

Algorithms

• **Algorithm**:- a method or a process followed to solve a problem.
→ It takes input to a problem (function) and transforms it to the ~~input~~ output.

* Algorithm is a finite set of instructions that, if followed, accomplishes particular task.
It ^{must} satisfy following criteria:-

- 1) **Input**:- Zero or more quantities are externally supplied.
- 2) **Output**:- At least one quantity is produced.
- 3) **Definiteness**:- Each instruction is clear and unambiguous.
- 4) **Finiteness**:- If we trace out the instructions of an algorithm then for all cases, the algorithm terminates after finite no. of steps.
- 5) **Effectiveness**:- Every instruction must be very basic so that it can be carried out, it also must be feasible.

Applications:-

- Data Science → ~~Data~~ DNA analysis
→ Tweet Analysis
- E Commerce → Flipkart

• Algorithm Properties :-

- (a) It must be correct.
- (b) It must be composed of a series of concrete steps.
- (c) There can be no ambiguity as to which step will be performed next.
- (d) It must be composed of a finite number of steps.
- (e) It must terminate.

• Algorithm Efficiency :-

Two goals:

- (a) To design an algorithm that is easy to understand, code, debug.
- (b) To design an algorithm that makes efficient use of the computer resources.

Analysis

↓
Apriori

- Time
- Space
- Cost

↓
Posterior

A Priori Analysis

- 1) It is done before execution of an algorithm.

Ex: 1

- 2) Absolute analysis

- 3) It is independent of language of compiler and types of hardware / OS.

- 4) It will give approximate answer.

- 5) It uses the asymptotic notations to represent how much time the algorithm will take in order to complete its execution.

A Posteriori Testing

- 1) It is done after execution of algorithm. Or after writing the program.

- 2) Relative analysis

- 3) It is dependent on language of compiler and type of hardware / OS.

- 4) It will give exact answer.

- 5) It doesn't use asymptotic notations to represent the time complexity of an algorithm.

A priori analysis Example:-

① Buying a cellphone

- Budget
- User age gap
- Technical specification
- User reviews

② Buying a home

- Budget
- Area
- Location & locality
- Amenities

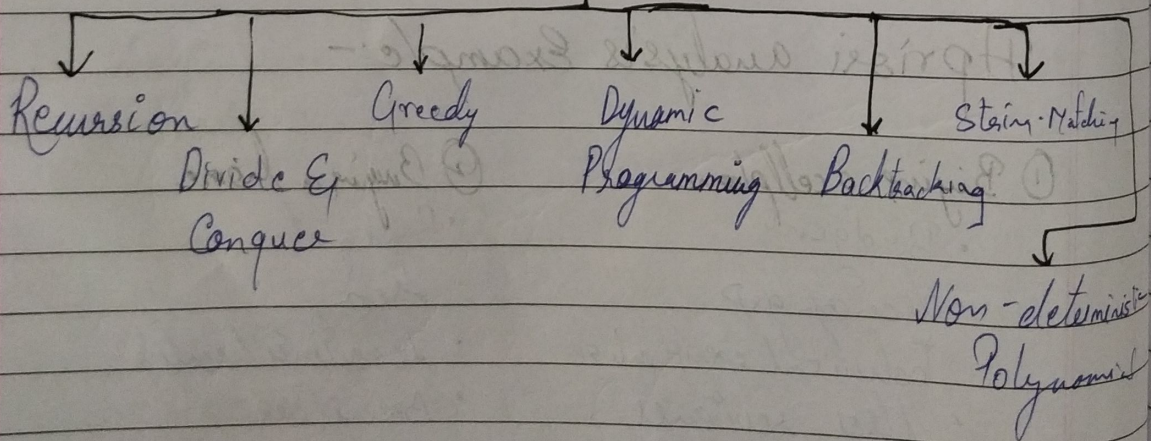
• How to analyze algorithms

- Time
- Space
- Performance Analysis
 - ① Best Case
 - ② Average Case
 - ③ Worst Case

* Specifications of good Algorithm

- ① Work ~~for~~ Correctly for all case
- ② Steps are clear
- ③ Effective Time utilization
- ④ Effective Space utilization
- ⑤ Give best solution

Algorithm Classification



Steps to Write an Algorithm:-

- Comments begin with // and continue until the end of line.
- Blocks are indicated with matching braces: { and }.
- A compound statement (i.e. a collection of single or simple statements) can be represented as a block. The body of a procedure also forms a block.
- Statements are delimited by ;
- An identifier begins with a letter.
- Assignment of values to variables is done using the assignment statement (variable) := (expression);
- There are two Boolean values true and false.
- The logical operators and, or, not.
- The relational operators $<$, $>$, $\leq$, $\geq$, $=$, $\neq$.
- Array indices start at zero.
- Loop:- for, while, repeat-until.
- Conditional statements:- if, if else.

Examples

1) If:-

```

If (n > 5) → ①
{
    statement → ①
}
Else → ①
    statement → ①
  
```

$$T(n) = 1 + 1 + 1 + \dots + 1 \\ = O(n^0)$$

2) For:-

Add main (s)

```

{
    s = 0
    for (i = 0; i < n; i++) → n+1
    {
        s = s + 1 → n
    }
    Return s;
}
  
```

$i = 0 \rightarrow$
 $0 \rightarrow 0$
 $1 \rightarrow \checkmark$
 $2 \rightarrow \checkmark$
 $\vdots$
 $n \rightarrow$

$$T(n) = n + 1 + n \\ = 2n + 1 = O(n)$$

$$S(n) = i, n, s \\ = 1 + 1 + 1 = 3 \\ = O(n^0)$$

3) Sum of two matrix $A + B = (A+B)^T$

Algo Add (A, B, n)

```
{
  for (i=0; i<n; i++) → n.
  {
    for (j=0; j<n; j++) → n.n
    {
      stmt; → n.n
    }
  }
}
```

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

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

4) Multiplication.

Algo Multiply (A, B, n)

```
{
  for (i=0; i<n; i++) → n.
  {
    for (j=0; j<n; j++) → n^2
    {
      stmt;
      for (k=0; k<n; k++) → n^3
      {
        stmt; → n^3
      }
    }
  }
}
```


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

$$T(n) = \underline{O(n^3)}$$

$$S(n) = 2n^2 + 4$$

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

5) while :-

$$I = 1$$

$$K = 1$$

while (K < n)

{

Statement

K++;

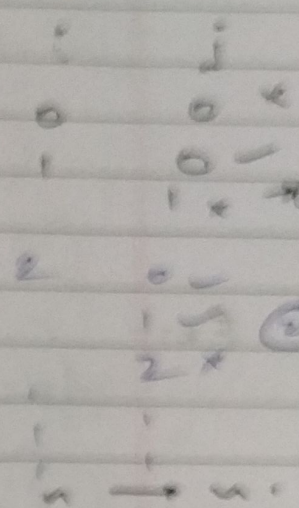
}

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

$$= O(n)$$

$$S(n) = 1 + 1 + \dots \\ = O(n)$$

6) for (i=0; i<n; i++)
 {
 for (j=0; j<i; j++)
 stmt;
 }

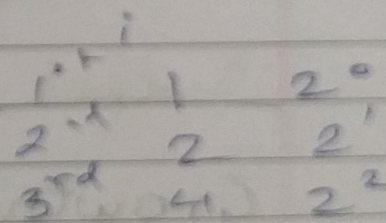


Time complexity = $1 + 2 + 3 + \dots + n$
 $= \frac{n(n+1)}{2}$

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

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

7) for (i=1; i<n; i=i*2)
 {
 stmt;
 }



Time Complexity :- $2^k \leq n$
 $k \log 2 = \log n$
 $k = \log_2 n$

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

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

Ex:-

for ($i=1; i \leq n; i++$) $\rightarrow n$

{

for ($j=1; j \leq n; j = j * 2$) $\rightarrow \log_2 n$

{

stmt; $\rightarrow \log_2(n)$

}

}

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

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

Growth of a Function.

