GURU NANAK DEV
ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

DESIGN AND ANALYSIS OF ALGORITHMS

Sixth Semester

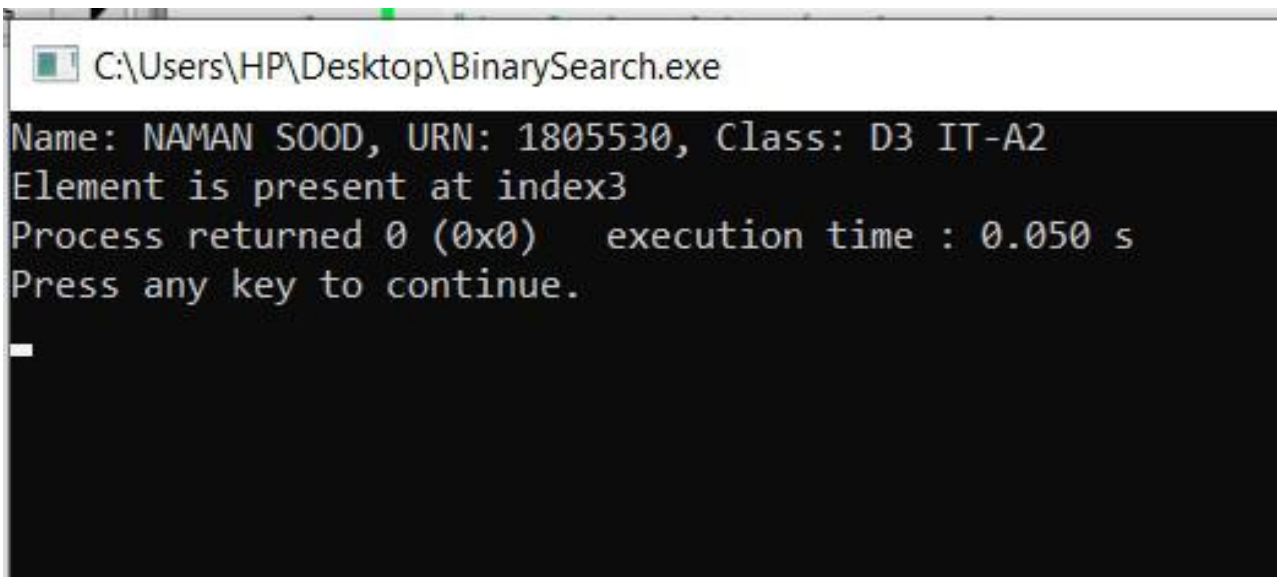
Session:- Jan-July 2021

Batch:- 2018-2022

SUBMITTED TO:-
ER. PARMINDER KAUR WADHWA
DEPARTMENT OF IT
ASSISTANT PROFESSOR

SUBMITTED BY:-
NAMAN SOOD
D3 IT-A2
1805530

EXPERIMENT NO. :- 1



EXPERIMENT NO. :- 1

Implement Binary Search Algorithm and compute its time complexity.

Binary Search :- Search a Sorted array by repeated - ly Dividing the search internal in half. Begin with an internal covering the whole array. If the value of the search key is less than the item in the middle of the interval to the lower half. otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Program :-

```
#include <bits/stdc++.h>
using namespace std;

int binarySearch( int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l)/2;
        if (arr[mid] == x)
            return mid;
```

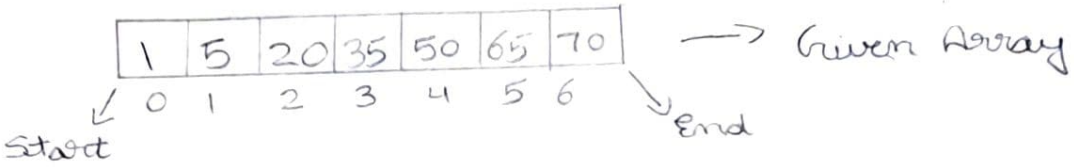
```
if (arr[mid] > x)
    return binarySearch(arr, l, mid-1, x);
return binarySearch(arr, mid+1, r, x);
}
return -1;
}
int main(void)
{
cout << "NAME - NAMAN SOOD  Class - D3 IT - A2
        URN - 1805530";
cout << endl;
int arr[] = {2,3,4,10,40};
int x = 10;
int n = sizeof(arr) / sizeof(arr[0]);
int result = binarySearch(arr, 0, n-1, x);
// using ternary operator for decission making.

(result == -1)  cout << "element is not present
        in array" : cout << "element is
        Present in index " << result;
    return 0;
}
```

Algorithm :-

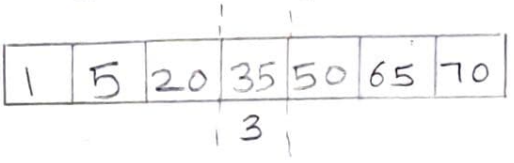
```
int BinSrch(Type a[], int i, int l, Type x)
// Given an array a[i:l] of elements in nondec-
-reasing order, 1 <= i <= l, determine whether
```

Binary Search



→ Given Array

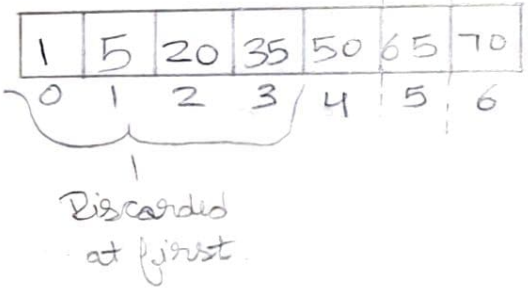
Binary search for 50 in 7 Elements Array.



$mid = \frac{0+6}{2} = 3$

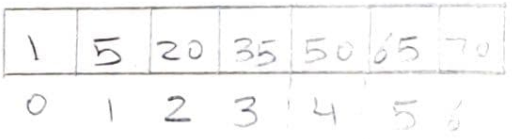
$35 < 50$

∴ Take 2nd half



$mid = \frac{4+6}{2} = \frac{10}{2} = 5$

(65 > 50) here going in left side.



50 found at 4th position

x is present, and if so, return j such that
x = = a[j];
Else return 0;
{
if (l = = i)
{
if (x = = a[i])
return i;
Else
return 0;
}
Else {
int mid = (i+l)/2;
if (x = = a[mid])
return mid;
Else if (x < a[mid])
return BinSrch (a, i, mid-1, x);
Else
return BinSrch (a, mid+1, l, x);
}
}

Analysis of Algorithm and Time Complexity :-

1. Let say the iteration in Binary Search terminates after K iterations. let say it gets terminated after 3 iterations. K = 3.

2. At each iteration the array is divided by half so let is take length of array at any iteration is n.

1 At iteration 1
Length of array = n.

2 At iteration 2.
length of array = n/2.

3) At iteration 3.
Length of array = (n/2)/2 = n/2²

. after K^th iteration
length of array = n/2^K

→ After K^th division, the length of array becomes 1

∴ length of array = n/2^K = 1

$$\Rightarrow n = 2^K$$

$$\Rightarrow \log_2(n) = \log_2(2^K)$$

$$\Rightarrow \log_2(n) = K \log_2(2)$$

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

∴ time complexity of Binary Search = O(log₂(n)).

