VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Aishwarya A G (1BM21CS011)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
June-2023 to September-2023

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **AISHWARYA A G (1BM21CS011),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Sneha S Bagalkot
Assistant professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

PROGRAM -1

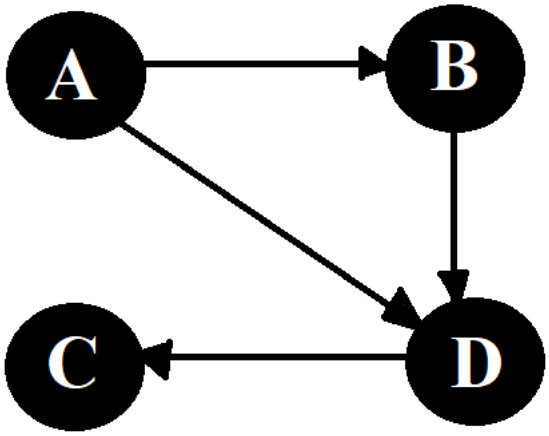
Write program to do the following:
a. Print all the nodes reachable from a given starting node in a digraph using BFS method.
b. Check whether a given graph is connected or not using DFS method.

a)BFS Traversal-

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

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
for(i=0;i<n;i++)
if(a[v][i] && !visited[i])
q[++r]=i;
if(f<=r)
{
visited[q[f]]=1;
bfs(q[f++]);
}
}
void main()
{
int v;
printf("\n Enter the number of vertices:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
q[i]=0;
visited[i]=0;
}
printf("\n Enter graph data in matrix form:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
printf("\n Enter the starting vertex:");
scanf("%d",&v);
```

```
bfs(v);
printf("\n The node which are reachable are:\n");
for(i=0;i<n;i++)
if(visited[i])
printf("%d\t",i);
getch();
}
```

Output:



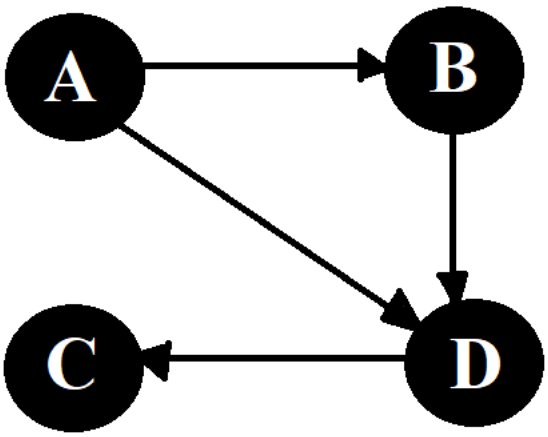

```
Enter the number of vertices:4

Enter graph data in matrix form:
1 1 1 1
0 0 1 0
1 1 0 0
1 1 0 0

Enter the starting vertex:1

The node which are reachable are:
1       2       3       4
```

b)DFS Traversal-

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

```
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v) {
        int i;
        reach[v]=1;
        for (i=1;i<=n;i++)
          if(a[v][i] && !reach[i]) {
                printf("\n %d->%d",v,i);
                dfs(i);
        }
}
void main() {
        int i,j,count=0;
        printf("\n Enter number of vertices:");
        scanf("%d",&n);
        for (i=0;i<n;i++) {
                reach[i]=0;
                for (j=0;j<n;j++)
                   a[i][j]=0;
        }
        printf("\n Enter the adjacency matrix:\n");
        for (i=0;i<n;i++)
          for (j=0;j<n;j++)
           scanf("%d",&a[i][j]);
        dfs(0);
        printf("\n");
        for (i=0;i<n;i++) {
                if(reach[i])
                   count++;
        }
        if(count==n)
          printf("\n Graph is connected"); else
          printf("\n Graph is not connected");
        getch();
}
```

Output-




```
Enter number of vertices:4

Enter the adjacency matrix:
1 0 0 1
1 1 1 1
0 0 0 1
1 0 0 0

1->2    2->3    3->4

Graph is connected
```

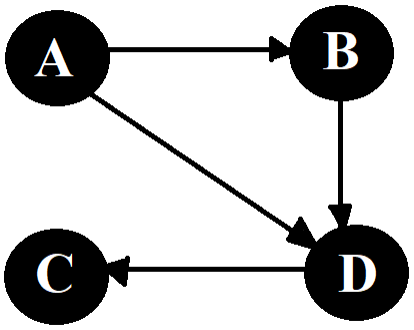
PROGRAM -2

Write a program to obtain the Topological ordering of vertices in a given digraph.