Name	Running Time ($T(n)$)
Constant time	$O(1)$
Logarithmic time	$O(\log n)$
Linear time	$O(n)$

Linearithmic time $O(n \log n)$

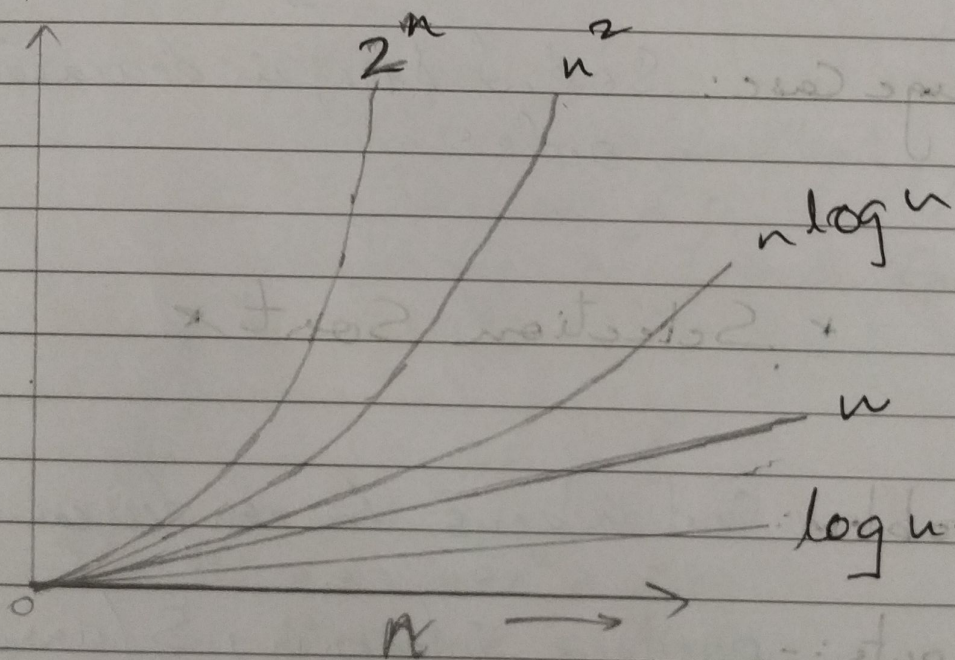
~~Qua~~ Quadratic time $O(n^2)$

Cubic time $O(n^3)$

Polynomial time $2^{f(\log n)} = \text{poly}(n)$

Factorial time $O(n!)$

Exponential time X^n



Complexity order:-

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

Best, Worst & Average Cases

Every input given of a particular size have different time consumed to run.

Task:- sort an array in increasing order.

Best Case: Sorted Array

Worst Case: Unsorted Array

Average Case: Sorted Array in decreasing order.

* Selection Sort *

Problem: Sort n keys in non decreasing order

Inputs:- positive integer n , S (array)

Outputs:- array S containing key in non decreasing order

Algorithm:-

```

void selectionsort (int n, keytype s[])
{
    index i, j, smallest;      → 1
    for (i = 1; i ≤ n; i++)    → n
    {
        smallest = i;          → n
        for (j = i + 1; j ≤ n; j++) → n · n
        {
            smallest = i;
            if (s[i] < s[smallest]) → n · n
            {
                smallest = j;
                exchange s[i] and s[smallest];
            }
        }
    }
}

```

$$T(n) = 1 + n + n + n^2 + n^2 + \dots$$

$$= 1 + 2n + 2n^2 + \dots$$

For Best Case. $n=6$

1	2	3	4	5	6
---	---	---	---	---	---

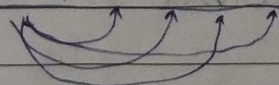
SA USA

1	2	3	4	5	6
---	---	---	---	---	---

5 → (n-1)

1	2	3	4	5	6
---	---	---	---	---	---

4 → (n-2)



$$\therefore 1+2+3+\dots+(n-2)+(n-1) = \frac{n(n-1)}{2}$$

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

$$\therefore T(n) = O(n^2) \rightarrow \text{Best Case.}$$

- For worst case:

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

Note:- Best Case for insertion sort is $O(n)$
Insertion is better.

• Asymptotic Analysis •

→ It is a method of describing limiting behavior.

Why Asymptotic Analysis?

→ It helps in choosing the best ~~or~~ algorithmic solution for a problem based on how particular solution will take to run.

→ One need not know exact values of comparable parameters.

Asymptotic complexity:- Way of expressing the main component of the cost of an algorithm using idealized units of computational work.

- Big O
 - Big Omega
 - Big Theta
- } (Growth Rate, Not Best, Avg & worst case)

➤ Big O:- For any non-negative function, $f(n)$ and $g(n)$, if there exists an integer n_0 and a constant $c > 0$ such that for all integers $n > n_0$,

$$f(n) \leq cg(n).$$

then $f(n)$ is Big O of $g(n)$

Q1) $f(n) = 3n + 5$. Find Big O

→ $O(n)$ but $O(n^2)$ $O(n^3)$ will also work but for a particular value of c .

Q2) $f(n) = n^2 + 10n$. Find Big O.

→ $n = 1$ then $1 + 10 = 11$

$$n^2 + 10n \leq 11n^2$$

$$\therefore c = 11 \text{ and } g(n) = n^2$$

$$f(n) \rightarrow O(n^2)$$

2) Big-Omega Ω :- For any non-negative functions $f(n)$ and $g(n)$, if there exists an integer n_0 and a constant $c > 0$ such that for integers $n > n_0$,

$$f(n) \geq c g(n)$$

then $f(n)$ is Big Ω of $g(n)$.

$$f(n) = n^2 + 10n$$

$$\textcircled{1} n^2 + 10n \geq n$$

$$n = 1$$

$$11 \geq 1$$

$$c = 1$$

$$\textcircled{2} n^2 + 10n \geq n^2$$

$$n = 2$$

$$4 + 20 \geq 4$$

$$24 \geq 4$$

$$c = 21$$

3) Big-Theta (Θ) :-

When big O and Ω meet, we ~~meet~~ indicate it by using Θ (big Θ) notation.

An algorithm is said to be $\Theta(f(n))$ if it is $O(f(n))$ and it is $\Omega(f(n))$.

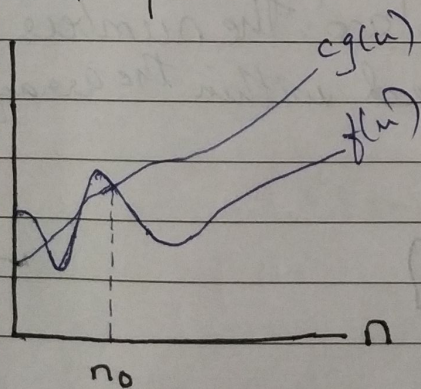
Definition:- For any non-negative functions, $f(n)$ and $g(n)$, if there exists an integer n_0 and constant $c_1, c_2 > 0$ such that for all integers $n > n_0$

$$c_1 * g(n) \leq f(n) \leq c_2 g(n)$$

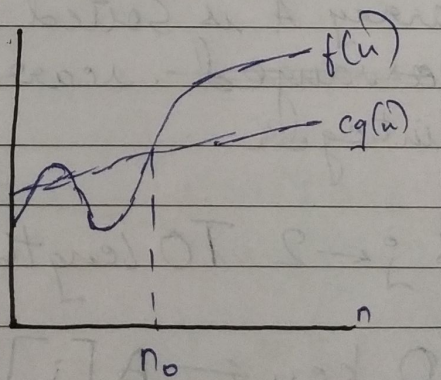
then $f(n)$ is Big Θ of $g(n)$

Ex:- $f(n) = 3n + 5$
 $g(n) = n$
 $n \leq 3n + 5 \leq 8n$
 $c_1 = 1$
 $c_2 = 8$

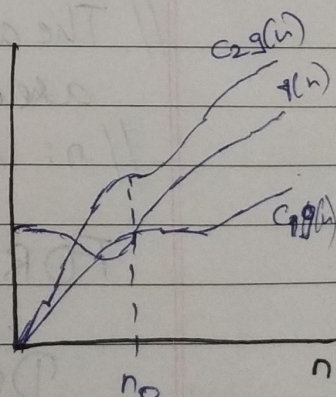
Graphical analysis ..



$$f(n) = O(g(n))$$



$$f(n) = \Omega(g(n))$$



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

Insertion Sort:-

Problem:- Sort n keys in nondecreasing order

Input:- positive integer n , array of keys of keys S indexed from 1 to n .

Output:- the array S containing the keys in increasing order.

Algorithm Insertion Sort:-

~~Insertio~~ INSERTION_SORT(A, n)

// The algorithm takes as parameter an array $A[1 \dots n]$ and length n of the array,

// The array A is sorted in place. The numbers are ~~arranged~~ rearranged within the array
// n : integer

FOR $j \leftarrow 2$ TO $\text{length}[A]$

DO $\text{key} \leftarrow A[j]$

{ ~~(n-1) - 1~~ } // Put $A[j]$ into sorted sequence $A[1 \dots j-1]$

$i \leftarrow j - 1$

While $i > 0$ and $A[i] > \text{key}$ $\leftarrow \frac{n}{2}$
{
DO $A[i+1] \leftarrow A[i]$

$$i \leftarrow i - 1$$

$$A[i+1] \leftarrow \text{key}$$

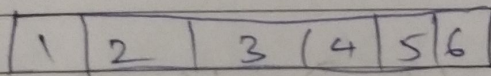
$$T(n) = n - 1 + \frac{n}{2}(n - 1)$$

$$= n - 1 + \frac{n^2 - n}{2}$$

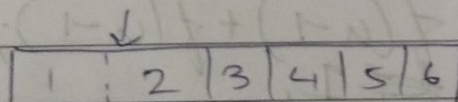
$$\frac{n^2 + n - 1}{2}$$

$$T(n) \approx O(n^2)$$

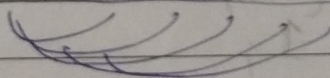
For Best Case :-



$$n = 6$$



$$5 \rightarrow n - 1$$



~~$$1 + 2 + 3 + \dots + (n - 2) + (n - 1)$$

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

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

For worst case :-
 $O(n^2)$

$$S(n) = O(n) \quad (n + 1 + \dots + 1)$$

$= n + 1$

Ranka
DATE / /
PAGE

Recursion :-

$$\begin{aligned} \text{fac}(5) &= 5 \times \text{fac}(4) \\ \text{fac}(4) &= 4 \times \text{fac}(3) \\ \text{fac}(3) &= 3 \times \text{fac}(2) \\ \text{fac}(2) &= 2 \times \text{fac}(1) \\ \text{fac}(1) &= 1 \end{aligned}$$

$$\text{fac}(n) = n * \text{fac}(n-1)$$

Fibonacci :-

$$\begin{aligned} f(n) &= f(n-1) + f(n-2); \\ f(1) &= 1; \\ f(2) &= 1; \end{aligned}$$

* Divide & Conquer Technique

→ The term 'divide and conquer'

→ It generally refers to the strategy of breaking the entity in a given population.