GURU NANAK DEV
ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

DESIGN AND ANALYSIS OF ALGORITHMS

Sixth Semester

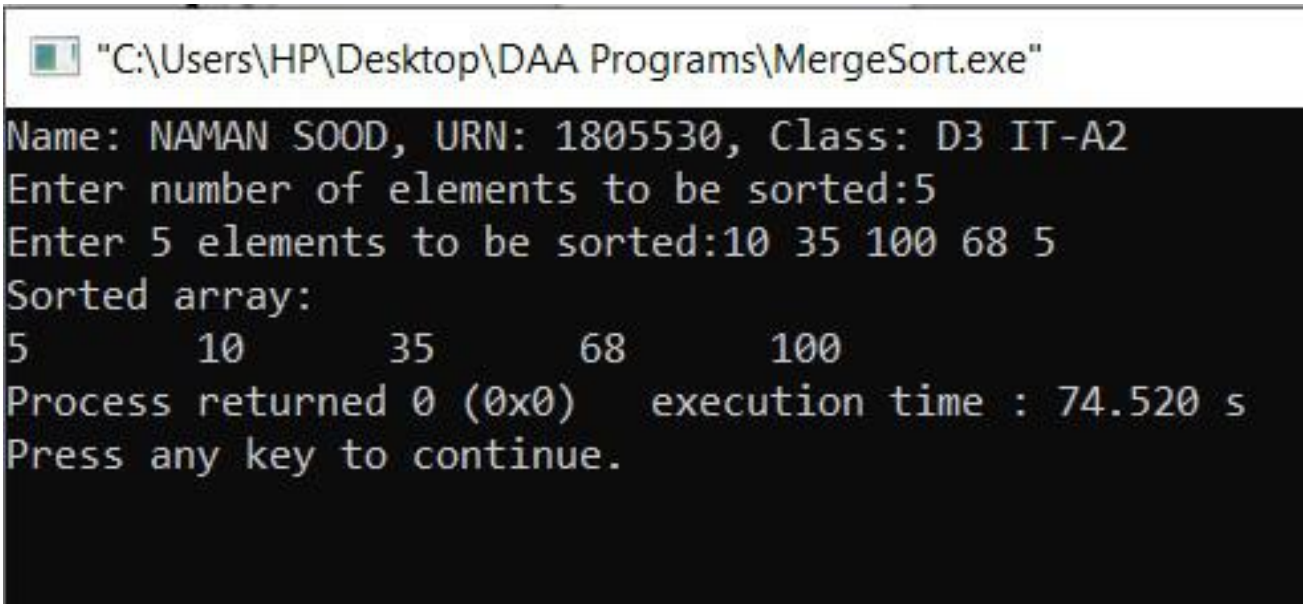
Session:- Jan-July 2021

Batch:- 2018-2022

SUBMITTED TO:-
ER. PARMINDER KAUR WADHWA
DEPARTMENT OF IT
ASSISTANT PROFESSOR

SUBMITTED BY:-
NAMAN SOOD
D3 IT-A2
1805530

EXPERIMENT NO. :- 2



EXPERIMENT NO. :- 2

Implement Merge Sort Algorithm and Demonstrate the Divide and Conquer - Technique.

Merge Sort :- Merge Sort Algorithm uses the "Divide" and Conquer Strategy where in we divide the Problem into subproblems and Solve subproblems individually.

Steps to Perform :-

1.) The list to be Sorted is divided into two arrays of Equal length by dividing the list on the middle Element, if the no. of Elements in the list is either 0 or 1 then list is considered Sorted.

2.) Each Sublist is Sorted individually by using merge sort recursively.

3) The Sorted Sublists are then Combined or merged together to form a Complete Sorted list.

Program :-

```
#include <iostream>
using namespace std;
```

```
Void merge ( int , int , int );
Void merge_sort (int* arr, int low, int high)
{
  int mid ;
  if ( low < high ) {
  mid = ( low + high ) /2;
  merge - sort ( arr, low, mid );
  merge - sort ( arr, mid + 1, high );
  merge ( arr , low, high, mid );
  }
}
Void merge ( int * arr , int low, int high, int mid)
{
  int i , j , k, c[50];
  i = low;
  k = low;
  j = mid + 1
  while ( i <= mid && j <= high)
  {
  if ( arr [i] < arr [j] ) {
        c(k) = arr [i];
        k++;
        i++; }
      Else {
    c [k] = arr [j]
        k++;
        j++
  }
```

```
}
while (i <= mid)
{
    c[k] = arr[i]
    k++;
    i++;
}
while (j <= high) {
    c[k] = arr[j];
    k++;
    j++;
}
for [i = low ; i < k ; i++)
{
    arr [i] = c[i]
}
}
int main ()
{
cout << "Name - NAMAN SOOD URN-1805530
CLASS - D3IT-A2";
cout << endl;
int myarray [30], num;
cout << "enter number of elements to be sorted:";
cin >> num;
cout << "Enter" << num << "elements to be
sorted.";
```

```
for (int i = 0; i < num ; i++)
  { cin >> my array [i];
  }
merge_sort (my array, 0, num-1);
cout << "Sorted array \n";
for (int i = 0 ; i < num ; i++)
{
   cout << my array [i] << " \t";
}
```

Algorithm :-

```
Void merge_sort ( int low, int high ) {
// a [low : high ] is a global array to be sorted
// Small (p) is true if there is only one element to
Sort
   if ( low = = high ) then.
-> return high ;
   if ( low < high ) {
      int mid = ( low + high ) / 2 ;
      Merge Sort ( low , mid );
     Merge Sort ( mid + 1 , high ); }
     Else if ( low > high ) then
          int mid = ( low + high ) / 2 ;
        Merge Sort ( low, mid );
        Merge Sort ( low, mid + 1 );
// combination for both - calling merge function.
  Else then
     Merge ( low , mid, high ) }
```

Analysis of Merge Sort :-

Computing time for Merge Sort :

If the time for the merging operation is proporti-
-onal to n, then

the computing time for merge sort is :-

$$T(n) = \begin{cases} a & , n = 1 \text{ [a] constant} \\ 2T(n/2) + cn & , n > 1 \text{ [c] constant} \end{cases}$$

In merge sort when $n = 1$ then it the same elem-
-ent to be present as only one element is present but, if $n > 1$, then elements need to be sorted and array of n is divided into $n/2$ array and for there more by implementing Divide and conquer.

GURU NANAK DEV
ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

DESIGN AND ANALYSIS OF ALGORITHMS

Sixth Semester

Session:- Jan-July 2021

Batch:- 2018-2022

SUBMITTED TO:-
ER. PARMINDER KAUR WADHWA
DEPARTMENT OF IT
ASSISTANT PROFESSOR

SUBMITTED BY:-
NAMAN SOOD
D3 IT-A2
1805530

EXPERIMENT No. :- 3

Program :-

```
#include <bits/stdc++.h>
using namespace std;
void swap(int *a, int *b)
{
int t = *a;
*a = *b;
*b = t;
}
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], &arr[j]);
}
}
swap(&arr[i+1], &arr[high]);
return (i+1);
}
```

EXPERIMENT NO. : 3

Analyze the time Complexity of Quick Sort Algorithm.

Quicksort :- Quicksort is a Divide and Conquer Algorithm. it picks an Element a pivot and partitions the given array around the Picked Pivot. The key process in quicksort is partition(). Target of partitions is, given an array and an Element x of array as Pivot, Put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and Put all greater Elements (greater than x) after x. All this should be done in linear time.

Algorithm :-

```
Quicksort (A, P, r)
if P < r
then q ← Partition (A, P, r)
Quicksort (A, P, q)
Quicksort (A, q+1, r)
Partition (A, P, r)
    x ← A[P]
    dn ← P
    up ← r
```

```
Void quicksort( int arr[], int low, int high )
{
 if ( low < high )
 {
  int Pi = Partition ( arr, low, high );
  quicksort ( arr, low, Pi-1 );
  quicksort ( arr, Pi +1, high );
  }
}
Void Print array ( int arr [], int size )
{
  int i;
  for (i=0 ; i < size ; i++)
    cout << arr [i] <<" ";
    cout << end l;
 }
int main ()
{
cout <<" Name : NAMAN SOOD ; URN:- 1805530, class:-
 D3IT-A2 ";
cout << end l;
int arr [] = {100, 70, 10, 90, 60, 20};
int n = size of (arr) / size of(arr [0]);
cout <<" input array is : " << end l;
Print array (arr, n);
cout << end l;
quicksort ( arr, 0, n-1);
 cout <<" Sorted array is: " << end l;
```

While (dn ← up)
While (A[dn] <= x)
dn ← dn+1
While [A[up] > x)
up ← up-1
if (dn < up) then.
Exchange A[dn] ← → A[up]
Else return up.

