Time complexity is $O(n \log n)$

II) For worst case consider an array which contains all elements are sorted only the last element is unsorted

0	1	2	3	4	5
5	8	16	24	25	1
p	i				j

Step 1:-

$i=1$, $j=3$

$8 > 5$

$1 < 5$

$i = i++ = i=2$

$j = j-- = 6$

Swap

Step 2:-

5	1	16	24	25	8
p	i				

$16 > 5$

$25 < 5$

Keep decrementing ~~if~~ j unill the $a[j] < p$

Step 3:- Partition takes place

5	1

8	16	24	25

Swap

Step 4:- Final Array is

1	5	8	16	24	25

∴ Time complexity is $O(n^2)$

* Master method For Quick Sort

Here,

Quick Sort function is called 2 times

$\therefore a = 2$

No of times array is breaking is 2

$b = 2$

Work done outside recursive is n times

$\therefore f(n) = n$

$\therefore c = 1$

$\therefore T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^c)$

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

$\because \log_a b = \frac{\log_2 a}{\log_2 b}$

$\therefore \log_2 2 = \frac{\log_2 2}{\log_2 2}$

$\log_2 2 = 1$

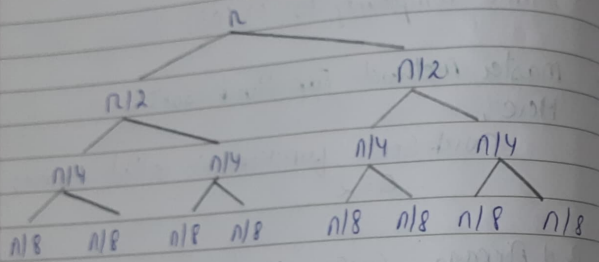
$\therefore c = \log_2 b$

$i.e\ 1 = 1$

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

* Tree Builder Method for Quick Sort

I) Best case



$n/2^k$

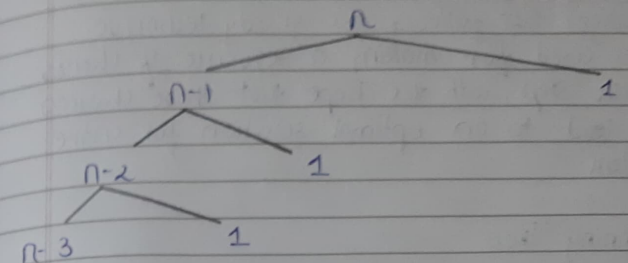
$\therefore n/2^k = 1$

$n = 2^k$

$k = \log n$

Since there $\log_2 n$ levels, complexity $O(n \log_2 n)$

I) For Worst case



$n - i$

$\therefore 1 + 2 + 3 + 4 + \cdots + n$

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

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

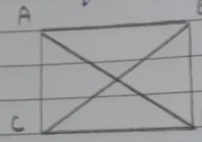
$\therefore$ Time complexity is $O(n^2)$

2.2 Greedy Technique

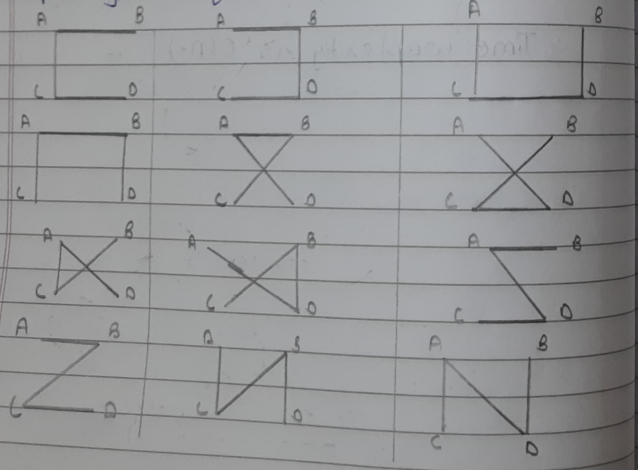
1) A greedy algorithm is an algorithmic paradigm that follows the greedy technique.
2) It is used for making a sequence of choices at each step, with the hope that these choices will lead to an optimal solution for entire problem

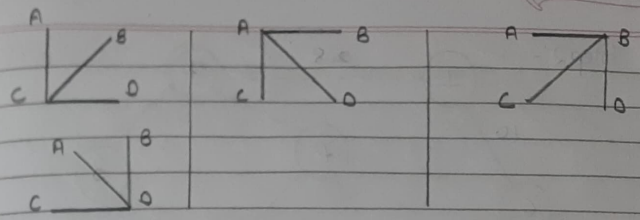
* Spanning Tree

1) A spanning tree in greedy techniques like Kruskal's or Prim's algorithm is a subset of graph edge that connects all vertices without forming cycles.