→ Algo

Algorithm :-

- 1) Divide the domain into Small (atomic level), where Small is the basic unit whose solution is known.
- 2) Solve individual sub-problems using the solution of SMALL.
- 3) Combine the sub-solutions to get the final answer.

* General Method :-

- 1) Algorithm ~~DA~~ DAndC(P)
 - {
 - if Small(P) then return S(P);
 - else
 - {
 - divide P into smaller instances $P_1, P_2, \dots, P_k, k \geq 2$
 - Apply D And C to each of these subproblems.
 - return Combine(D And C(P_1), D And C(P_2), ..., ~~D And C(P_k)~~, D And C(P_k));
 - }
 - }

~~Binary Search.~~

~~Algorithm P~~

Master Method

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

where $a \geq 1$, $b > 1$ & $f(n)$
is always $\geq +ve$

$f(n)$ = asymptotically $+ve$

The theorem is to find the asymptotic
bounds of $T(n)$

$n \rightarrow$ size of the problem

$a =$ no. of sub problems in
Recursion.

$\frac{n}{b} \rightarrow$ size of each sub
Problem

$f(n) =$ work done outside
recursive problem.

$$f(n) = O(n^c)$$

Case 1:

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

Case 2:-

$$c = \log_b a$$

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

Case 3:-

$$c > \log_b a$$

$$\Rightarrow T(n) = \Theta(f(n))$$

* Master Theorem is applicable only for Divide & Conquer Technique.

$$Q1) T(n) = 2T\left(\frac{n}{2}\right) + n^2$$

$$a=2, b=2, c=2$$

∴ By Master Method

$$\log_b a = \frac{\log a}{\log b} = \frac{\log 2}{\log 2} = 1$$

$$\therefore c > \log_b a \quad [\text{Case 3}]$$

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

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

$$a=8, b=2, c=2$$

By Master Method;

$$\log_b a = \frac{\log a}{\log b} = \frac{\log 8}{\log 2} = \frac{3 \log 2}{1 \log 2}$$

$$= 3 = 1.5 \times 2$$

$$c < \log_b a \quad [\text{Case 1}]$$

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

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

$$= O[n^3]$$

$$3) T(n) = 8T\left(\frac{n}{4}\right) + n^2$$

$$a=8, b=4, c=2$$

$$\log_b a = \frac{\log a}{\log b} = \frac{\log 8}{\log 4} = \frac{3 \log 2}{2 \log 2} = \frac{3}{2} = 1.5$$

$$\therefore c > \log_b a \Rightarrow [\text{Case 3}]$$

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

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

Q4) $T(n) = T\left(\frac{9}{10}n\right) + n$

$$a = 1, b = \frac{10}{9}, c = 1$$

$\therefore$ ~~By~~ By Master's Theorem,

$$\log_b a = \frac{\log a}{\log b} = \frac{\log 1}{\log \frac{10}{9}} = 0$$

$$c > \log_b a \quad [\text{case 3}]$$

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

Q5) $T(n) = 4T\left(\frac{n}{2}\right) + n^3$

$$a = 4, b = 2, c = 3$$

~~log a~~ By Master's Theorem,

$$\log_b a = \frac{\log a}{\log b} = \frac{\log 4}{\log 2} = 2$$

$$\therefore c > \log_b a \quad [\text{case (3)}]$$

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

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

$$a=4, b=2, c=1$$

By Master's Theorem,

$$\log_b a = \frac{\log a}{\log b} = 2$$

$$\therefore c < \log_b a$$

case ①

$$T(n) = O(n^{\log_b a}) = O[n^2]$$

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

$$a=2, b=2, c=1$$

By Master's Theorem,

$$\log_b a = \frac{\log a}{\log b} = \frac{\log 2}{\log 2} = 1$$

$$\therefore c = \log_b a \quad [\text{case 2}]$$

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

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

$$a=3, b=3, c=1$$

By Master's Theorem;

$$\log_b a = \frac{\log a}{\log b} = \frac{\log 3}{\log 3} = 1$$

$$\therefore c = \log_b a \quad [\text{Case 2}]$$

$$T(n) = \Theta(n^c \log n) \\ = \Theta[n \log n]$$

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

$$a=16, b=8, c=2$$

By Master's Theorem;

$$\log_b a = \frac{\log a}{\log b} = \frac{\log 16}{\log 8} = \frac{4}{3} = 1.33$$

$$\therefore c > \log_b a \quad \text{Case 3}$$

$$T(n) = \Theta[n^c] = \Theta[n^2]$$

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

$$a = 1, b = 2, c = 0$$

$$\log_b a = \frac{\log a}{\log b} = \frac{\log 1}{\log 2} = 0$$

$$\therefore c = \log_b a = 0$$

$$T(n) = O[n^c \log_b a] = O[n^0 \log n] \\ = O[\log n]$$

* BINARY SEARCH *

Algorithm Recursive Binary Search

// The algorithm takes as parameters an array $A[1..n]$, the search key and lower-higher

// index pair of the array.

// Output - The algorithm returns index of the search key in the given array, if it's present.

int binary_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 half.
int imid = midpoint(imin, imax);
// three-way comparison

IF (A[imid] > key)

// key is in ← lower subset

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

ELSE IF (A[imid] < key)

// key is in → higher subset

RETURN binary_search(A, key, imid+1, imax);

ELSE

// key has been found

RETURN imid;

}

Master Theorem for Binary Search

$$a=1, b=2, n=2$$

$$T(n) = 1 \cdot T\left(\frac{n}{2}\right) + 1 \rightarrow \text{Iteration}$$

* Min-Max *

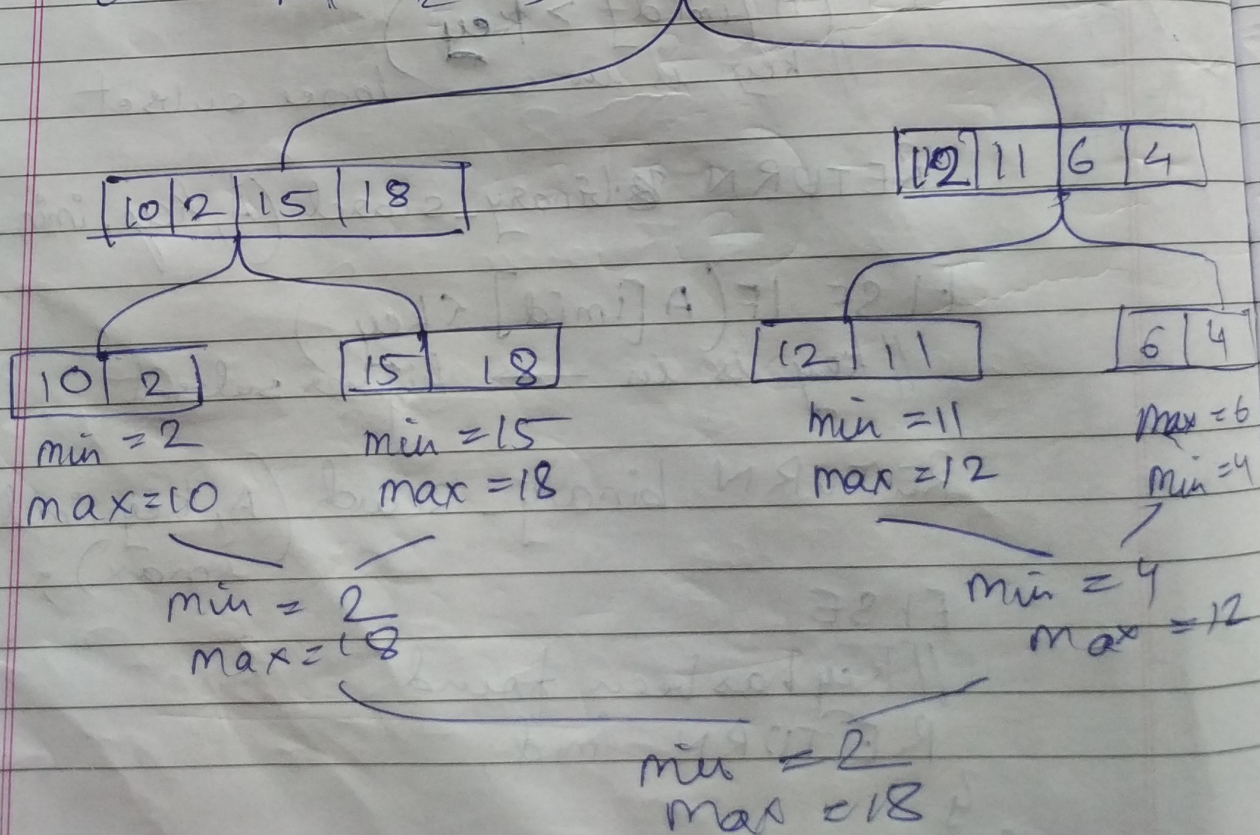
→ Is a recursive or backtracking algorithm which is used in decision making and game theory

game tree

Ex:-

10	2	15	18	12	11	5	4
----	---	----	----	----	----	---	---

1 2 3 4 5 6 7 8



Algorithm: Recursive Max-Min

VOID MaxMin(int i, int j, Type &max, Type &min,
// $A[i:n]$ is a global array. Parameters i and j are
integers, $1 \leq i \leq j \leq n$.