```
#include<stdio.h>
void dfs(int);
int a[10][10],vis[10],exp[10],n,j,m;

void main()
{
  int i,x,y;
  printf("enter the number of vertices\n");
  scanf("%d",&n);
  for(i=1;i<=n;i++)
  {
    for(j=1;j<=n;j++)
    {
        a[i][j]=0;
    }
    vis[i]=0;
  }
  printf("enter the number of edges\n");
  scanf("%d",&m);
  for(i=0;i<m;i++)
  {
    printf("enter an edge\n");
    scanf("%d %d",&x,&y);
    a[x][y]=1;
  }
  j=0;
  for(i=0;i<n;i++)
  {
    if(vis[i]==0)
        dfs(i);
  }
  printf("topological sort\n");
  for(i=n-1;i>=0;i--)
  {
    printf("%d",exp[i]);
  }
  getch();
```

```
}

void dfs(int v)
{
  int i;
  vis[v]=1;
  for(i=0;i<n;i++)
  {
    if(a[v][i]==1 && vis[i]==0)
    dfs(i);
  }
  exp[j++]=v;
}
```

Output:




```
Enter the number of vertices : 4
Enter the adjacency matrix -
0 1 1 0
0 0 1 0
0 0 0 1
0 0 0 0
Topological Sorting (JOB SEQUENCE) is:-
1 2 3 4

Process returned 0 (0x0)   execution time : 13.063 s
Press any key to continue.
```

PROGRAM -3

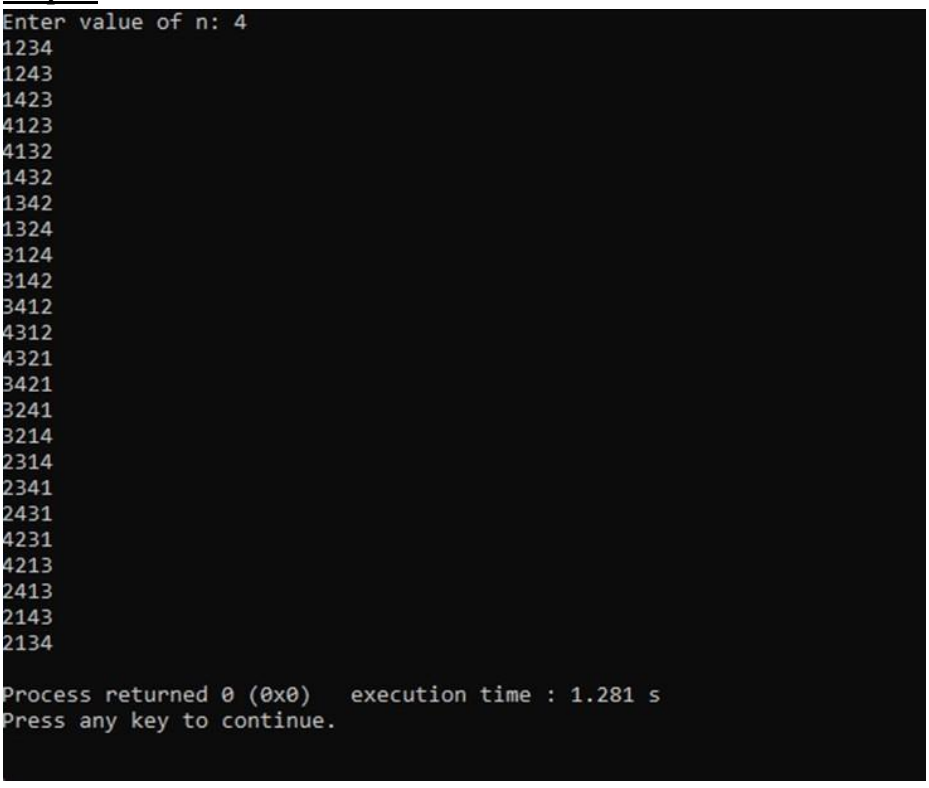
Implement Johnson Trotter algorithm to generate permutations.

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

int LEFT_TO_RIGHT = 1;
int RIGHT_TO_LEFT = 0;
int n;

int searchArr(int a[], int n, int mobile)
{
   int i;
        for (i = 0; i < n; i++)
                if (a[i] == mobile)
                        return i + 1;
}

int getMobile(int a[], int dir[], int n)
{
        int mobile_prev = 0, mobile = 0;
        for (int i = 0; i < n; i++) {
                if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {
                        if (a[i] > a[i - 1] && a[i] > mobile_prev) {
                                mobile = a[i];
                                mobile_prev = mobile;
                        }
                }


                if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {
                        if (a[i] > a[i + 1] && a[i] > mobile_prev) {
                                mobile = a[i];
                                mobile_prev = mobile;
                        }
                }
        }

        if (mobile == 0 && mobile_prev == 0)
                return 0;
        else
                return mobile;
}
```

```
void printOnePerm(int a[], int dir[], int n)
{
	int mobile = getMobile(a, dir, n);
	int pos = searchArr(a, n, mobile);

	if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT){
		int temp = a[pos - 1];
	a[pos - 1] = a[pos - 2];
	a[pos - 2] = temp;}

	else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT){
	int temp = a[pos];
	a[pos] = a[pos - 1];
	a[pos - 1] = temp;
	}


	for (int i = 0; i < n; i++) {
		if (a[i] > mobile) {
			if (dir[a[i] - 1] == LEFT_TO_RIGHT)
				dir[a[i] - 1] = RIGHT_TO_LEFT;
			else if (dir[a[i] - 1] == RIGHT_TO_LEFT)
				dir[a[i] - 1] = LEFT_TO_RIGHT;
		}
	}

	for (int i = 0; i < n; i++)
		printf("%d ",a[i]);
	printf("\n");
}


int fact(int n)
{
	int res = 1;
	for (int i = 1; i <= n; i++)
		res = res * i;
	return res;
}

void printPermutation(int n)
{
	int a[n];
```

```
        int dir[n];
        for (int i = 0; i < n; i++) {
                a[i] = i + 1;
    printf("%d ",a[i]);
        }
        printf("\n");

        for (int i = 0; i < n; i++)
                dir[i] = RIGHT_TO_LEFT;

        for (int i = 1; i < fact(n); i++)
                printOnePerm(a, dir, n);
}


void main()
{
        printf("Enter the value of n: ");
        scanf("%d",&n);
        printPermutation(n);

}
```