2) Spanning Trees of these graph are



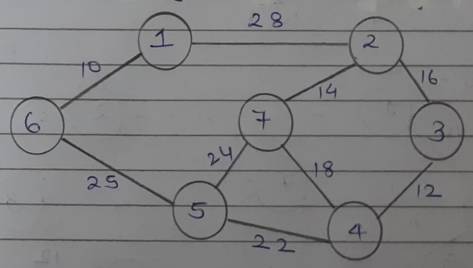


* Prims (Minimum Spanning Tree)

1) Prims algorithm is a greedy algorithm used to find minimum spanning tree (MST) of a connected, undirected graph.
2) It starts with an arbitary vertex, incrementally adds the shortest edge connecting vertex in MST to a vertext outside and repeats untill all vertices are invoded and no cycles.
3) Numericals:-

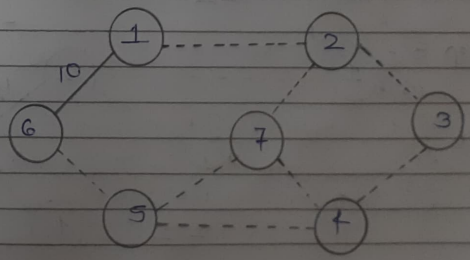
I) Find MST using Prims

a)

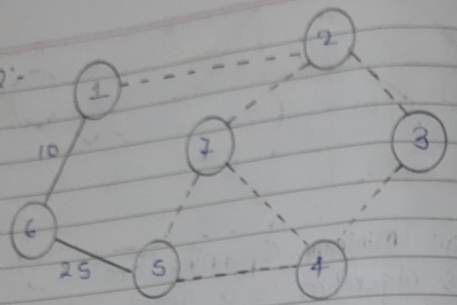


In Prims
i) select any root vertex
ii) Then select shortest edge from that vertex

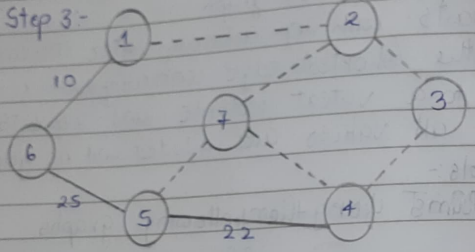
→ Step 1:-



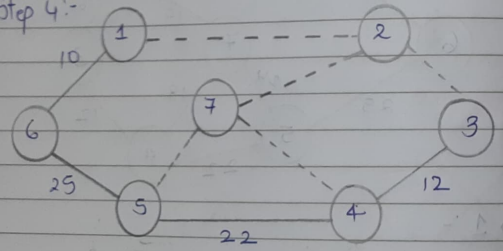
Step 2:-



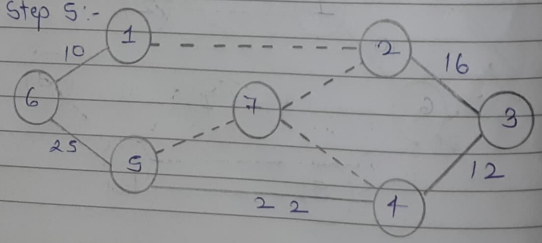
Step 3:-



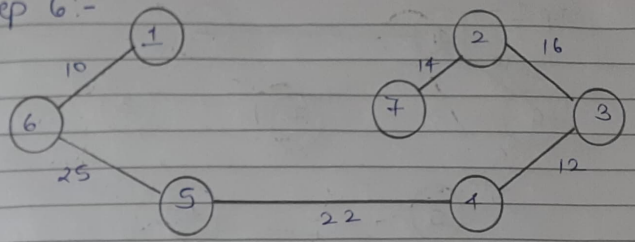
Step 4:-



Step 5:-



Step 6:-



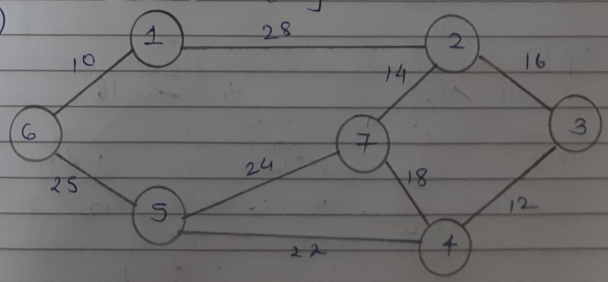
Minimum cost of Tree is
$= 10+25+22+12+16+14$
$= 99$