// The effect is to set max and min to the largest
and the smallest values in $a[i:j]$ respectively.

{

IF ($i == j$)

max = min = $a[i]$;

ELSE IF ($j == i + 1$) // for 2 elements

{

IF ($a[i] < a[j]$)

{

max = $a[j]$;

min = $a[i]$;

ELSE

{

max = $a[i]$;

min = $a[j]$;

}

ELSE

{

Type max1, min1;

// If P is not small divide P into sub
problems. Find where to split the set.

int mid = (i+j)/2;

// solve the sub problems.

Max Min(i, mid, max, min);
Max Min(mid+1, j, max1, min1);

// Combine the solutions.

IF (max < max1) max = max1;
IF (min > min1) min = min1;
}

Here a = 2

$$\frac{n}{b} = 2$$

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

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

Example on Recurrence

1) void Test(int n)
 if (n > 0)
 print();
 Test(n-1);
}

→ Substitution

$$T(n) = T(n-1) + x \rightarrow (1)$$

Delaying (1) by 1

$$T(n-1) = T(n-2) + x \rightarrow (2)$$

Putting (2) in (1)

$$T(n) = T(n-2) + 2x \rightarrow (3)$$

delaying (1) by 2

$$T(n-2) = T(n-3) + x \rightarrow (4)$$

Putting eq (4) in (3)

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

General suppose k^{th} time

$$T(n) = T(n-k) + kx \rightarrow (5)$$

$k = n$ in eq (5)

$$T(n) = T(0) + nx$$

$$= 1$$

$$\therefore O(n)$$

figure out into

Ex:- $T(n) = T(n-1) + \frac{2}{n}$ $\rightarrow$ ①

Delay eq ① by ①

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

$$T(n) = T(n-2) + \frac{2}{n} + \frac{2}{n-1} \rightarrow ③$$

Delay ① by ②

$$T(n-2) = T(n-3) + \frac{2}{n-2}$$

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

Put $k = n$

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

+ ...

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

$$n-(k-1)$$

$$k = n$$

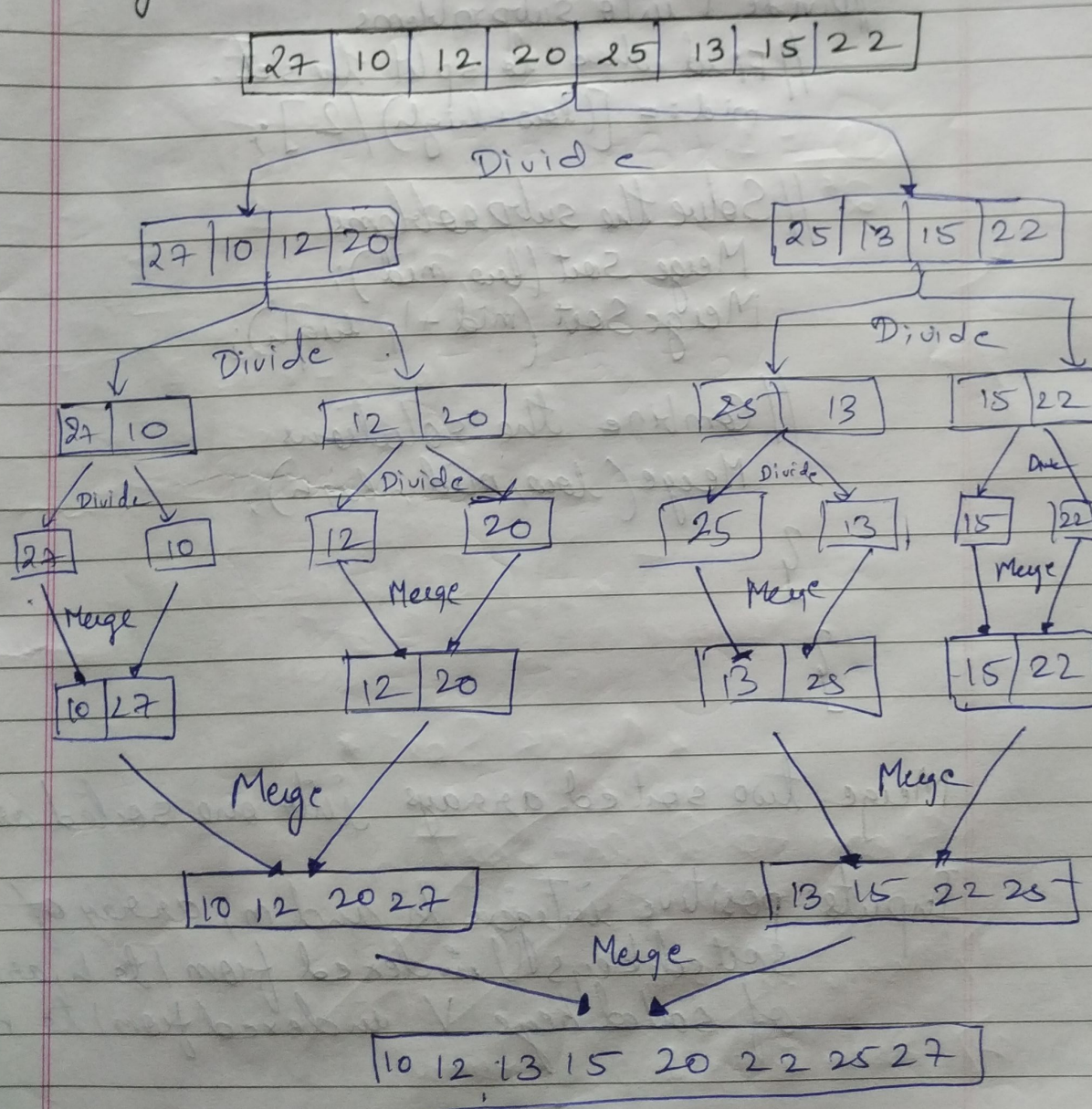
$$T(n) = T(0) + \frac{2}{n} + \frac{2}{n-1} + \frac{2}{n-2} + \dots + \frac{2}{n-(k-1)}$$

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

$$T(n) = 2 \log_2 n$$

★ MERGE SORT ★

Eg:-



Algorithm Merge Sort (low, high)

// $a[\text{low}:\text{high}]$ 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 $(\text{low} < \text{high})$ then // If there are more than one elements

{ Divide P into subproblems

// Find where to split the set.

$\text{mid} := \lfloor (\text{low} + \text{high}) / 2 \rfloor;$

// Solve the subproblems.

Merge Sort (low, mid);

Merge Sort (mid + 1, high);

// Combine the solutions

Merge (low, mid, high);

}

}

Merge two sorted arrays into one sorted array

Inputs: positive integers n and m , array of sorted keys U indexed from 1 to n , array of sorted keys V indexed from 1 to m .