Analysis of Quicksort :-

If First element divides array into two halves

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

- first Recursive case of quicksort
- second Recursive case of quicksort
- Partitioning involves comparison of First with each element.
  ∴ Time ∝ n
  Time = bn
  b = any constant

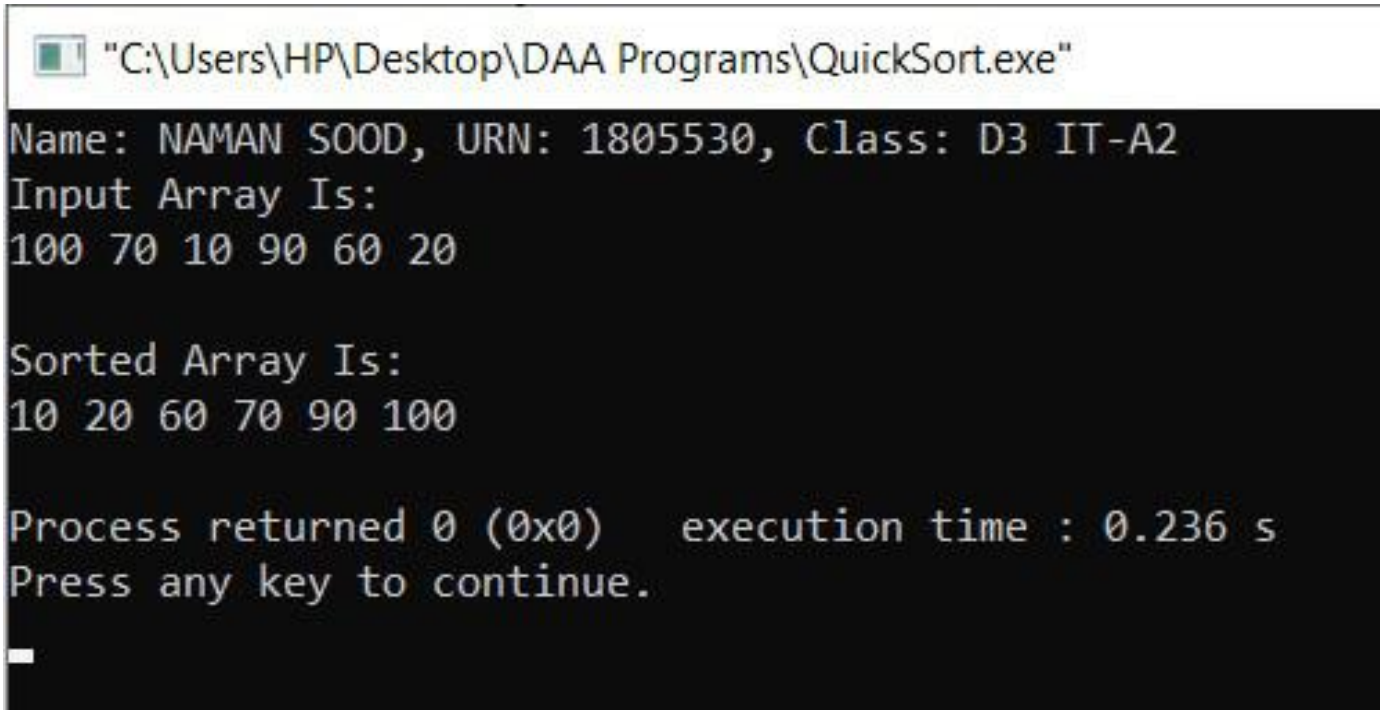
$T(n) = 2T(n/2) + bn$
$= 2(2T(n/4) + bn/2] + bn$
$= 4T(n/4) + 2\frac{bn}{2} + bn$

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

let $2^k = n$, $K = 2$
$2^k = n$

```
printArray(arr, n);
return 0;
}
```

OUTPUT



Taking log on both sides

$$\log_2 2^k = \log_2 n$$

$$k \log_2 2 = \log_2 n$$

$$\boxed{k = \log_2 n}$$

$$T(n) = 2^k T(1) + k bn$$

$$= 2^k a + bn.$$

a = Time for one element any constant value.

$$T(n) = a 2^k + bn \log_2 n$$

$$T(n) = a 2^k + bn \log_2 n$$

$$\boxed{T(n) = an + bn \log_2 n}$$

Time complexity

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

GURU NANAK DEV
ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

DESIGN AND ANALYSIS OF ALGORITHMS

Sixth Semester

Session:- Jan-July 2021

Batch:- 2018-2022

SUBMITTED TO:-
ER. PARMINDER KAUR WADHWA
DEPARTMENT OF IT
ASSISTANT PROFESSOR

SUBMITTED BY:-
NAMAN SOOD
D3 IT-A2
1805530

EXPERIMENT NO. :- 4(a)

Program :-

```
# include <bits/stdc++.h>
using namespace std;
# define V5
int minKey (int Key[], bool mstSet[])
{
 int min = INT_MAX, min_index;
 for (int v = 0; v < V; v++)
 if (mstSet[v] == false && Key[v] < min)
     min = Key[v], min_index = v;
 return min_index;
}
 void PrintMST (int Parent[], int graph[V][V])
{
  cout << "Edge \t Weight \n";
  for (int i = 1; i < V; i++)
  cout << Parent[i] << "- " << i << "\t" << graph
  [i][Parent[i]] << "\n";
}
 Void PrimMST (int graph[V][V])
{
 int Parent[V];
 int Key[V];
 bool mstSet[V];
 for [int i = 0; i < V; i++)
```

EXPERIMENT NO. :- 4(a)

Solve minimum cost spanning tree problem using greedy method. [Prims Algorithm]

Prims Algorithm :- Prim's Algorithm is a Greedy Algorithm. It starts with an Empty spanning tree. The idea is to maintain two sets of vertices. the first set contains the vertices also included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connects the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST. A group of edges that connects two sets of vertices in a graph is called cut in graph theory., So at every step of prims algorithm, we find a cut (of two sets,) Pick the minimum weight edge from the cut and include this vertex to MST Set.