Output:



PROGRAM -4

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define max 200000000

int a[max];
int b[max];

void merging(int low, int mid, int high) {
  int l1, l2, i;

  for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
    if(a[l1] <= a[l2])
      b[i] = a[l1++];
    else
      b[i] = a[l2++];
  }

  while(l1 <= mid)
    b[i++] = a[l1++];

  while(l2 <= high)
    b[i++] = a[l2++];

  for(i = low; i <= high; i++)
    a[i] = b[i];
}

void sort(int low, int high) {
  int mid;

  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
```

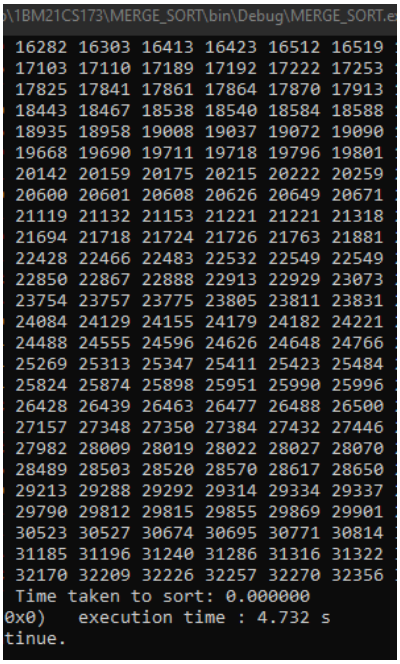
```
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}

int main() {
  int i,n;
  clock_t start,end;
  printf("Enter the number of elements: ");
  scanf("%d",&n);
  for(i=0;i<n;i++){
   a[i]=rand();
  }
printf("List before sorting\n");
  for(i = 0; i <n; i++)
     printf("%d ", a[i]);
    start=clock();
  sort(0, n-1);
    end=clock();
  printf("\nList after sorting\n");
  for(i = 0; i <n; i++)
     printf("%d ", a[i]);
     double tt=(double)(end-start)/CLOCKS_PER_SEC;
   printf("\nTime taken to execute: %f",tt);
}
```

Output:

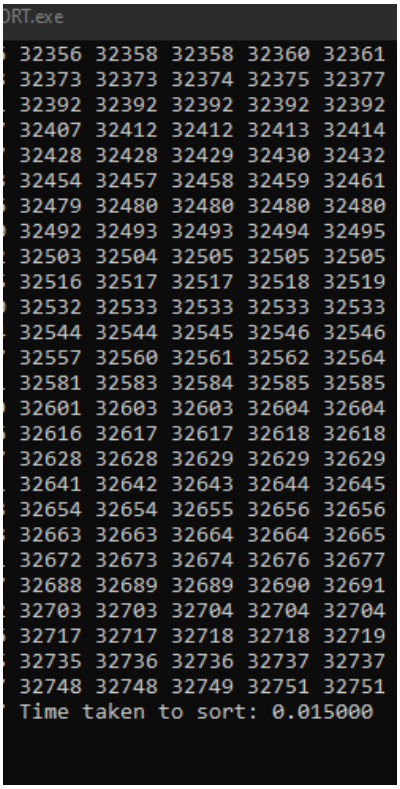
Input- 5