Outputs: an array S indexed from 1 to $h+m$ containing the keys in U and V in a single sorted array.

void merge (int h , int m , const key type $U[]$,
const key type $V[]$,
key type $S[]$)

{
 index i, j, k .
 $i = 1; j = 1; k = 1;$

while ($i \leq h \ \&\& \ j \leq m$)

{
 $S[k] = U[i];$
 $i++;$

}

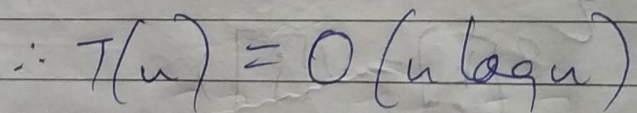
else
 {
 $S[k] = V[j];$
 $j++;$

 }
 $k++;$

}
if ($i \leq h \ \&\& \ j > m$)
 copy $V[j]$ through $V[m]$ to $S[k]$ through $S[h+m];$

else if ($j \leq m \ \&\& \ i > h$)
 copy $U[i]$ through $U[h]$ to $S[k]$ through $S[h+m];$
}

Merge sort:-



* Quick Sort *

Steps for Quick Sort

- 1) Assign the pivot P
- 2) Assume index i starts from first

Algorithm QuickSort (p, q)

// Sort the elements $a[p], \dots, a[q]$ which reside in
 // global array $a[1:n]$ into ascending order; $a[p+1]$
 // is considered to be defined and must be $\geq$ all
 // the elements in $a[j:n]$.

```

{
    if ( $p < q$ ) then // If there are more than one element
    {
        // divide P into two subproblems.
         $j := \text{Partition}(a, p, q+1);$ 
        // j is the position of the partitioning element.
        // Solve the subproblems.
        QuickSort ( $p, j-1$ );
        QuickSort ( $p, j+1, q$ );
        // There is no need for combining
        // solutions.
    }
}
    
```

Algorithm Partition (a, m, p)

// Within $a[m], a[m+1], \dots, a[p-1]$ 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 $p-1$, $a[k] \leq t$ for $m \leq k < q$ and $a[k] \geq t$
 // for $q < k < p$, q is returned, set $a[p] = \infty$.

RANK
DATE
PAGE 40

```

{
    v := a[m]; l := 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; ret
    return j;
}

```

Algorithm Interchange(a, i, j) =

// Exchange a[i] with a[j]

```

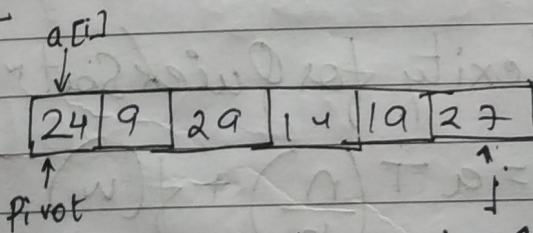
{
    p := a[i];
    a[i] := a[j];
    a[j] := p;
}

```


Steps:—

- 1) Assign the pivot P.
- 2) Assume index i starts from the first element array and j starts from last.
- 3) Compare pivot with $a[i]$ & $a[j]$.
- 4) Track i till you get $a[i] < a[j]$.
- 5) Track j till you get $a[j] > \text{pivot}$.
- 6) Perform swapping & continue tracking when you get $i > j$.
Stop. ~~swap~~ Swap the ~~element~~ element with the pivot element.
- 7) Apply D&C approach.

Ex:—

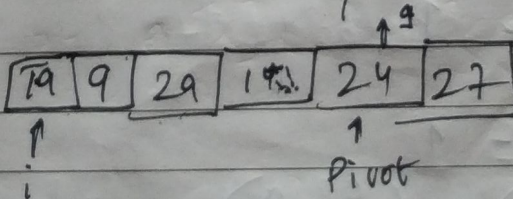


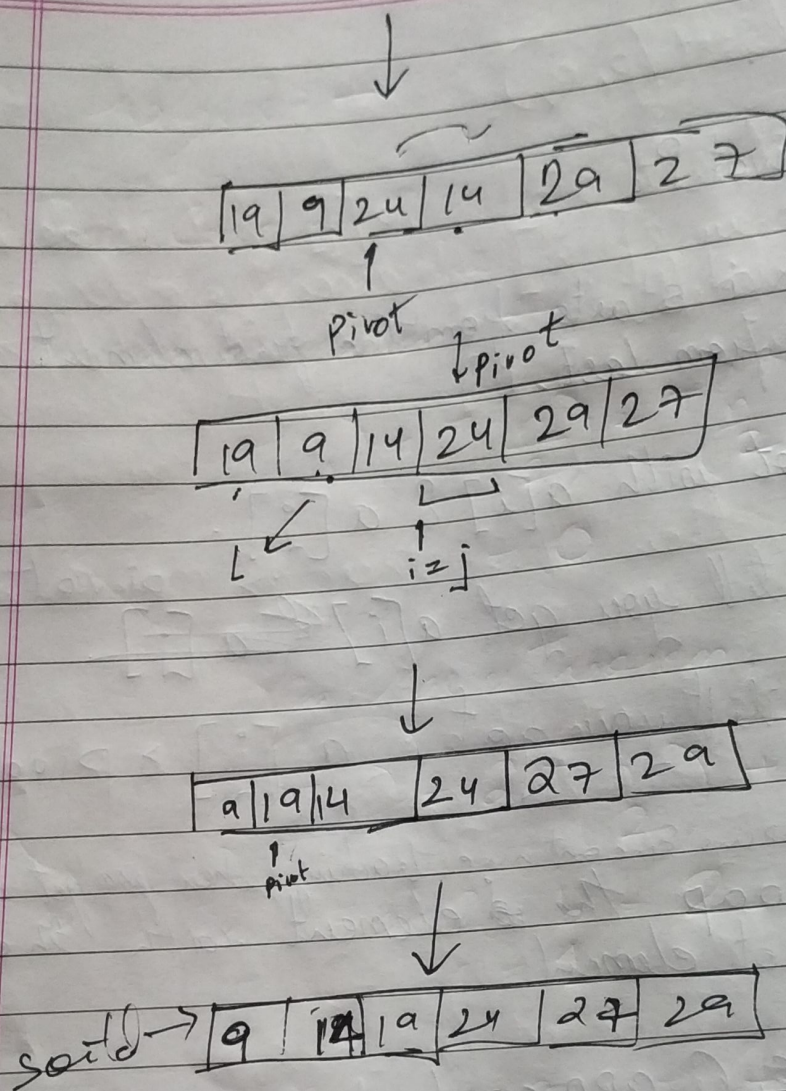
$\therefore a[\text{pivot}] < a[j]$
no swap

Now $i = 1$
 $j = 5$

$a[\text{pivot}] > a[i]$

swap





* Time complexity for QuickSort

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

$$a=2, b=2$$

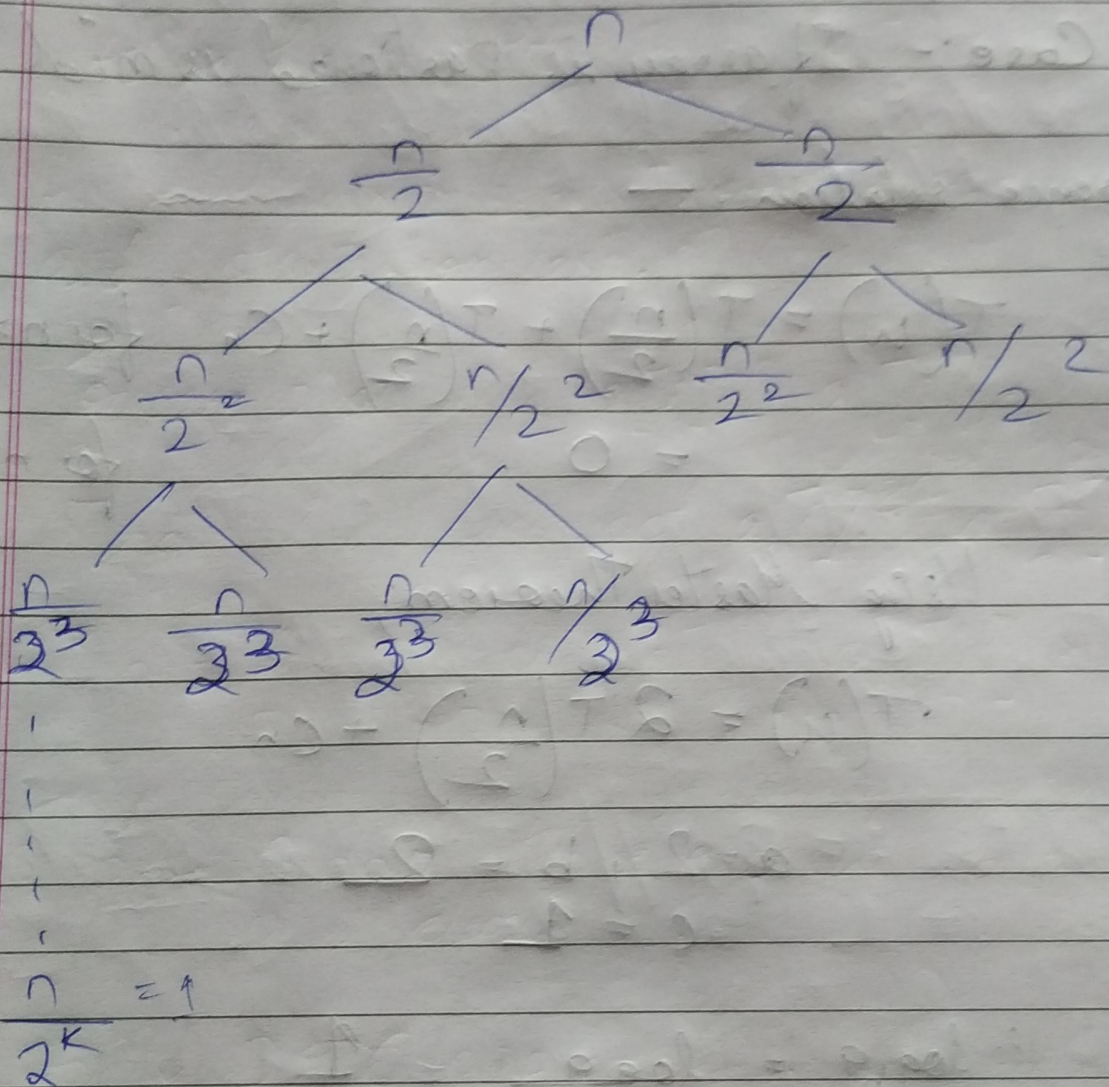
$$f(n) = n, c=1$$

$$\log_b a = 1$$

$$\therefore c = 1$$

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

By recursion tree



$$\therefore T(n) = n + \frac{n}{2} + \frac{n}{4} + \dots$$

$$= n \log n$$

Analysis :-

1) Best Case:- If array is partitioned in mid

Recurrence relation :-

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + Cn \quad \text{for } n > 1$$
$$= 0 \quad \text{for } n = 1$$