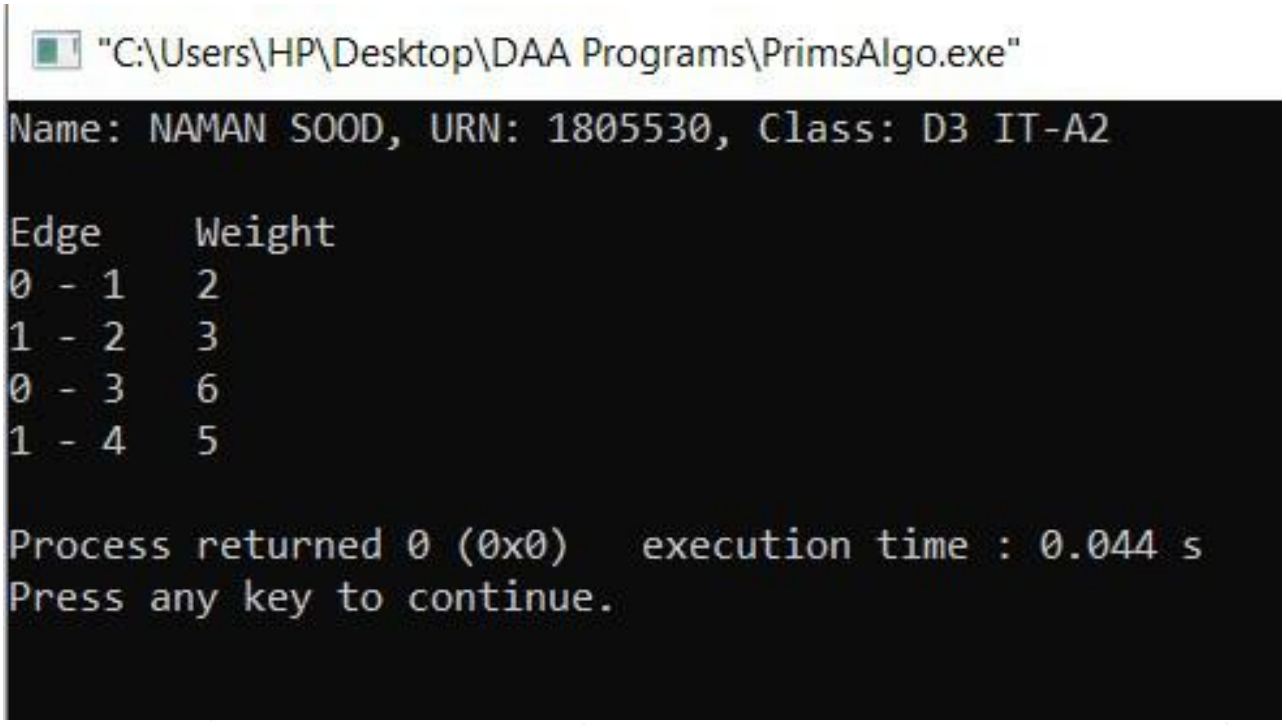
Prim's Algorithm :-

```
float Prim (int E[][SIZE], float cost[][SIZE],
        int n, int t[][2]),
```

```
Key[i] = INT_MAX, mstset[i] = false;
Key[0] = 0;
Parent[0] = -1;
for(int count = 0; count < V-1; count++)
{
int u = min Key(Key, mstset);
mstset[u] = true;
for(int v = 0; v < V; v++)
if (graph[u][v] && mstset[v] == false && graph
[u][v] < Key[v])
Parent[v] = u, Key[v] = graph[u][v];
}
Print MST(Parent, graph);
}
Print MST(Parent, graph);
}
int main()
{
Cout << "Name : NAMAN SOOD, URN: 1305530, Class:
D3 IT- A2 \n";
Cout << endl;
int graph[V][V] = {{0,2,0,6,0}, {2,0,3,8,5}
{0,3,0,0,7}, {6,8,0,0,9}
{0,5,7,9,0}};
Prim MST(graph);
return 0;
}
```

```
// E is the set of edges in G, cost[1:n][1:n] is the
// cost of adjacency matrix of an n vertex graph such
// as the cost[i][j] is either a positive real no.
// or infinity if no edge (i,j) exists. A minimum
// spanning tree is computed and stored as a set
// of edges in the array t[1:n-1][1:2]. (t[i][1],
// t[i][2]) is an edge in minimum cost spanning
// tree. the final cost is returned.
{
    int near[SIZE], j, k, l;
    let (k, l) be an edge of minimum cost in E;
    float mincost = cost[k][l];
    t[1][1] = k; t[1][2] = l;
    for (int i=1; i<=n; i++) // initialize near
     if (cost[i][1] < cost[i][k]) near[i] = 1;
      else near[i] = k;
   near[k] = near[1] = 0;
   for (i=2; i<=n-1; i++) {
    // find n-2 additional edges for t.
let j be an index such that near[j] != 0 and
 cost[j][near[j]] is minimum;
 t[i][1] = j; t[i][2] = near[j]
mincost = mincost + cost[j][near[j]];
near[j] = 0;
 for (k=1; k<=n; k++) // update near[]
     if ((near[k] != 0) && (cost[k][near[k]] > cost[k][j]))
      near[k] = j; }
   Return (mincost); }
```

OUTPUT


"C:\Users\HP\Desktop\DAA Programs\PrimsAlgo.exe"
Name: NAMAN SOOD, URN: 1805530, Class: D3 IT-A2
Edge Weight
0 - 1 2
1 - 2 3
0 - 3 6
1 - 4 5
Process returned 0 (0x0) execution time : 0.044 s
Press any key to continue.

Analysis of Time complexity of Prims Algorithm :-

1) If adjacency list is used to represent the graph, then using breadth first search, all the vertices can be traversed in $O(V+E)$ time.

2) We traverse all the vertices of graph using breadth first search and use a min heap for storing the vertices not yet included in MST.

3) To get minimum weight edge, we use min heap as a priority queue.

3) Min heap operations like extracting minimum element and decreasing key values takes $O(\log V)$ time.

So, overall time complexity.

$$= O(E+V) \times O(\log V)$$
$$= O((E+V)\log V)$$
$$= O(E \log V)$$

This time complexity can be improved and reduced to $O(E + V \log V)$ using Fibonacci heap.

GURU NANAK DEV
ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

DESIGN AND ANALYSIS OF ALGORITHMS

Sixth Semester

Session:- Jan-July 2021

Batch:- 2018-2022

SUBMITTED TO:-
ER. PARMINDER KAUR WADHWA
DEPARTMENT OF IT
ASSISTANT PROFESSOR

SUBMITTED BY:-
NAMAN SOOD
D3 IT-A2
1805530

EXPERIMENT NO. :- 4(b)

Program :-

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> ipair;
struct graph
{
  int V, E;
  vector< pair< int , ipair >> edges;
  Graph ( int V, int E )
   {
    this -> V = V;
    this -> E = E;
 }
 void add edge ( int u, int v, int w )
{
   edges. push - back ({w,{u, v}});
 }
 int KruskalMST ();
};
 struct Disjoint Sets
{
  int * Parent, * rank;
   int n;
  Disjoint Sets ( int n )
  {
```

EXPERIMENT NO. :- 4(b)

Solve Minimum cost spanning tree problem using greedy method. [Kruskal's Algorithm]

Kruskal's Algorithm :- Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected and undirected graph is spanning tree with weight less than or equal to weight of every other spanning tree. The weight of spanning tree is the sum of weights given to each edge of the spanning tree.

Kruskal's Algorithm :-

```
Float Kruskal (Int E[][SIZE], float cost[][SIZE],
               int n, int t[][2])
// E is the set of edges in G. G has n vertices
// cost[u][v] is the cost of edge (u,v). t is the
// set of edges in the minimum-cost spanning tree.
// The final cost is returned
{
  int parent[SIZE];
```

```
this -> n = n;
Parent = new int [n+1];
rank = new int [n+1];
for (int i = 0; i <= n; i++)
{
  rank [i] = 0;
   Parent [i] = i;
  }
}

  int find (int u)
{
  if ( u != Parent [u])
  Parent [u] = find ( Parent[u] );
  return Parent [u] ;
}

  void merge ( int x, int y)
{
  x = find (x) , y = find (y)
  if ( rank [x] > rank [y] )
        Parent [y] = x;
   Else
        Parent [x] = y ;
  if (rank  [x] = = rank [y] )
            rank [y] + + ;
   }
};

   int Graph :: Kruskal MST ()
{
```

```
Construct a heap out of the edge costs using heapify;
for (int i=1; i<=n; i++) parent[i] = -1;
// Each vertex is in a different set.
i = 0; float mincost = 0.0;
while ((i<n-1) && (heap not empty)) {
    delete a minimum cost edge (u,v) from edge
    heap and reheapify using adjust;
    int j = find(u); int k = find(v);
    if (j!=k) {
    i++;
    t[i][1] = u; t[i][2] = v;
    mincost += cost[u][v];
    union (j, k);
    }
}
if (i!=n-1) cout << "no spanning tree" << endl;
  else return (mincost);
}
```

Kruskal's Algorithm Time Complexity :-

Worst case time complexity of Kruskal's Algorithm

$= O(E \log V)$ or $O(E \log E)$