* Kruskal (Minimum Spanning Tree)

1) Kruskal algorithm is a greedy algorithm used to find the minimum spanning (MST) of a connected, undirected graph.
2) It works by sorting all the edges in ascending order of their weights and then iteratively adding the smallest edge that doesn't form a cycle in the growing MST
3) The process continues untill all vertices are included.
4) Numerical:-

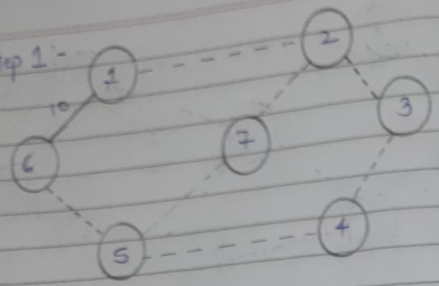
I) Find MST using Kruskal

a)

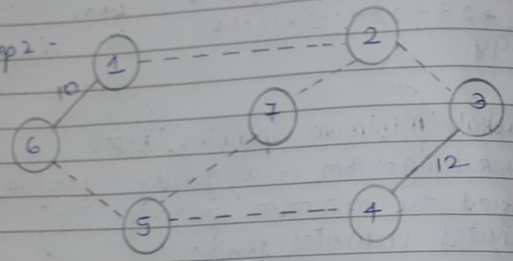


In Kruskal's select shortest edge

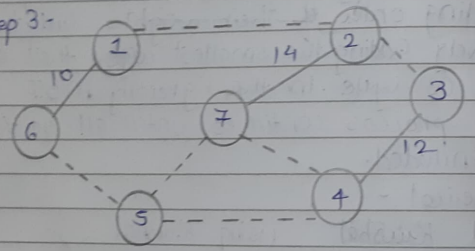
Step 1:-



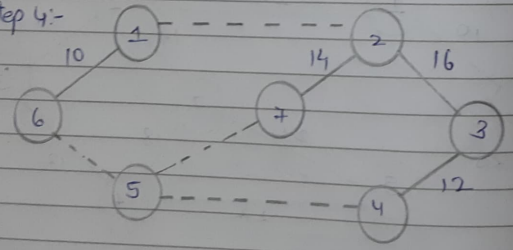
Step 2:-



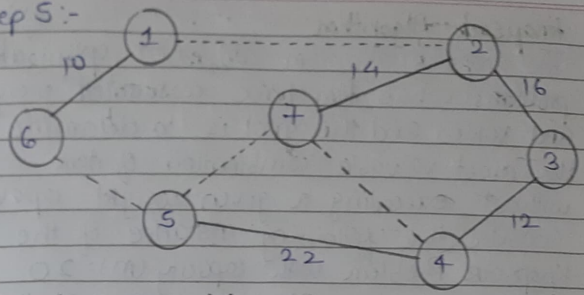
Step 3:-



Step 4:-

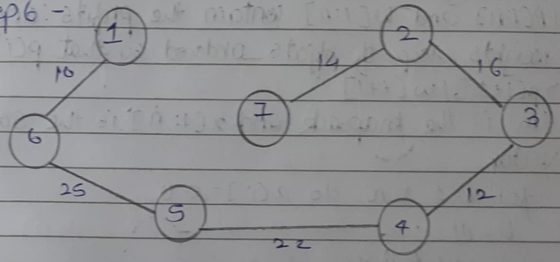


Step 5:-



18 won't be connected as it will form a closed cycle

Step 6:-



Again 24 won't be connect as it will form a close cycle

Total cost = 10+25+22+12+16+14 = 99

* General Method for Greedy Technique

```
Algorithm Greedy (a,n)
// a[1:n] contains the n inputs
Solution = 0; // Initialize the solution
for i = 1 to n do
  {
  x = select (a)
  if Feasible (solution x) then
  Solution = Union (solution, x);
  }
return solution; }
```

* Knapsack Algorithm

1) The knapsack algorithm addresses optimization problems where item have associated weights and values and the goal is to determine the most valuable combination of item without exceeding a given weight capacity.