Using Master Theorem,

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

$$a = 2, b = 2$$
$$c = 1$$

$$\therefore \log_b a = \frac{\log a}{\log b} = 1$$

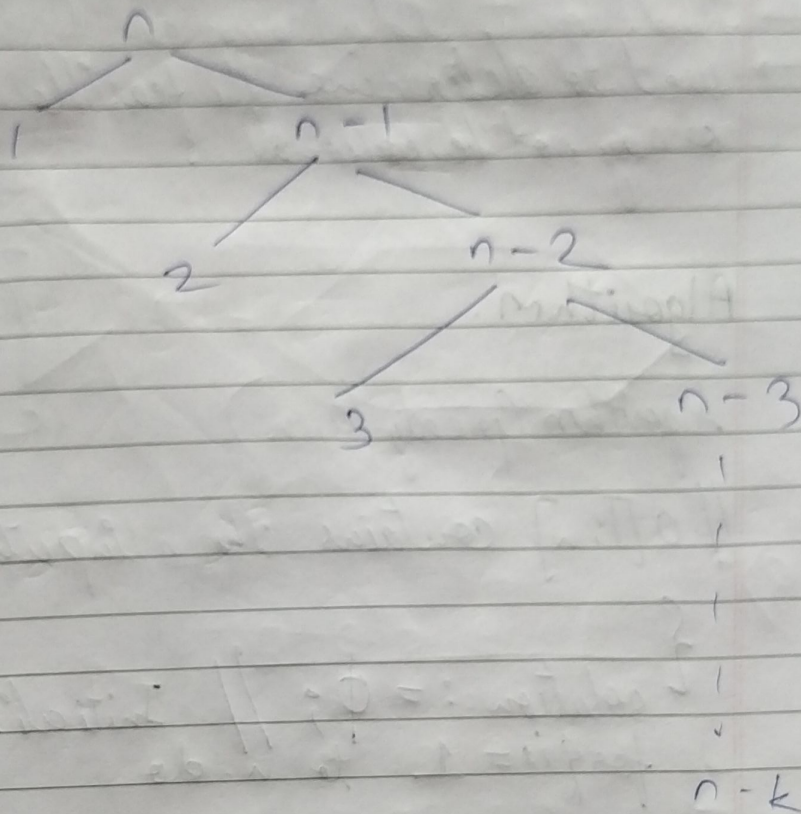
$$\therefore c = \log_b a$$

$$T(n) = O(n^1 \log n) = O(n \log n)$$

2) Worst Case :-

If pivot is a maximum or minimum of all elements in sorted list

By Recursion tree



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

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

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

By Master Method

$$a=2, b=1, c=1$$

$$f(\log_b a) = 2$$

$$\therefore T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$$

Greedy Technique

- Used for solving optimization problems
- used to determine the feasible solution that or may not be optimal.

Algorithm.

Algorithm Greedy(a, n)

// a[1:n] contains the n inputs.

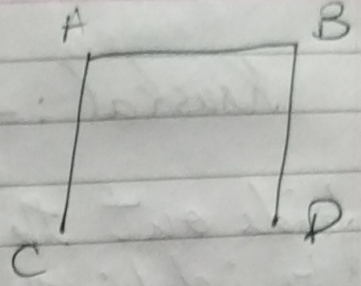
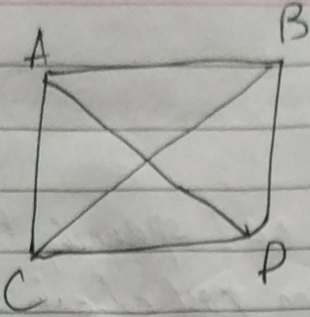
```

{
  solution :=  $\phi$ ; // Initialize the solution
  for i = 1 to n do
  {
    x := Select(a);
    if Feasible(solution, x) then
      solution := Union(solution, x);
  }
  return solution;
}

```

Spanning Tree:— A graph that covers all the vertices and doesn't contain any cycle.

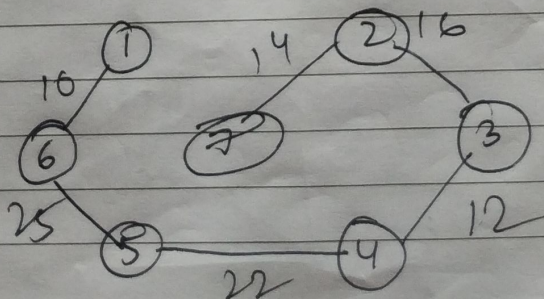
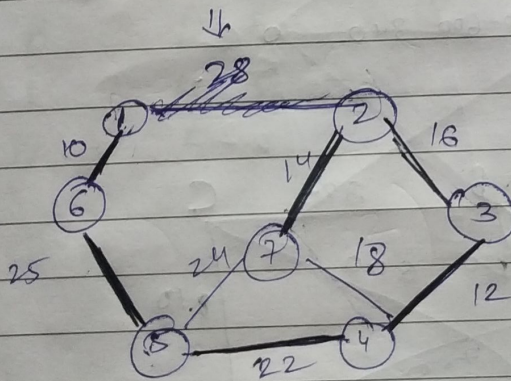
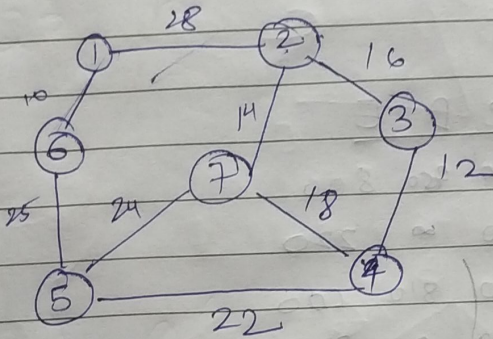
Ex:-



* PRIMS *

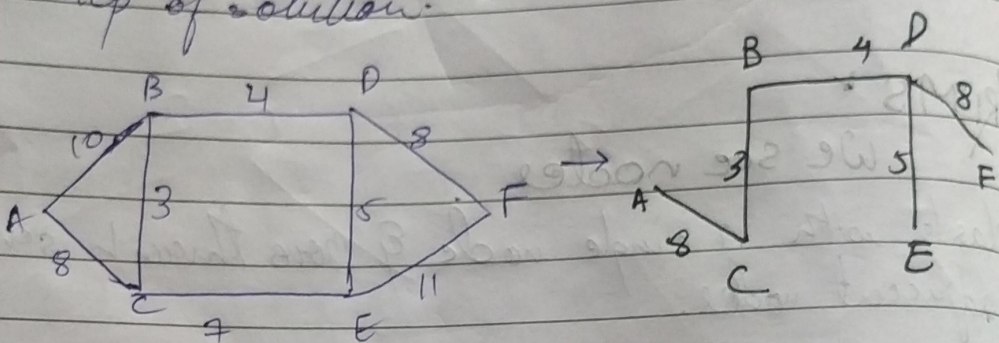
We see nodes

Eg. Starts with a single node & moves through several adjacent nodes.



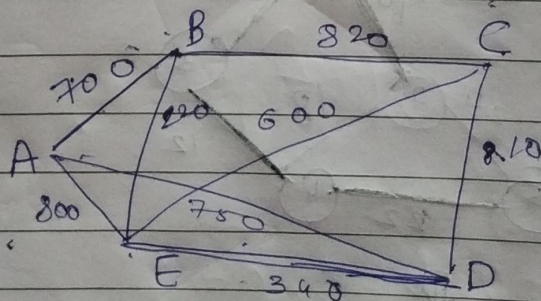
Kruskal :- edge

→ find out the minimum cost spanning tree by including the minimum cost edge in each step of solution.