```
Enter the number of elements: 5
Enter the numbers

List after sorting
41 6334 18467 19169 26500 Time taken to sort: 0.000000
Process returned 0 (0x0)   execution time : 1.594 s
Press any key to continue.
```

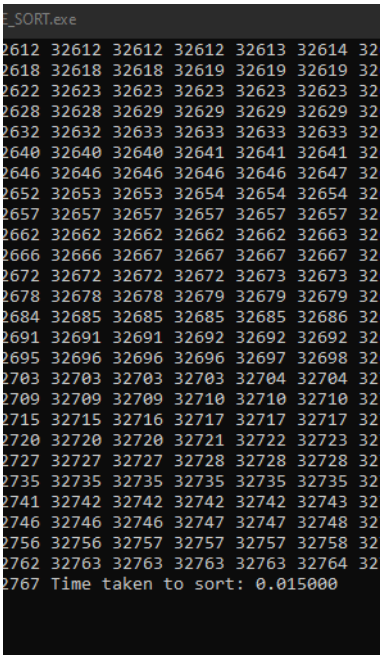
Input-1000



Input- 60000



Input-100000



PROGRAM -5

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

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

void swap(int* a, int* b)
{
	int t = *a;
	*a = *b;
	*b = t;
}

int partition(int arr[], int low, int high)
{
	int pivot = arr[low];
	int i = low;
```

```
        int j=high+1;
      while(i<=j){
                do {
        i++;
                }while(pivot>=arr[i]);
                do{
        j--;
                }while(pivot<arr[j]);
                if(i<j){
       swap(&arr[i],&arr[j]);
                }}
                if(i>j){
        swap(&arr[low], &arr[j]);}
        return j;
}

void quickSort(int arr[], int low, int high)
{
        if (low < high) {
                int k = partition(arr, low, high);
                quickSort(arr, low, k - 1);
                quickSort(arr, k + 1, high);
        }
}
void main()
{
   int N,i;
   clock_t start,end;
        printf("Enter the number of elements: ");
        scanf("%d",&N);
        int arr[N];
        for(i=0;i<N;i++){
     arr[i]=rand();
        }
        printf("\nArray before sorting: \n");
        for (int i = 0; i < N; i++){
                printf("%d ", arr[i]);}
        start=clock();
        quickSort(arr, 0, N - 1);
        end=clock();
```

```
        printf("\nSorted array: \n");
        for (int i = 0; i < N; i++){
                printf("%d ", arr[i]);}
         double tt=(double)(end-start)/CLOCKS_PER_SEC;
   printf("\n\nTime taken to execute: %f",tt);
}
```

Output:

Input - 10

```
Enter size of array: 10
Enter the 1 element: 9
Enter the 2 element: 8
Enter the 3 element: 7
Enter the 4 element: 5
Enter the 5 element: 6
Enter the 6 element: 4
Enter the 7 element: 3
Enter the 8 element: 1
Enter the 9 element: 2
Enter the 10 element: 10
1 2 3 4 5 6 7 8 9 10
The time taken to sort: 0.000000
Process returned 0 (0x0)   execution time : 16.305 s
Press any key to continue.
```

Input - 100000

```
C:\Users\Admin\Desktop\1BM21CS173\QUICK_SORT\main.exe
1 32701 32701 32701 32701 32701 32702 32702 32702 32702
4 32704 32704 32704 32704 32704 32704 32705 32705 32705
6 32706 32707 32707 32707 32707 32707 32707 32707 32708
9 32709 32709 32709 32709 32710 32710 32710 32710 32710
2 32712 32712 32712 32712 32712 32712 32712 32713 32713
4 32714 32714 32714 32715 32715 32716 32716 32716 32716
7 32717 32717 32718 32718 32718 32718 32718 32718 32719
0 32720 32720 32720 32720 32720 32720 32720 32720 32721
2 32722 32722 32723 32723 32723 32723 32723 32723 32723
5 32725 32725 32726 32726 32726 32726 32726 32726 32726
8 32728 32728 32728 32729 32729 32729 32729 32730 32730
1 32731 32732 32732 32732 32732 32732 32732 32732 32732
4 32734 32734 32734 32734 32735 32735 32735 32735 32735
6 32736 32737 32737 32737 32737 32737 32737 32737 32737
0 32740 32740 32741 32741 32741 32741 32741 32741 32741
3 32743 32743 32743 32743 32743 32743 32744 32744 32744
5 32745 32745 32745 32745 32745 32745 32745 32745 32745
6 32746 32746 32747 32747 32747 32747 32747 32747 32747
8 32748 32748 32749 32749 32749 32749 32749 32749 32749
1 32751 32751 32751 32751 32751 32752 32752 32752 32752
4 32754 32755 32755 32755 32755 32755 32756 32756 32756
7 32757 32757 32758 32758 32758 32758 32758 32758 32758
0 32760 32760 32760 32760 32760 32760 32761 32761 32761
2 32762 32763 32763 32763 32763 32763 32763 32763 32763
5 32765 32765 32765 32765 32765 32765 32765 32766 32766
7 32767 32767 32767 32767 32767 32767 32767 32767 32767
The time taken to sort: 0.000000
Process returned 0 (0x0)   execution time : 12.094 s
Press any key to continue.
```

Input- 200000

```
0 32680 32681 32681 32682 32682 32683 32684 32684 32684 32685 32685
8 32688 32688 32688 32689 32689 32689 32690 32691 32691 32691 32691
3 32693 32693 32694 32694 32694 32694 32694 32695 32695 32696 32696
0 32700 32700 32701 32701 32702 32702 32702 32703 32703 32703 32703
6 32706 32707 32707 32707 32707 32708 32708 32709 32709 32709 32709
2 32712 32713 32713 32713 32713 32713 32714 32714 32715 32715 32716
8 32718 32719 32719 32719 32719 32719 32720 32720 32720 32720 32720
4 32725 32725 32725 32725 32725 32726 32726 32727 32727 32727 32727
1 32731 32732 32733 32733 32733 32733 32734 32734 32735 32735 32735
7 32737 32737 32737 32738 32738 32739 32739 32740 32741 32742 32742
4 32744 32745 32745 32745 32745 32746 32746 32746 32746 32746 32746
1 32751 32751 32751 32752 32752 32754 32754 32756 32756 32756 32757
0 32760 32760 32761 32761 32762 32762 32762 32762 32762 32763 32763
5 32766 32766 32766 32767 32767 32767 32767 32767 32767

Time taken to execute: 0.015000
```