```
int mst_wt = 0;
sort (edges.begin(), edges.end());
Disjoint sets ds(V);
vector< pair<int, ipair>>:: iterator it;
for (it = edges.begin(); it != edges.end(); it++)
{
  int u = it->second.first;
  int v = it->second.second;
  int set_u = ds.find(u);
  int set_v = ds.find(v);
  if (set_u != set_v)
  {
    cout << u << "_" << v << endl;
    mst_wt += it->first;
    ds.merge (set_u, set_v);
  }
}
  return mst_wt;
}
int main()
{
cout << "Name:- NAMAN SOOD", URN:- 205530, class:
        D3 IT-A2 |n";
cout << endl;
int V = 9, E = 14;
graph g(V, E)
g.addEdge(0, 1, 4);
g.addEdge(0, 7, 8);
g.addEdge(1, 2, 8);
```

Analysis :-

1.) The edges are maintained as min heap.

2.) The next edge can be obtained in O(log E) time if graph has E edges.

3.) Reconstruction of heap takes O(E) time.

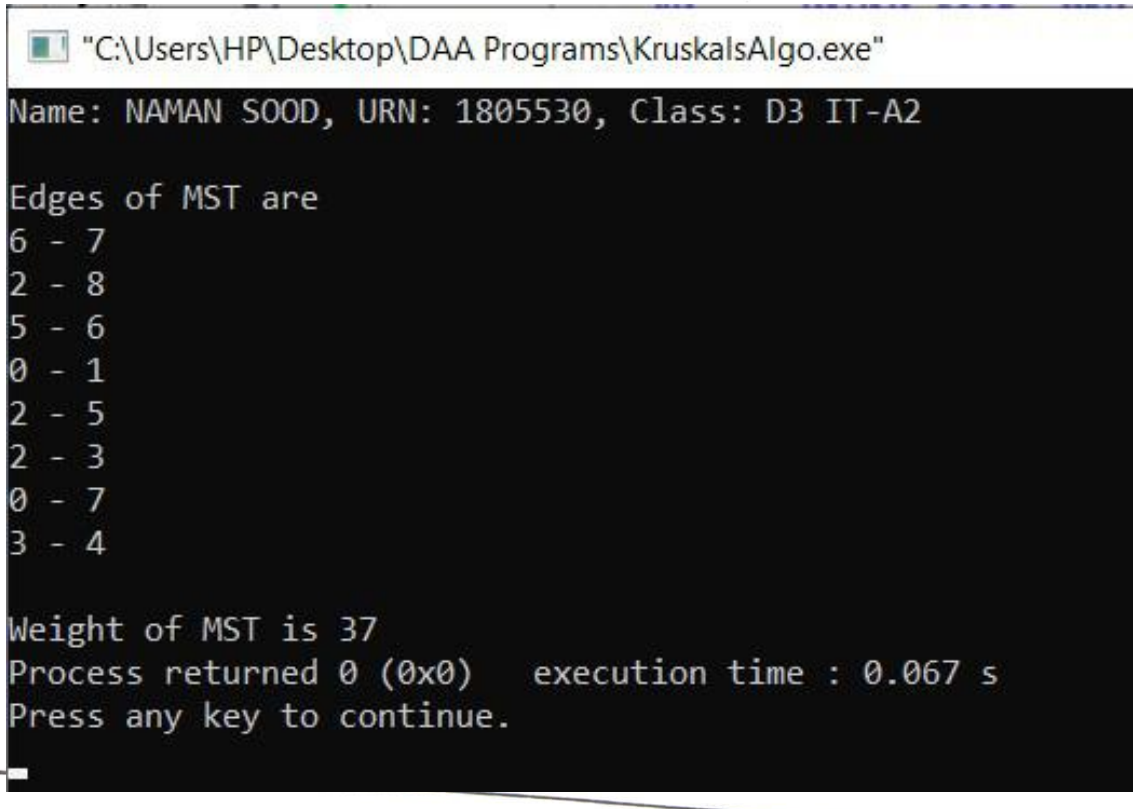
4.) So, Krusical's algorithm takes O(E log E) time.

5.) The value of E can be at most $O(V^2)$

6.) So, O(log V) and O(log E) are same.

```
g.addEdge(1,7,11);
g.addEdge(2,3,7);
g.addEdge(2,8,2);
g.addEdge(2,5,4);
g.addEdge(3,4,9);
g.addEdge(3,5,14);
g.addEdge(4,5,10);
g.addEdge(5,6,2);
g.addEdge(6,7,1);
g.addEdge(6,8,6);
g.addEdge(7,8,7);
cout<<"Edges of MST are\n";
int mst_wt = g.kruskalMST();
cout<<"\nWeight of MST is "<<mst_wt;
return 0;
}
```

OUT PUT



GURU NANAK DEV
ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

DESIGN AND ANALYSIS OF ALGORITHMS

Sixth Semester

Session:- Jan-July 2021

Batch:- 2018-2022

SUBMITTED TO:-
ER. PARMINDER KAUR WADHWA
DEPARTMENT OF IT
ASSISTANT PROFESSOR

SUBMITTED BY:-
NAMAN SOOD
D3 IT-A2
1805530

EXPERIMENT NO. :- 5

Program :-

```
# include <limits.h>
# include <stdio.h>
# define V 9
int min distance( int dist[], bool spt set[])
{
  int min = INT_MAX, min_ index
   for ( int v = 0; v<V ; v++)
          if (Spt Set[v] == false && dist [v] <= min)
            min = dist [v], min - index = v;
     return min_ index;
}
  Void Print Solution ( int dist [])
{
   Print f (" Vertex \t\t distance from source \n");
   for ( int i =0 ; i < v; i++)
      Print f ("% d \t \t % d \n", i , dist [i]);
}
  Void dijkstra ( int graph [V][V] , int src)
{
   int  dist [V];
   bool Spt Set [V];
   for ( int i = 0 ; i< V ; i++)
          dist [i] = int_MAX , Spt Set[i] = false;
```

Expt. No. 5 Page No. 20 Date :

EXPERIMENT NO. :- 5

Implement greedy algorithm to solve single source - shortest path problem.

Dijkstra's Algorithm :- Dijkstra's algorithm is very similar to Prim's algorithm for min- -imum spanning tree. Like Prim's MST, we generate a PST (Shortest Path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. at every step of algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source

Algorithm :-

```
Shortestpaths ( int V, float cost [] [SIZE], float
                dist [], int n )
// dist [j] , 1 <= j <= n is set to the length of the
// shortest path from vertex V to vertex j in the
// digraph G with n vertices. dist [V] is set to zero
// G is represented by its cost adjacency matrix
// cost [1:n] [1:n]
```