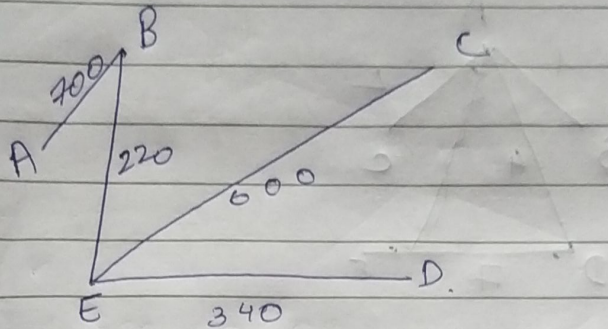


9)

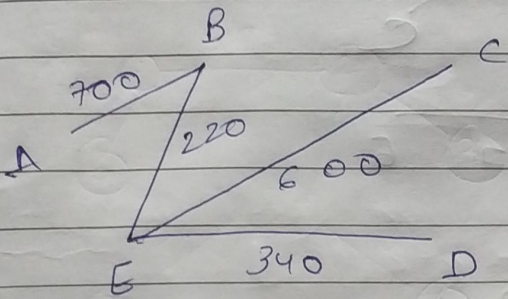
	A	B	C	D	E
A	0	700	∞	750	800
B	700	0	820	∞	220
C	∞	820	0	810	600
D	750	∞	810	0	340
E	800	220	600	340	0



Prims:-

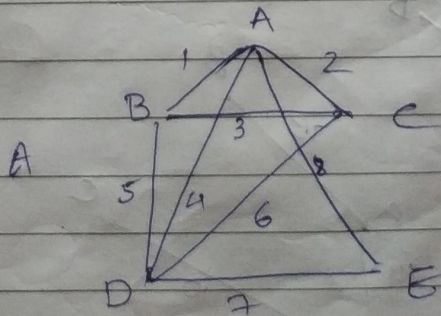


Kruskal

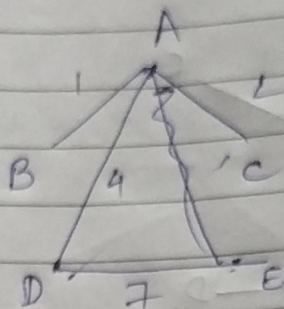


Q)

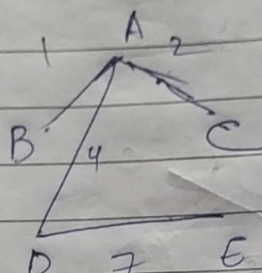
	A	B	C	D	E
A	0	1	2	4	8
B	1	0	3	5	∞
C	2	3	0	6	∞
D	4	5	6	0	7
E	8	∞	∞	7	0



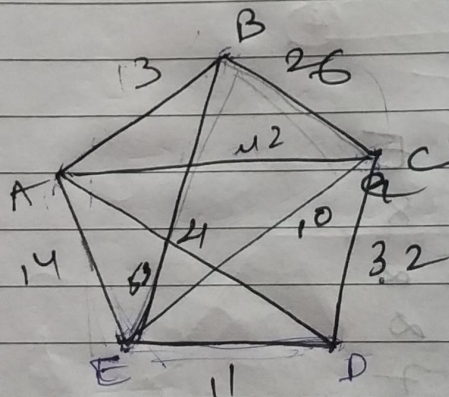
Kruskal



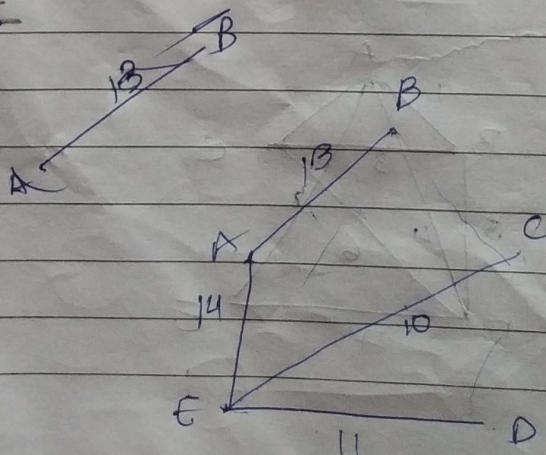
11



Numerical 3:-

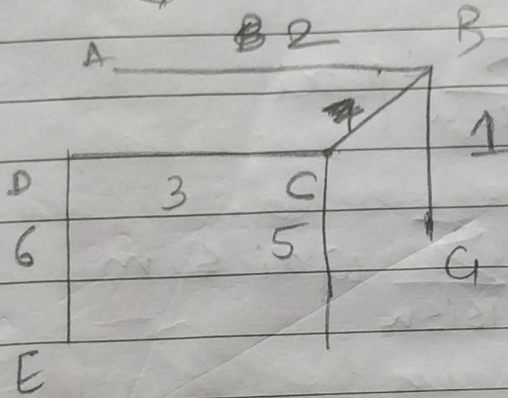
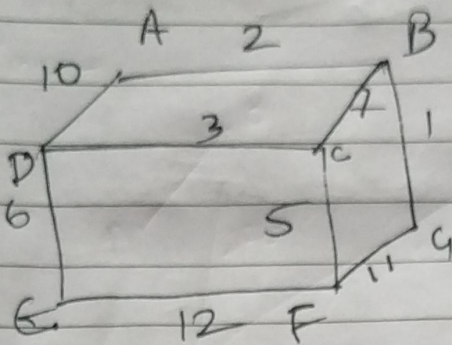


Kruskal Prims





Q4)



* Knap Sack *

$$\text{maximize } \sum_{1 \leq i \leq n} p_i x_i$$

$$\text{subject to } \sum_{1 \leq i \leq n} w_i x_i \leq m$$

$$\text{and } 0 \leq x_i \leq 1, 1 \leq i \leq n$$

w = weight

$m \rightarrow$ max capacity of the bag

x_i = object to put in the bag

Eg:- Object 1 2 3 4 5 6 7 8

P	3	6	9	4	18	9	12	10
w _i	1	5	7	4	2	8	3	6

①

P/w	3	1.2	1.2	1	9	1.12	4	1.6
w	③	⑥	⑦	④	②	⑧	③	⑥

Descending order.

20 kg

RANK
53

w_i	Taken	Balance	$x_i = T/w$	P_i	$x_i w_i$	$P_i x_i$
2	2	18	1	18	2	18
3	3	15	1	12	3	12
1	1	14	1	8	1	3
6	6	8	1	10	6	10
5	5	3	1	6	5	6
7	3	0	3/7	9	3	3.8
8	6	0	0	9	0	0
4	0	0	0	4	0	0
					20	52.8

27

27

Obj	1	2	3	4	5	6
P_i	9	4	18	12	10	9
w_i	5	5	7	2	1	1
$\frac{P_i}{w_i}$	1.8	0.8	2.6	6	10	9
	5	6	4	3	①	②

5 kg

w_i	Taken	Balance	$x_i = T/w$	P_i	$x_i w_i$	$P_i x_i$
10	1	14	1	10	1	10
9	1	13	1	9	1	9
6	2	11	1	12	2	12
26	7	4	1	18	7	18
18	4	0	5/4	9	4	9.2
0	5	0	0	4	0	0
					15 kg	56.2 kg

14.4
3.2
26

Dijkstra's algorithm

Single Source Shortest Path Algorithm

$S =$ Source Node
initially

$$N = \{S\}$$

$N =$ Set of permanently labelled nodes whose shortest path have been determined.

$D_j =$ Minimum cost b/w source node and node j .

$D_s =$ Minimum cost b/w source node and source node.

$$D_j = \min (D_i + C_{ij})$$

$D_i =$ min cost of the permanently labelled node i .

$C_{ij} =$ cost of edge ij

[shortest path algorithm from starting node to target node in weighted graph]

Dijkstra's algorithm:-

- Steps →
- 1) The ^{first} step is to mark all nodes as unvisited.
 - 2) Mark the picked starting node with a current distance of 0 and the rest nodes with infinity.
 - 3) Now, fix the starting node as the current node.
 - 4) For the current node, analyse all of its unvisited neighbours and measure their distances by adding the current distance of the current node to the weight of the edge that connects the neighbour node and current node.
 - 5) Compare the recently measured distance with the current distance of the neighbouring node and make it as the new current distance of the neighbouring node.
 - 6) After that, consider all of the unvisited neighbours of the current node, mark the current node as visited.
 - 7) If the destination node has been marked visited then stop, an algorithm has ended, and.
 - 8) Else, choose the unvisited node that is marked with the least distance, fix it as the new current node, and repeat the process again from step 4.