Input- 500000

```
9 32749 32749 32749 32749 32749 32749 32750 32750 32750
1 32751 32751 32751 32751 32751 32751 32751 32751 32751
2 32752 32752 32752 32752 32752 32752 32752 32752 32752
4 32754 32754 32754 32754 32754 32754 32754 32754 32754
5 32756 32756 32756 32756 32756 32756 32756 32756 32756
7 32757 32757 32757 32757 32757 32757 32757 32757 32758
8 32758 32758 32758 32758 32759 32759 32759 32759 32759
9 32759 32759 32760 32760 32760 32760 32760 32760 32760
1 32761 32761 32761 32761 32761 32762 32762 32762 32762
3 32763 32763 32763 32763 32763 32763 32763 32763 32763
4 32764 32764 32764 32764 32764 32764 32764 32764 32765
5 32765 32765 32765 32765 32765 32765 32766 32766 32766
6 32766 32766 32766 32766 32766 32766 32766 32767 32767
7 32767 32767 32767 32767 32767 32767 32767 32767 32767

Time taken to execute: 0.047000
```

PROGRAM -6

Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

```
#include<stdio.h>
void swap(int *a,int *b)
{
   int temp = *a;
   *a = *b;*b = temp;
}
void bottom_up_heapify(int n ,int a[])
{
   int p,item,c;
   for(int i=(n-1)/2;i>=0;i--)
   {
      p =i;
      item = a[p];
      c = 2*p+1;
      while(c<n)
      {
         if(c+1<n){
            if(a[c]<a[c+1])
               c = c+1;
         }
         if(item<a[c])
         {
            a[p] = a[c];
            p = c;
            c = 2*p+1;
         }
         else{
            break;
         }
      }
      a[p] = item;
   }
}

void heap_sort(int n,int a[])
{

   bottom_up_heapify(n,a);
```

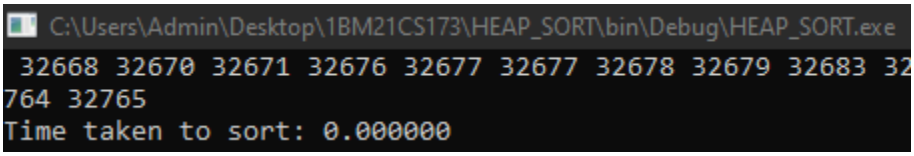
```
    for(int i=n-1;i>=0;i--)
    {
        swap(&a[0],&a[i]);
        bottom_up_heapify(i,a);
    }
}

int main()
{
    int n;
    printf("Enter the value of n");
    scanf("%d",&n);
    int a[n];
    printf("Enter the elements of the array");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    heap_sort(n,a);
    printf("Sorted elements : \n");
    for(int i=0;i<n;i++)
    {
        printf("%d\t",a[i]);
    }
    return 0;
}
```

OUTPUT:

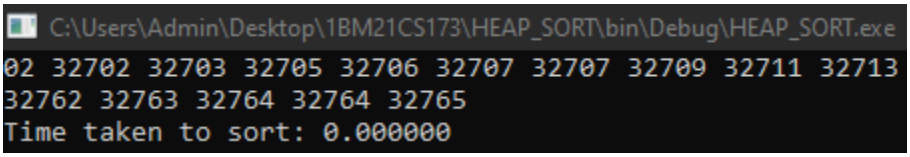
For n=5000

```
 32468 32469 32471 32471 32482 32488 32498 32499 32501 325
676 32677 32678 32683 32685 32686 32701 32702 32702 32702
Time taken to sort: 0.000000
```