```
    dist[src] = 0;
    for (int count = 0; count < V-1; count++) {
        int u = min distance(dist, sptSet);
    sptSet[u] = true;
    for (int v = 0; v < V; v++)
        if (!sptSet[v] && graph[u][v] && dist[u]!
            = INT_MAX && dist[u] + graph[u][v] < dist[v])
        dist[v] = dist[u] + graph[u][v];
    }
     Print Solution(dist);
}
int main()
{
    int graph[V][V] = {{0,4,0,0,0,0,0,8,0},
                       {4,0,8,0,0,0,0,11,0},
                       {0,8,0,7,0,4,0,0,2},
                       {0,0,7,0,9,14,0,0,0},
                       {0,0,0,9,0,10,0,0,0},
                       {0,0,4,14,10,0,2,0,0},
                       {0,0,0,0,0,2,0,1,6},
                       {8,11,0,0,0,0,1,0,7},
                       {0,0,2,0,0,0,6,7,0}};
        dijkstra(graph,0);
        return 0;
}
```

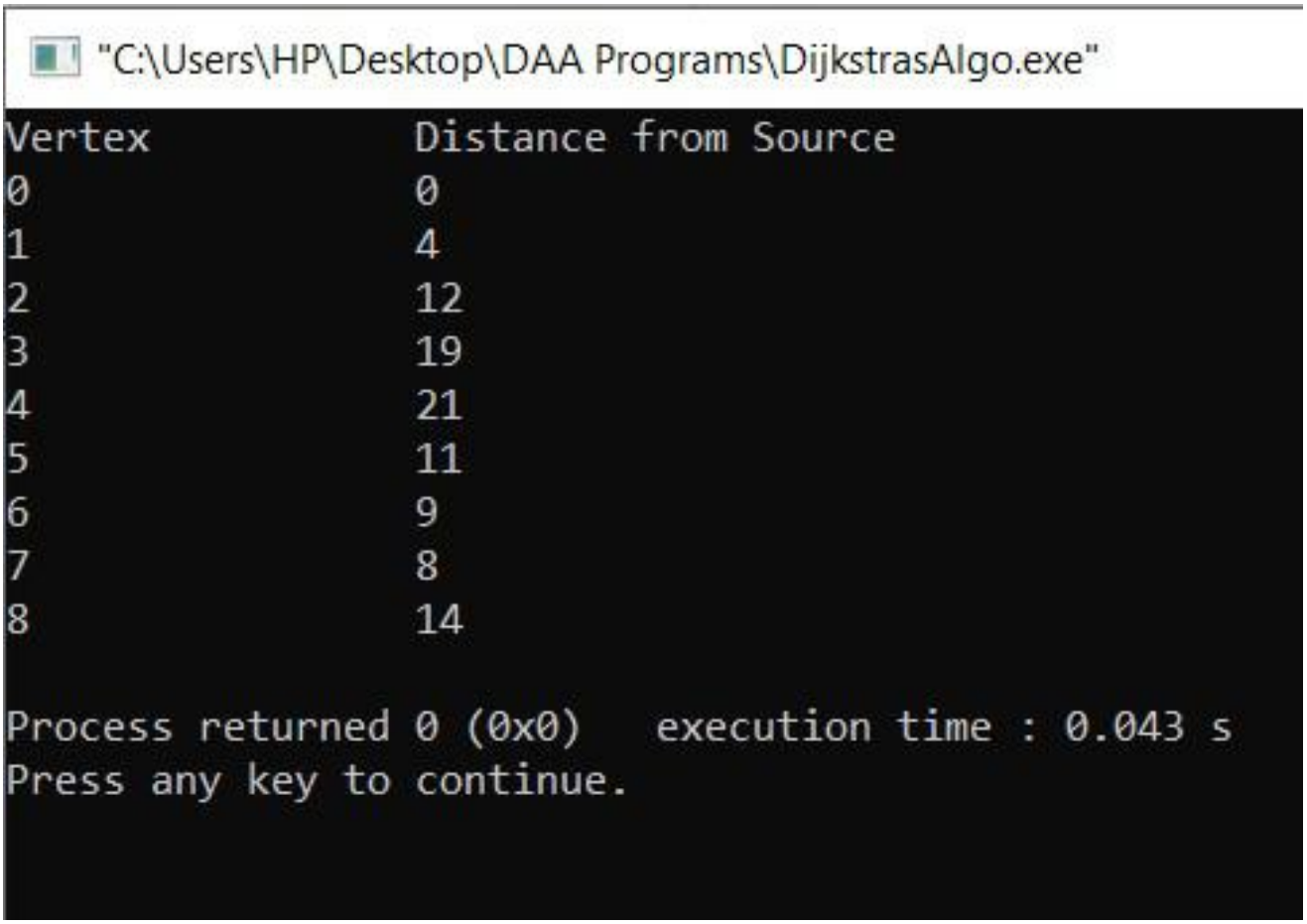
```
{
  int u; bool S[SIZE];
  for [int i=1; i<=n; i++) { // initialize S.
    S[i] = false; dist[i] = cost[v][i];
  }
  S[v] = true; dist[v] = 0.0; // Put v in S
  for ( int num = 2; num < n; num++) {
  // determine n-1 paths from v
  choose u from among those vertices not
  in S such that dist[u] is minimum;
  S[u] = true; // Put u in S.
  for ( int w = 1; w<=n; w++) // update distances
    if ((S[w] = false) && (dist[w] > dist[u] + cost[u]
        [w]))
      dist[w] = dist[u] + cost[u][w];
  }
}
```

Analysis of Dijkstra's Algorithm :-

Time complexity :- $O((V+E)\log V)$

E denotes Edges
V denotes Vertices

OUTPUT


"C:\Users\HP\Desktop\DAA Programs\DijkstrasAlgo.exe"
Vertex Distance from Source
0 0
1 4
2 12
3 19
4 21
5 11
6 9
7 8
8 14
Process returned 0 (0x0) execution time : 0.043 s
Press any key to continue.

Expt. No. 5 Page No. 22 Date :

Time Complexity Analysis :-

1) With adjacency representation of list, all vertices of graph can be traversed using BFS in $O(V+E)$ time.

2) in min heap, operations like extract-min and decrease key value takes $O(\log V)$ time

3) So, overall time complexity becomes $O(E+V) \times O(\log V)$ which is $O((E+V) \times \log V) = O(E \log V)$

4) This time complexity can be reduced to $O(E + V \log V)$ using Fibonacci heap.

EXPERIMENT NO. 6

Program :-

```
#include <bits/stdc++.h>
using namespace std;
int max(int a, int b)
{
 return (a > b) ? a : b;
}
 int Knapsack(int W, int wt[], int val[], int n)
{
   int i, W;
   int K[n+1][W+1];
for (i = 0; i <= n; i++)
{
   for (W = 0; W <= W; W++)
   {
      if (i == 0 || W == 0)
            K[i][W] = 0;
      Else if (wt[i-1] <= W)
            K[i][W] = max(val[i-1] + K[i-1]
            [W - wt[i-1]], K[i-1][W]);

   Else,
        K[i][W] = K[i-1][W];
   }
```

EXPERIMENT No. 6

Use Dynamic programming to solve Knapsack problem.

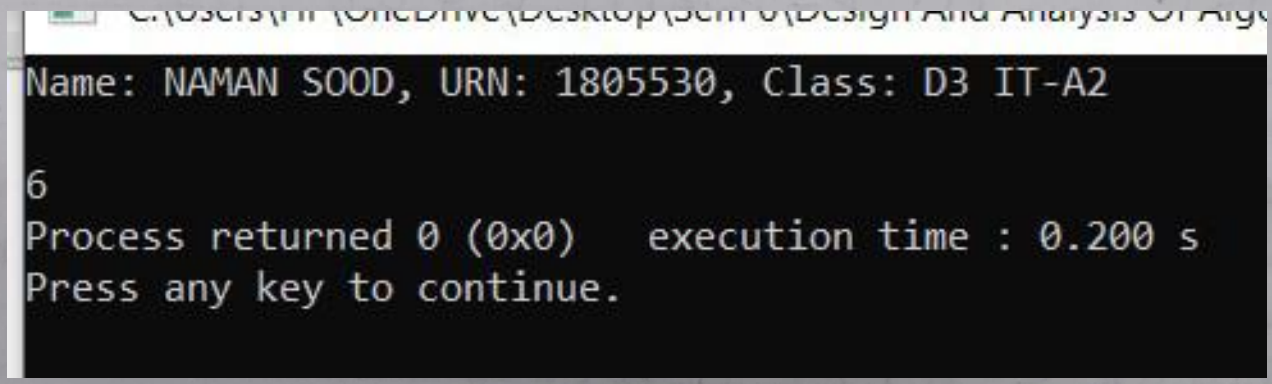
0/1 Knapsack Problem:- We are given n number of objects with weights Wi and profits Pi where I varies from 1 to n and also a knapsack with capacity m. The problem is, we have to fill the bag with help of n objects and the resulting profit has to be maximum. This Problem is similar to ordinary knapsack problem but we may not take a fraction of an object.

Algorithm for 0/1 Knapsack Problem:-

```
Void DKnap (float Pl), float W[], int x0, int n, float m)
{ Struct PW Pair[SIZE]; int b[MAXSIZE], next;
b[0]=1; Pair[1].P = Pair[1].W = 0.0; //S^0
b[1] = next = 2; // Next free spot in pair[]
for (int i = 1; i <= n-1; i++) { // Generates:
int k = t; int u = largest (Pair, W, t h, i, m);
for (int j = t; j <= u ; j++) { // Gen. S_i^{i-1} and merge
float PP = Pair[j].P+P[i]; float WW = Pair[j]
While ((k <= h) && (Pair[k].W <= WW)) {
Pair[next].P = Pair[k].P; Pair[next].W =
Pair[k]; next ++; k ++; }
```

```
}
return K[n][W];
}
int main()
{
cout << "Name: NAMAN SOOD, URN: 1805530,
        Class: D3 IT-A2 \n";
cout << endl;
   int Val[] = {2, 3, 1, 4};
   int Wt[] = {3, 4, 6, 5};
   int W = 8;
  int n = sizeof(Val) / sizeof(Val[0]);
   cout << knapsack(W, wt, val, n);
   return 0;
}
```

Output :-