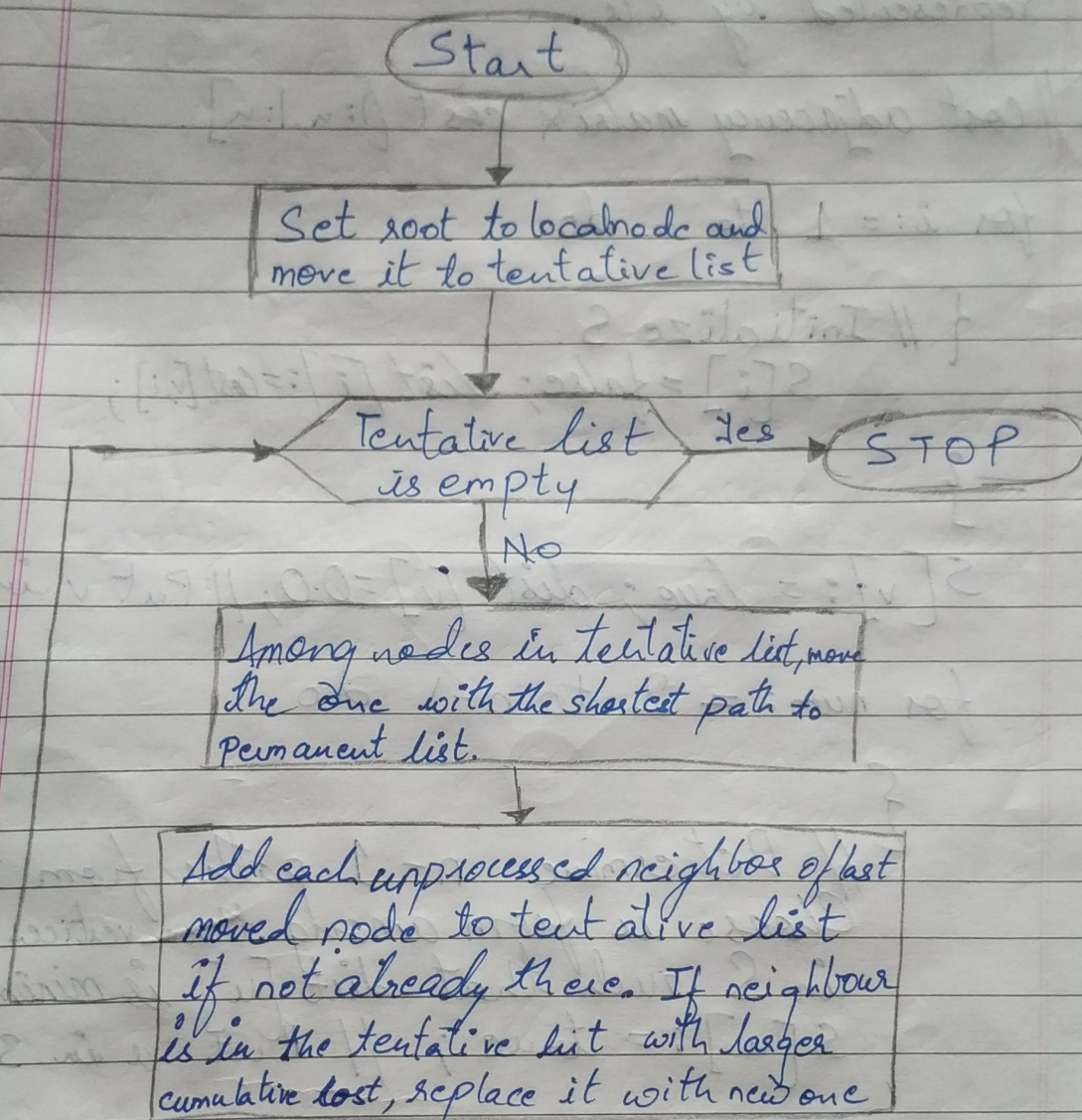
Advantages :-

- 1) ~~Disadv~~ One of the main advantages of it is its little complexity which is almost linear.
- 2) It can be used to calculate the shortest path between a single node to all other nodes and a single source node to a single destination node by stopping the algorithm once the shortest distance is achieved for the destination node.
- 3) It only works for directed, weighted graphs and all edges should have non-negative values.

* Disadvantages :-

- 1) It does an obscured exploration that consumes a lot of time while processing.
- 2) It is unable to handle negative edges.
- 3) As it heads to the acyclic graph, so can't achieve the a shortest path and
- 4) Also there is a need to maintain tracking of vertices, have been visited.

Flowchart: —



Algorithm Shortest Path Dijkstra's

Algorithm Shortest Paths (v , cost, dist, n)

// dist $[j]$, $1 \leq j \leq n$, is set to the length of the shortest.

// path from vertex v to vertex j in a diagraph G is i with n .

// vertices. $\text{dist}[v]$ is set to zero. Q is represented by i to.

// cost adjacency matrix $\text{cost}[1:n, 1:n]$.

for $i := 1$ to n do

{ // Initialize S .
 $S[i] := \text{false}$; $\text{dist}[i] := \text{cost}[v, i]$;

}

$S[v] := \text{true}$; $\text{dist}[v] = 0.0$; // Put v in S

for $\text{num} := 2$ to $n-1$ do

{ // Determine $n-1$ paths from v .
Choose u from among those vertices not in S such that $\text{dist}[u]$ is minimum;
 $S[u] := \text{true}$; // Put u in S .

for (each w adjacent to u with $S[w] = \text{false}$) do

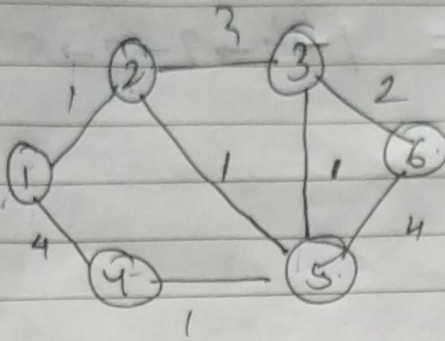
// Update distances

if ($\text{dist}[w] > \text{dist}[u] + \text{cost}[u, w]$) then
 $\text{dist}[w] := \text{dist}[u] + \text{cost}[u, w]$

}

}

Ex 1:-



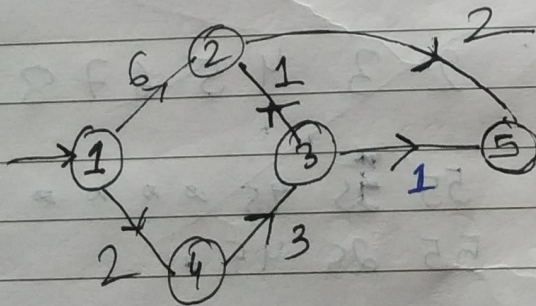
Source node = 1

	Source	Vselect	2	3	4	5	6
①	1	-	1	∞	4	∞	∞
②	1	2	1	4	4	∞	∞
③	(1, 2)	5	1	4	4	2	6
	(1, 2, 5)	4	1	4	4	2	6
	(1, 2, 5, 4)	3	1	4	4	2	6
	(1, 2, 5, 4, 3)	6	1	2	4	2	6

Shortest path = 1, 2, 5, 4, 3, 6

1 → 2 → 5 → 4 → 3 → 6

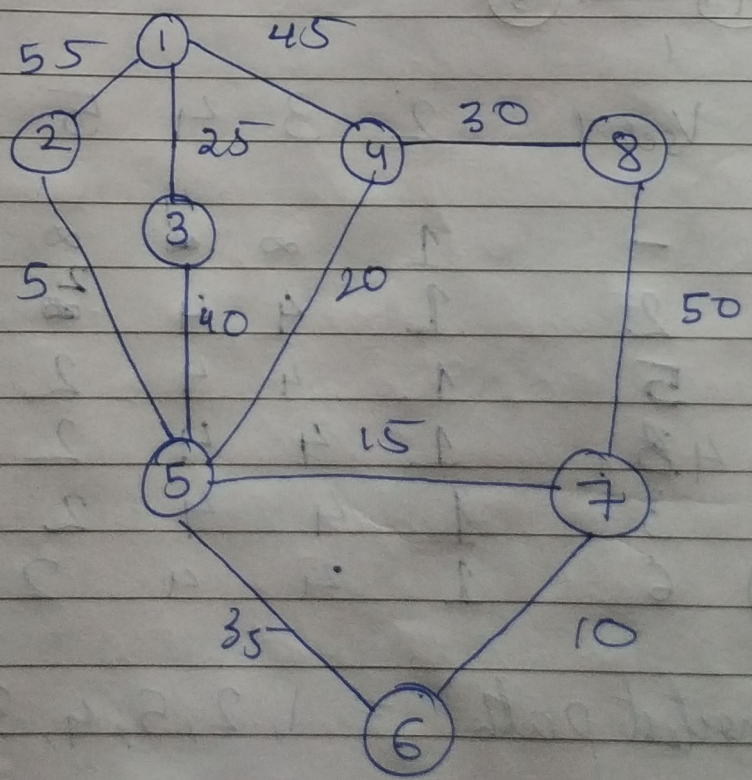
Eg 2:-



Source	Vselect	2	3	4	5
1	-	6	∞	2	∞
①	4	6	5	2	∞
(1, 4)	3	6	5	2	6
(1, 4, 3)	2	6	5	2	6
(1, 4, 3, 2)	5	6	5	2	6

shortest path = 1 → 4 → 3 → 2 → 5

Q2)



Source node = 1

Applications:-

- ① Social Network Page Graph
- ② Biological Network Graph
- ③ Neural Network Graph
- ④ Google map Graph
- ⑤ Computer network Graph