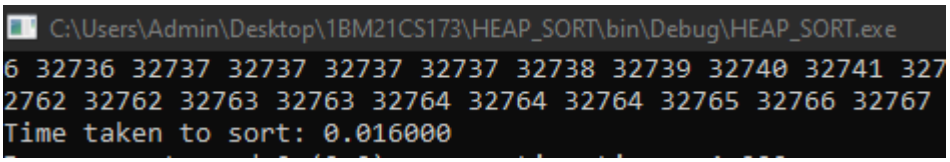
For n=10000



For n=20000



For n=50000



PROGRAM -7

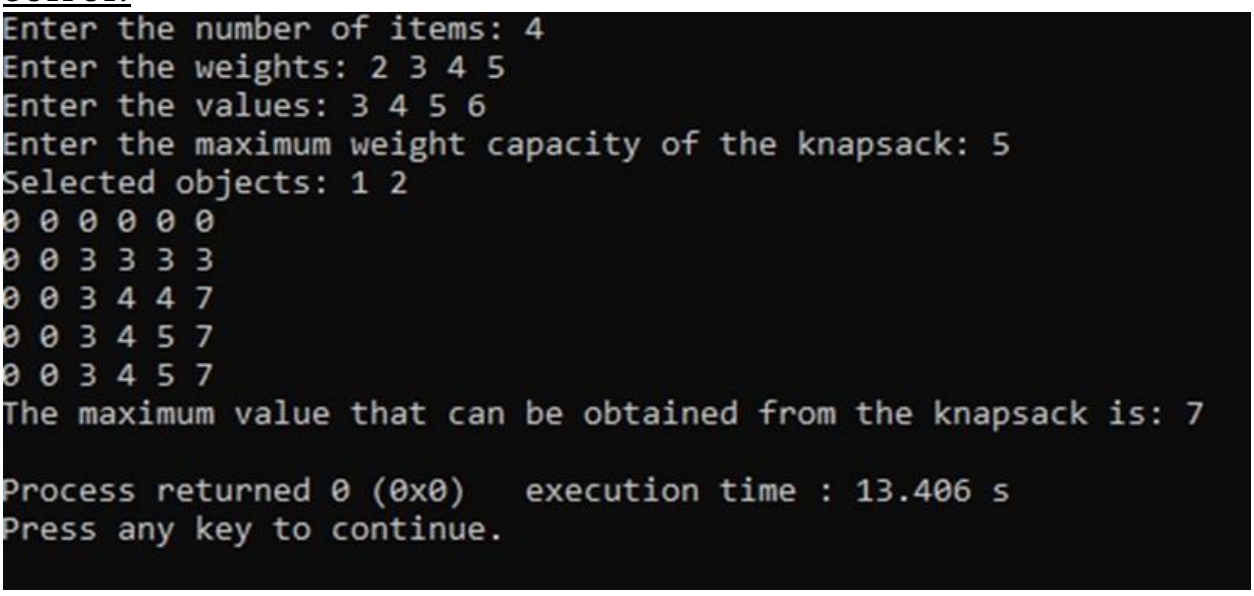
Implement 0/1 Knapsack problem using dynamic programming.

```
#include <stdio.h>
#include <stdbool.h>

int max(int a, int b) {
   return (a > b) ? a : b;
}

int knapsack(int W, int weights[], int values[], int n) {
   int dp[n + 1][W + 1];

   for (int i = 0; i <= n; i++) {
      for (int w = 0; w <= W; w++) {
         if (i == 0 || w == 0)
            dp[i][w] = 0;
         else if (weights[i - 1] <= w)
            dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
         else
            dp[i][w] = dp[i - 1][w];
```

```
        }
    }

    // Backtrack to find the selected items
    int selected[n];
    int i = n, w = W, count = 0;

    while (i > 0 && w > 0) {
        if (dp[i][w] != dp[i - 1][w]) {
            selected[count++] = i;
            w -= weights[i - 1];
            i--;
        } else {
            i--;
        }
    }

    printf("Selected objects: ");
    for (int j = count - 1; j >= 0; j--)
        printf("%d ", selected[j]);
    printf("\n");
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            printf("%d ",dp[i][w]);
        }
        printf("\n");
    }

    return dp[n][W];
}

int main() {
    int n, W;
    printf("Enter the number of items: ");
    scanf("%d", &n);

    int weights[n], values[n];
    printf("Enter the weights: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &weights[i]);
```

```
    printf("Enter the values: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &values[i]);

    printf("Enter the maximum weight capacity of the knapsack: ");
    scanf("%d", &W);

    int result = knapsack(W, weights, values, n);
    printf("The maximum value that can be obtained from the knapsack is: %d\n", result);

    return 0;
}
```

OUTPUT:

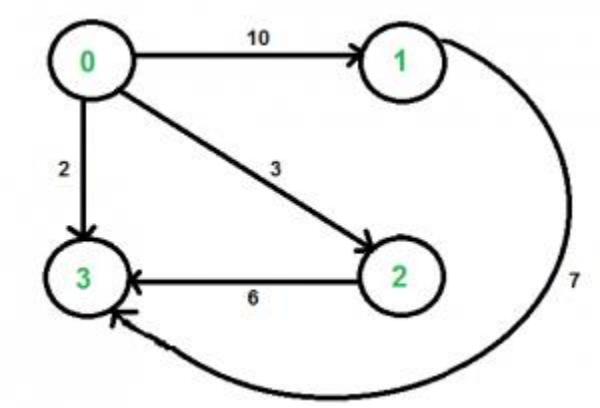


PROGRAM -8

Implement All Pair Shortest paths problem using Floyd's algorithm.