```
if ((k<=h)&&(Pair[k].W==WW)){
if (PP < Pair[k].P) PP=Pair[k].P; k++;
if (PP > Pair[next-1].P){
Pair[next.P = PP; Pair[next].W=WW; next++;
} While ((k<=h)&& (Pair[k].P <= Pair[next
-1].P)) k++; }
// merge in Remaining terms from S^(i-1)
while (k<=h){
Pair[next].P = Pair[k].P; Pair[next].W=Pair
[k).W; next++; k++;
} // initialize for S^(i-1)
t = h+1; h = next-1; b[i+1] = next; }
traceback (P, W, Pair, X, m, n);
}
```

Time Complexity of 0/1 Knapsack Problem :-

Time complexity of 0/1 Knapsack Problem is $O(N*W)$

EXPERIMENT No. 7

Program :-

```
# include <bits/stdc++.h>
using namespace std;
# define V 4
# define INF 99999
Void print solution (int dist [] [V]);
{
 int dist [V] [V], i, j, k;
 for (i = 0 ; i < V ; i++)
          for (j = 0 ; j < V ; j++)
             dist [i] [j] = graph [i] [j];
 for ( k = 0 ; k < V; k++)
 {
   for (i = 0 ; i < V ; i++)
     {
        for (j = 0; j < V; j++)
          {
            if (dist [i] [k] + dist [k] [j] < dist [i] [j]
             dist [i] [j] = dist [i] [k] + dist [k] [j]
        }
        }
 }
   Print solution( dist );
 }
  Void print solution ( int dist [] [V])
```

EXPERIMENT No. 7

Solve all Pairs Shortest Path problem using dynamic programming (Floyd Warshall Algorithm)

Floyd Warshall Algorithm :- Floyd - Warshall Algorithm is an Algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. this Algorithm works for both the directed and undirected graphs. but, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative). floyd Warshall algorithm is also called as floyd's algorithm, Roy-floyd algorithm, Roy Warshall algorithm, or WFI algorithm. This algorithm follows the dynamic Programming approach to find the shortest Paths

Algorithm For Floyd Warshall Algorithm :-

n = no. of vertices.
A = matrix of dimension n*n
for k = 1 to n
  for i = 1 to n

```
{
  cout << " The following matrix shows the
     shortest distances " " between every pair
          of vertices \n ";
  for ( int i = 0; i < V; i++ )
  {
     for ( int j = 0; j < V; j++ )
     {
        if (dist [i][j] == INF )
              cout << " INF " << " ";
        else,
              cout << dist [i][j] << " ";
     }
     cout << endl;
  }
}
int main ()
 {
    cout << " Name : NAMAN SOOD, URN: 1805530,
    Class : D3 IT-A2 \n " ;
    cout << endl;
    int graph [V][V] = { { 0, 3, INF, 5 }
                         { 2, 0, INF, 4 }
                         { INF, 1, 0, INF },
                         { INF, INF, 2, 0 } };
    floyd Warshall (graph);
     return 0;
}
```

for $j = 1$ to n

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

Return A.

Time complexity of Floyd Warshall Algorithm:-

Time complexity of Floyd Warshall Algorithm is $O(V^3)$

Output :-

```
Name: NAMAN SOOD, URN: 1805530, Class: D3 IT-A2

The following matrix shows the shortest distances between every pair of vertices
0       3       7       5
2       0       6       4
3       1       0       5
5       3       2       0

Process returned 0 (0x0)   execution time : 0.143 s
Press any key to continue.
```

EXPERIMENT No. 8

Program :-

```
#define N8
#include <stdbool.h>
#include <stdio.h>
#include <bits/stdc++.h>
using namespace std;
Void Print Solution(int board[N][N])
{
 for(int i = 0; i < N ; i++) {
        for (int j = 0; j < N ; j++)
                Printf("%d ", board[i][j]);
          Printf("/n");

   }
}
 bool is safe(int board[ N][N], int row, int col)
 {
    int i , j ;
       for (i = 0; i < col ; i++)
              if (board [row][i])
                     return false;
    for (i = row ; j = col ; i >=0 && j >=0; i--
                                             j--)
      if (board[i][j])
                  return false;
    for ( i = row , j = col ; j >= 0 && i < N; i++, j--)
```

EXPERIMENT NO. 8

Use Backtracking to Solve 8-queens Problem.

8-Queen's Problem :- N-Queens problem is to Place n-queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal. It can be seen that for n = 1, the problem has a trivial solution, and no solution exists for n = 2 and n = 3. So, the eight queens problem is the Problem of placing eight queens on an 8x8 chessboard sketch that none of them attack one another (no two are in the same raw, column, or diagonal)

Algorithm for 8-queens Problem :-

```
Void NQueens (int k, int n)
// Using backtracking, this Procedure Prints all
// Possible placements of n queens on an n*n
// chessboard so that they are nonattacking. {
 for (int i=1; i<=n; i++){
  if (Place(k,i)) {
  x[k] = i;
  if (k == n) {
```

```
if ( board[i][j])
        return false;
    return true
}
bool solveNQutil (int board[N][N], int col)
{
  if (col >= N)
        return true;
  for (int i = 0; i<N; i++) {
        if (issafe (board, i, col)) {
            board [i] [col] = 1;
  if (solveNQutil (board, col+1))
            return true;
          board [i] [col] = 0;
    }
  }
  return false;
  }
  bool solveNQ()
{
  int board [N] [N] = {{ 0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0}};
```

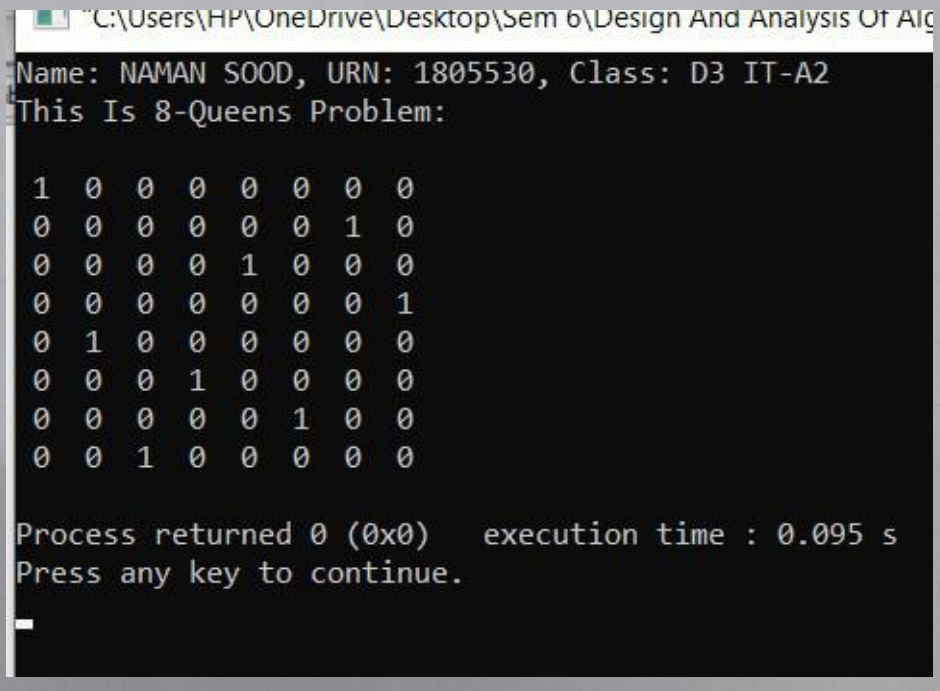
```
for (int j = 1; j <= n; j++)
cout << x[j] << " ";
cout << endl; }
Else N queens (k+1, n);
}}
}.
```

Time Complexity of 8-queens Problem :-

Time complexity of 8-queens Problem is $O(N^N)$.