2) Consider the following instance of the knapsack Problem with capacity (m) = 20

Objects	Obj 1	Obj 2	Obj 3
Profit(Pi)	25	24	15
Weight (wi)	18	15	10

// P[1:n] and w[1:n] contain the profits and // weights resp of objects ordered so that p[i]/w[i] >p[i+1]/w[i+1]
// m is the knapsack and x[1:n] is the solution vector

```
for i=1 to n do x[i]=0.0
  U=m
for i=1 to n do
{
  if U>0 and wi ≤ U
      U=U-Wi
      P=P+Pi
  else break
if (U>0)
P= P + Pi(U/wi);
```

Maximise $\sum_{1 \le i \le n} P_i x_i$
Subject to $\sum_{1 \le i \le n} w_i x_i \le m$
and $0 \le x_i \le 1, \quad 1 \le i \le n$

* Time/Space complexity for knapsack Algorithm

1) Time complexity of the sorting + Time complexity of the loop of the maximise profit = $O(N \log N) + O(N) = O(N \log N)$
2) Space complexity is $O(1)$

* Numerical

Find the maximum profit for maximum = 15

Object	1	2	3	4	5	6
Profit(Pi)	9	4	18	12	10	9
Weight(Wi)	5	5	7	2	1	1

→

Object	1	2	3	4	5	6	
Pi	9	4	18	12	10	9	
Wi	5	5	7	2	1	1	
Pi/Wi	1.8	1.8	2.57	6	10	9	
Order	E	F	D	C	A	B	→ Largest Pi/Wi first

	Wi	Taken	Balance	xi = Taken/Wi	Pi	$\sum x_i w_i$	$\sum P_i x_i$
A→	1	1	14	1	10	1	10
B→	1	1	13	1	9	1	9
C→	2	2	11	1	12	2	12
D→	7	7	4	1	18	7	18
E→	5	4	0	0.8	9	4	7.2
			Stop			Σ=15	Σ=56.2

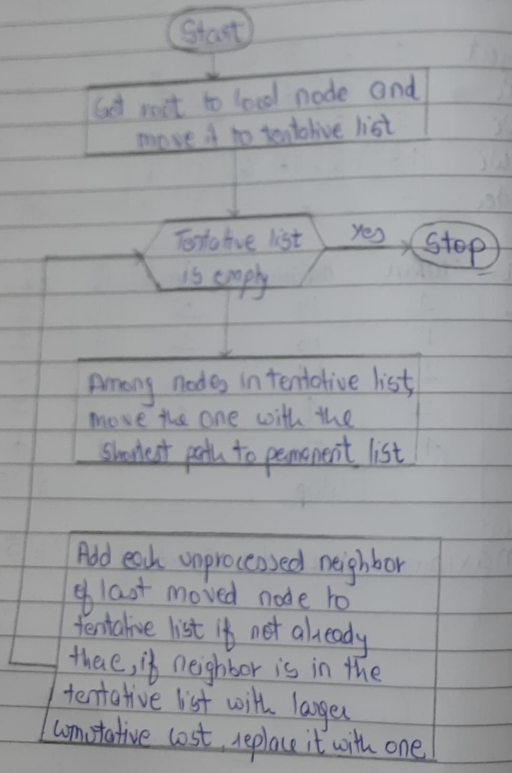
∴ maximum profit for capacity of 15 is 56.2

Dijkstra's Algorithm (Shortest Path Algorithm)

1) Dijkstra's Algorithm is for finding shortest path between nodes in a weighted graph
2) It works on both directed and undirected graph with non-negative weights

- Flowchart and Algorithm for Dijkstra's Algorithm

I) Flowchart



II) Algorithm

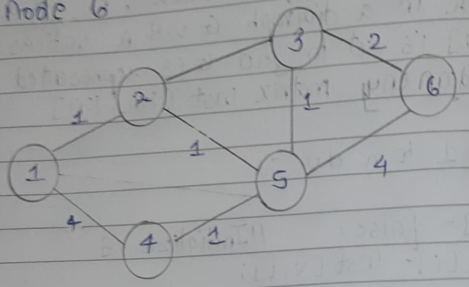
```
Algorithm Shortest Path (v, cost, dist, n)
// dist[j], 1 ≤ j ≤ n, is set to the length of
// the shortest path from vertex v to vertex j in
// a graph in a digraph G with n vertices.
// dist[v] is set to zero. G is represented by its
// cost adjacency matrix cost[1:n, 1:n]
{
  for i = 1 to n do
  {
    S[i] = false;      // Initialize S
    dist[i] = cost[v, i];
  }
  S[v] = true;
  dist[v] = 0.0;       // Put v in S
  for num = 2 to n-1 do
  {
    // Determine n-1 path from v. Choose u from
    // those vertices not in S that dist[u] is
    // minimum;
    S[u] = true;       // Put u in S
    for (each w adjacent to u with S[w] = false) do
      // Update distances
      if (dist[w] > dist[u] + cost[u, w])) then
        dist[w] = dist[u] + cost[u, w];
  }
```