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

void main(){
   int n,i,j,k;
printf("Enter the number of vertices: ");
scanf("%d",&n);
int d[n][n],adj[n][n];
printf("Enter the weighted adjacency matrix:(9999 for infinity)\n");
for(i=0;i<n;i++){
   for(j=0;j<n;j++){
      scanf("%d",&adj[i][j]);
   }
}
 for(i=0;i<n;i++){
      for(j=0;j<n;j++){
         d[i][j]=adj[i][j];
      }
   }
   for(k=0;k<n;k++){
      for(i=0;i<n;i++){
         for(j=0;j<n;j++){
         if(d[i][j]<d[i][k]+d[k][j])
            d[i][j]=d[i][j];
         else
            d[i][j]=d[i][k]+d[k][j];
         }
      }
   }
   printf("Distance matrix:\n");
for(i=0;i<n;i++){
   for(j=0;j<n;j++){
     printf("%d  ",d[i][j]);
   }
   printf("\n");
}
}
```

Output:




```
Enter the number of vertices: 4
Enter the weighted adjacency matrix:(9999 for infinity)
0 10 3 2
9999 0 9999 7
9999 9999 0 6
9999 9999 9999 0
Distance matrix:
0  10  3  2
9999  0  9999  7
9999  9999  0  6
9999  9999  9999  0

Process returned 4 (0x4)   execution time : 69.288 s
Press any key to continue.
```

PROGRAM -9

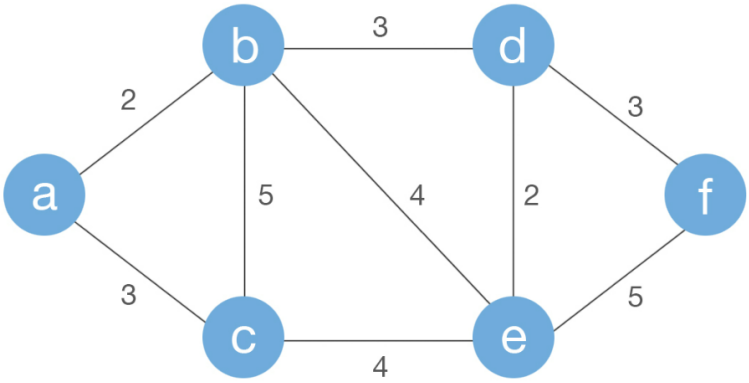
Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.

PRIM'S ALGORITHM:

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

int cost[10][10],vt[10],et[10][10],vis[10],j,n;
int sum=0;
int x=1;
int e=0;
void prims();

void main()
{
  int i;

  printf("enter the number of vertices\n");
  scanf("%d",&n);
  printf("enter the cost adjacency matrix\n");
  for(i=1;i<=n;i++)
  {
     for(j=1;j<=n;j++)
     {
          scanf("%d",&cost[i][j]);
     }
     vis[i]=0;
   }
   prims();
   printf("edges of spanning tree\n");
   for(i=1;i<=e;i++)
   {
        printf("%d,%d\t",et[i][0],et[i][1]);
   }
   printf("weight=%d\n",sum);
   getch();
```

```
}

void prims()
{
  int s,min,m,k,u,v;
  vt[x]=1;
  vis[x]=1;
  for(s=1;s<n;s++)
  {
     j=x;
     min=999;
     while(j>0)
     {
           k=vt[j];
           for(m=2;m<=n;m++)
           {
            if(vis[m]==0)
            {
                 if(cost[k][m]<min)
                 {
                   min=cost[k][m];
                   u=k;
                   v=m;
                 }
             }
          }
          j--;
   }
   vt[++x]=v;
   et[s][0]=u;
   et[s][1]=v;
   e++;
   vis[v]=1;
   sum=sum+min;
 }
}
```

OUTPUT:




```
enter the number of vertices
6
enter the cost adjacency matrix
0 2 3 999 999 999
2 0 5 3 4 999
3 5 0 999 4 999
999 3 999 0 2 3
999 4 4 2 0 5
999 999 999 3 5 0
edges of spanning tree
1,2     2,4     4,5     4,6     1,3     weight=13
```

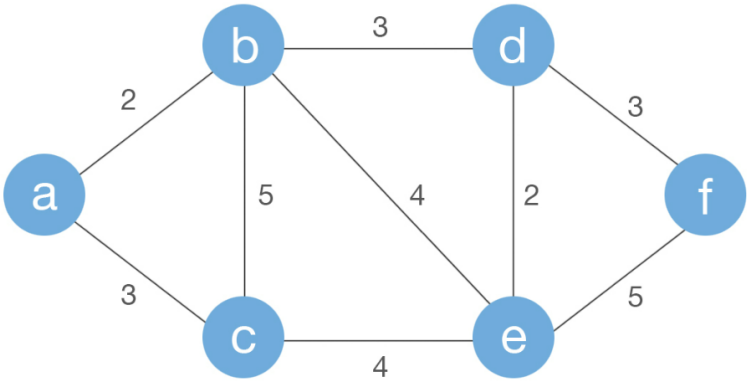
KRUSKAL'S ALGORITHM:

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

int find(int v,int parent[10])
{
  while(parent[v]!=v)
  {
    v=parent[v];
  }
  return v;
}

void union1(int i,int j,int parent[10])
{
  if(i<j)
    parent[j]=i;
  else
    parent[i]=j;
}

void kruskal(int n,int a[10][10])
{
  int count,k,min,sum,i,j,t[10][10],u,v,parent[10];
  count=0;
  k=0;
  sum=0;
  for(i=0;i<n;i++)
    parent[i]=i;
  while(count!=n-1)
  {
    min=999;
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {

          if(a[i][j]<min && a[i][j]!=0)
          {
              min=a[i][j];
              u=i;
```