```
if (solveNQUtil(board, 0) == false) {
    Printf("Solution does not exist");
    return false;
}
Print solution(board);
  return true;
}
int main()
{
  cout << "Name :- NAMAN SOOD, URN : 1805530,
        Class :- D3 IT-B2 \n";
  cout << "This is 8 queens Problem : \n";
   cout << endl;
     solveNQ()
     return 0;
}
```

Output :-



EXPERIMENT No. 9

Program :-

```
#include<iostream>
using namespace std;
void display subset (int subset[], int size){
    for (int i=0; i<size; i++){
        cout << subset[i] << " ";
    }
    cout << endl;
}
void subset sum (int set[], int subset[], int n,
int subsize, int total, int node count, int sum)
{
    if(total == sum) {
    display subset ( subset, sub size );
    subset sum (set, subset, n, subsize-1, total-set,
     [node count], node count+1, sum);
    return;
    }
    else,
    {
     for (int i = node count; i<n; i++) {
      subset[subsize] = set[i];
     subset sum (set, subset, n, subsize+1, total +
      set[i], i+1, sum);
}
```

Expt. No. 9 Page No. 31 Date :

EXPERIMENT No. 9

Solve Sum of Subsets problem using Backtracking.

Sum of Subsets Problem :- The Subset sum problem is to find subsets of the given set S = (S1 S2 S3 ---- Sn) where the elements of the set S are n positive integers in such a manner that $s \in S$ and sum of the elements of subsets is equal to some positive integer 'X' The subset sum problem can be solved by using the backtracking approach. In this implicit tree is a binary tree. The root of the tree is selected in such a way that represents that no decision is yet taken on any input. We assume that the elements of the given set are arranged in increasing order: $S1 \leq S2 \leq S3 \leq \cdots \leq Sn$.

The left child of the root node indicated that we have to include 'Si' from the set 'S' and the right child of the root indicates that we have to execute 'Si' Each node stores the total of the Partial Solution elements. If at any stage the sum equals to "X" then the search is successful and terminates.

```
    }
}
void findSubset (int set[], int size, int sum){
    int *subset = new int[size];
    subsetSum(set, subset, size, 0, 0, sum);
    delete[] subset;
}

int main(){
cout << "Name : NAMAN SOOD, URN : 1805530,
        Class : D3 IT-A2 \n";
    cout << endl;
    cout << "the subsets are : \n";
    int weights[] = {5, 10, 12, 13, 15, 18};
    int size = 6;
    findSubset(weights, size, 30);
}
```

Output :-

```
Name: NAMAN SOOD, URN: 1805530, Class: D3 IT-A2

The Subsets are:
5  10  15
5  12  13
12  18

Process returned 0 (0x0)   execution time : 0.079 s
Press any key to continue.
```

Algorithm for the Sum of Subsets Problem :-

```
Void Sum of Sub (float s, int k, float r)
// find all subsets of W[1:n] that sum to m.
// The values of X[j], 1<=j<k, we have already
// been determined. S = Σ^(k-1) W[j] are in non-
// decreasing order.
// It is assumed that W[1] <= m and Σ^n W[j]>=
// m. { Generate left child. Note that S+W[k]<=m.
X[k] = 1;
if (S+W[k] == m) {
for (int j = 1; j <= k ; j++)
cout << X[j] << " ";
cout << endl;
}
// There is no recursive call here.
// as W[j] > 0, 1<=j <= n.
else if (S + W[k] + W[k+1] <= m)
Sum of Sub (S+W[k], k+1, r-W[k]);
// Generate right child
if ((S+r-W[k] >= m) && (S+W[k+1] <= m))
{ X[k] = 0
Sum of Sub (S, k+1, r-W[k]); }
}
```

Time Complexity of Sum of Subsets Problem :-

Time complexity of Sum of Subsets Problem is $O(2^N)$

EXPERIMENT No.10

Program :-

```
# include <iostream>
using namespace std;
Void full suffixmatch ( int shift Arr [], int border
 Arr [], string Pattern ) {
  int n = Pattern . size ();
  int i  = n;
   int j  = n +1;
   border Arr [i] = j;
   While ( i >0 ) {
   While ( j <=n && Pattern [i-1] != Pattern [j-1] )
   {  if ( Shift Arr [j] == 0)
         Shift Arr [j] = j - i;
      j = border Arr [j];
   }
    i --;
    j --;
     border Arr [i] = j;
  }
}
Void Partial suffix match (int shift Arr [], int border
[], string Pattern ) {
 int n = Pattern . size ();
  int j;
     j = Border Arr [0];
```

EXPERIMENT No. 10

Implement Boyer Moore Algorithm.

Boyer Moore Algorithm :- It is considered as the most Efficient string matching algor-ithm. A simplified version of it or the entire algorithm is used in text editors for search and substitute commands. the Algorithm scans the characters of the pattern form right to left begining with the rightmost one. in case of a mismatch (or a complete match of the whole pattern) It uses two Pre-computed functions to shift the window to right.

The two shifts of the functions are as follows.

- **Good Suffix Shift or matching Shift** :- Aligns only matching pattern characters against target characters already successfully matched.

- **Bad character shift or occurance shift** :- Adaids repeating unsuccessfull comparisons against target character.

```
for (int i = 0; i < m; i++) {
    if (shiftArr[i] == 0)
        shiftArr[i] = j;
        if (i == j)
        j = borderArr[j];
    }
}
Void SearchPattern (string mainString, string pattern,
int array[], int * index) {
int PatLen = pattern.size();
 int strlen = mainString.size();
int borderArray [PatLen + 1];
 int shiftarray [Patlen + 1];
 for (int i = 0; i <= Patlen; i++) {
 shiftarray[i] = 0;
}
full suffix match (shiftarray, borderarray, pattern);
Partial SuffixMatch( shiftarray, borderarray, pattern);
int shift = 0;
while (shift <= (strlen - Patlen)) {
   int j = Patlen - 1
while (j >= 0 && pattern[j] == mainString[shift
+j]) {
j--;
}
if (j < 0) {
(*index)++;
 array[(*index)] = shift;
```

Expt. No. 10 Page No. 34 Date :

Algorithm for Boyer moore Algorithm :-

- Full Suffix match (Shift Array, border Array, Pattern)

```
Begin
  // n = Pattern length
  // j = n
  // j = n+1
border Array [i] := j
while i > 0, do
  while j <= n AND Pattern [i-1] ≠ Pattern [j-1] do,
  if shift array [j] = 0, then
  shift array [j] = j - i ;
  j = border array [j] ;
  done
  decrease i and j by 1
  border array [i] = j
  done
  End
```

- Partial Suffix match (Shift array, border array, Pattern)

```
Begin
  // n = Pattern length
  // j = border Array [0]
  for index of all characters 'i' of Pattern, do
    if shift Array [i] = 0, then
      shift array [i] = j
    if i = j then
      j = border Array [j]
```

```
        shift += shiftArray[0];
    } else {
        shift += shift array[j+1];
    }
}
}
int main() {
cout << "Name :- NAMAN SOOD, URN : 1805530,
        Class : D3 IT- A2";
 cout << endl;
String main string = "ABAAA BCD BB ABCDDEB
                    CABCD";
String Pattern :- "ABCD";
int locArray [main string.size()];
 int index = -1;
Search Pattern (main string, Pattern, locArray
    , &index);
for (int i = 0 ; i <= index ; i++) {
 cout << "Pattern found at Position :"
    << locArray[i] << endl;
}
}
```

done
End.

- Search Pattern (text, Pattern)
Begin
// PatLen = Pattern length
// StrLen = text size.
for all entries of shift array, do
set all entries to 0
done.
call full suffix match (Shift array, border array, Ptrn)
call PartialSuffix match (Shift array, border array, Pattern)
shift = 0
while shift <= (StrLen - PatLen), do
j = Patlen - 1
while j >= 0 and Pattern[j] = text[shift+j], do.
decrease j by 1
done
if j < 0, then
Print the shift as, there is a match
Shift = shift + shift array [0]
Else
shift = shift + shift array [j+1]
done
End.

Output :-

```
Name: NAMAN SOOD, URN: 1805530, Class: D3 IT-A2
Pattern found at position: 4
Pattern found at position: 10
Pattern found at position: 18

Process returned 0 (0x0)   execution time : 0.192 s
Press any key to continue.
```

Time Complexity of Boyer Moore Algorithm :-

time complexity of Boyer Moore Algorithm is $O(mn)$, where m is length of substring and n is length of string.