- Notation for Dijkstra's Algorithm (Numerical)

S = Source node, N = Set of permanently labelled node
D_j = Minimum cost b/w source node & node j
always 0 ← D_S = Minimum cost b/w source node & source node
$D_j = \min(D_i + C_{ij})$ .. where C_{ij} = cost of i to j edge
D_i = permanent labeled node

Numerical

1) Find the shortest path for destination of node 6



→ $D_5 = D_1 = 0$

$N = \{1\}$

Nodes connected to node 1 are 2, 4

$\therefore D_2 = \min(D_1 + C_{12})$

$D_2 = (0+1)$

$D_2 = 1$

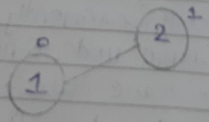
$D_4 = \min(D_1 + C_{14})$

$D_4 = 0 + 4$

$D_4 = 4$

Since D_2 is lowest node 2 will be added permanent node set

$N = \{1, 2\}$



$\therefore N = \{1, 2\}$

Nodes connected to node 1 are 4 → 2 has become permanent node

Nodes connected to node 2 are 5, 3

$\therefore D_4 = \min(D_1 + C_{14})$

$D_4 = 0 + 4$

$D_4 = 4$

$D_3 = \min(D_2 + C_{23})$

$D_3 = 1 + 3$

$D_3 = 4$

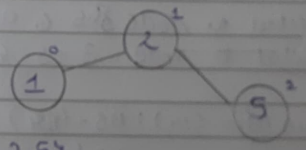
$D_5 = \min(D_2 + C_{25})$

$D_5 = 1 + 1$

$D_5 = 2$

Since D_5 is lowest, node 5 will be added permanent node set

$N = \{1, 2, 5\}$



$N = \{1, 2, 5\}$

Nodes connected to node 1 is 4

Nodes connected to node 2 are 5

Nodes connected to node 5 are 4, 3, 6

$D_4 = \min(D_1 + C_{14}; D_5 + C_{54})$

$D_4 = 0 + 4 \; ; \; 2 + 1$

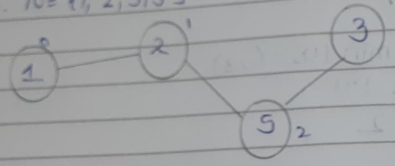
$D_4 = 4 \; ; \; 3$

$D_4 = 3$

$D_3 = \min(D_2 + C_{23}; D_5 + C_{53})$
$D_3 = (1+3)(2+1)$
$D_3 = 4; 3$
$D_3 = 3$
$D_6 = \min(D_5 + C_{56})$
$D_6 = 2+4$
$D_6 = 6$

Since D_3 is lowest, node 3 will be added in permanent node set

$\therefore N = \{1, 2, 5, 3\}$



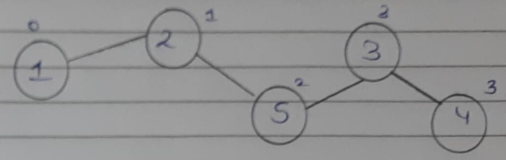
$\therefore N = \{1, 2, 5, 3\}$
Node connected to node 1 is 4
Node connected to node 2 is none
Node connected to node 5 is 6 and 4
Node connected to node 3 is 6

$D_4 = \min(D_1 + C_{14}; D_5 + C_{54})$
$D_4 = 0+4 \quad ; \quad 2+1$
$D_4 = 4 \quad ; \quad 3$
$D_4 = 3$

$D_6 = \min(D_6 + C_{56}; D_3 + D_{36})$
$D_6 = 2+4; 3+2$
$D_6 = 6, 5$
$D_6 = 5$

$\therefore$ Since D_4 is lowest, node 4 will be added in permanent node set

$N = \{1, 2, 5, 3, 4\}$



$\therefore N = \{1, 2, 5, 3, 4\}$
Node connected to node 1 is none
Node connected to node 2 are none
Node connected to node 5 is 6
Node connected to node 3 is 6
Node connected to node 4 is none

$\therefore D_6 = \min(D_5 + C_{56}; D_3 + C_{36})$
$D_6 = 2+4; 4+2$
$D_6 = 6; 6$

$\therefore$ Lastly node 6 will be added in permanent node set

$\therefore N = \{1, 2, 5, 3, 4, 6\}$