```
                v=j;
            }
          }
      }
      i=find(u,parent);
      j=find(v,parent);
      if(i!=j)
      {
          union1(i,j,parent);
          t[k][0]=u;
          t[k][1]=v;
          k++;
          count++;
          sum=sum+a[u][v];
      }
      a[u][v]=a[v][u]=999;
    }
    if(count==n-1)
    {
      printf("spanning tree\n");
      for(i=0;i<n-1;i++)
      {
          printf("%d %d\n",t[i][0],t[i][1]);
      }
      printf("cost of spanning tree=%d\n",sum);
    }
    else
      printf("spanning tree does not exist\n");
  }


void main()
{
   int n,i,j,a[10][10];

   printf("enter the number of nodes\n");
   scanf("%d",&n);
   printf("enter the adjacency matrix\n");
   for(i=0;i<n;i++)
     for(j=0;j<n;j++)
```

```
        scanf("%d",&a[i][j]);
  kruskal(n,a);
  getch();

}
```

OUTPUT-



```
enter the number of nodes
6
enter the adjacency matrix
0 2 3 999 999 999
2 0 5 3 4 999
3 5 0 999 4 999
999 3 999 0 2 3
999 4  4 2 0 5
999 999 999 3 5 0
spanning tree
0 1
3 4
0 2
1 3
3 5
cost of spanning tree=13
```

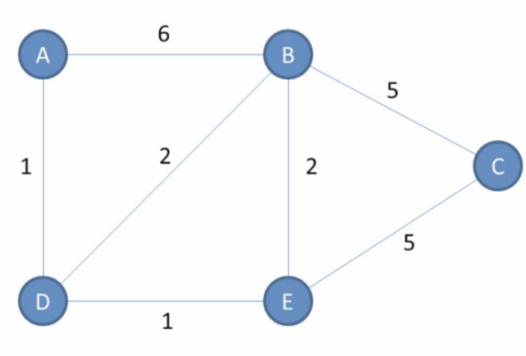
PROGRAM -10

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

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

void dijkstra(int n,int cost[10][10],int src)
{
   int i,j,u,dis[10],vis[10],min;
   for(i=1;i<=n;i++)
   {
         dis[i]=cost[src][i];
         vis[i]=0;
    }
    vis[src]=1;
    for(i=1;i<=n;i++)
    {
         min=999;
         for(j=1;j<=n;j++)
         {
           if(vis[j]==0 && dis[j]<min)
           {
             min=dis[j];
             u=j;
            }
         }
         vis[u]=1;
         for(j=1;j<=n;j++)
         {
           if(vis[j]==0 && dis[u]+cost[u][j]<dis[j])
           {
              dis[j]=dis[u]+cost[u][j];
            }
         }
   }

 printf("shortest path\n");
```

```
 for(i=1;i<=n;i++)
   printf("%d->%d=%d\n",src,i,dis[i]);
}


void main()
{
   int src,j,cost[10][10],n,i;

   printf("enter the number of vertices\n");
   scanf("%d",&n);
   printf("enter the cost adjacency matrix\n");
   for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
          scanf("%d",&cost[i][j]);
   printf("enter the source vertex\n");
   scanf("%d",&src);
   dijkstra(n,cost,src);
   getch();
}
```

OUTPUT-




```
enter the number of vertices
5
enter the cost adjacency matrix
0 6 999 1 999
6 0 5   2 2
999 5 0 999 5
1 2 999 0 1
999 2 5 1 0
enter the source vertex
1
shortest path
1->1=0
1->2=3
1->3=7
1->4=1
1->5=2
```

PROGRAM -11

Implement “N-Queens Problem” using Backtracking.

```
#include<stdio.h>
#include<math.h>

int board[20],count;

int main()
{
int n,i,j;
void queen(int row,int n);
printf("Enter number of queens:");
scanf("%d",&n);
queen(1,n);
return 0;
}

void print(int n)
{
int i,j;
printf("\n\nSolution %d:\n\n",++count);

for(i=1;i<=n;++i)
  printf("\t%d",i);

for(i=1;i<=n;++i)
{
  printf("\n\n%d",i);
  for(j=1;j<=n;++j)
  {
   if(board[i]==j)
    printf("\tQ");
   else
    printf("\t*");
  }
}
}
```

```
int place(int row,int column)
{
int i;
for(i=1;i<=row-1;++i)
{
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
}

return 1;
}

void queen(int row,int n)
{
int column;
for(column=1;column<=n;++column)
{
  if(place(row,column))
  {
   board[row]=column;
   if(row==n)
    print(n);
   else
    queen(row+1,n);
  }
}
}
```

OUTPUT:

```
 - N Queens Problem Using Backtracking -

Enter number of Queens:4


Solution 1:

        1       2       3       4

1       -       Q       -       -

2       -       -       -       Q

3       Q       -       -       -

4       -       -       Q       -

Solution 2:

        1       2       3       4

1       -       -       Q       -

2       Q       -       -       -

3       -       -       -       Q

4       -       Q       -       -
```