ADA LAB RECORD-1

DIVYASHREE K
1BM19CS054
CSE-4A

Question 1: Write a recursive program to a. Solve Towers-of-Hanoi problem b. To find GCD

CODE:

```
#include<stdio.h>

void merge(int b[],int c[],int a[],int nOfB,int nOfC){
    int i=0,j=0,k=0;
    // printf("Printing the Divided array B\n");
    // for(i=0;i<nOfB;i++){
    //     printf("%d\t",b[i]);
```

```
    // }
    // printf("\n");
    // printf("Printing the Divided array C\n");
    // for(i=0;i<nOfC;i++){
    //     printf("%d\t",c[i]);
    // }
    // printf("\n");
    // i=0;j=0;k=0;
    while(i<nOfB&&j<nOfC){
        if(b[i]<=c[j]){
            a[k]=b[i];
            i++;
        }else{
            a[k]=c[j];
```

```
            j++;
        }
        k++;
    }
    //printf("Values of i=%d and
j=%d\n",i,j);
    if(i==nOfB){
        for(j;j<nOfC;j++){
            a[k]=c[j];
            k++;
        }
    }
    else{
        for(i;i<nOfC;i++){
            a[k]=b[i];
```

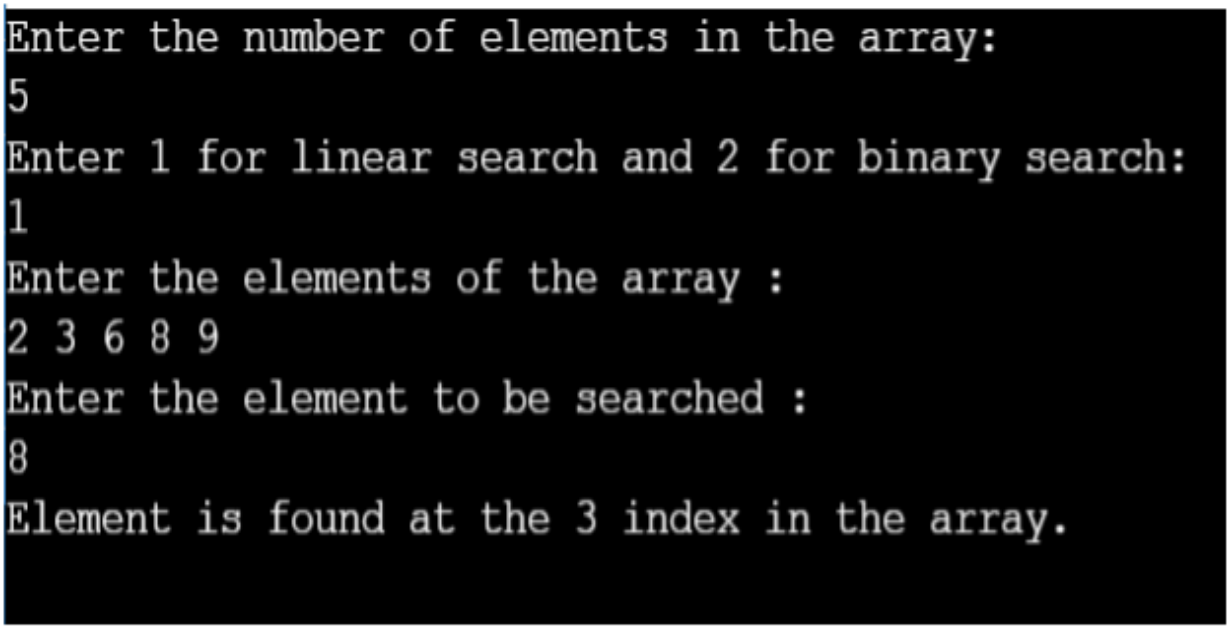
```
            k++;
        }
    }
    // printf("Printing the Divided array A\n");
    // for(i=0;i<k;i++){
    //     printf("%d\t",a[i]);
    // }
    // printf("\n");
}
void inputArray(int a[],int n){
    int i;
    for(i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
```

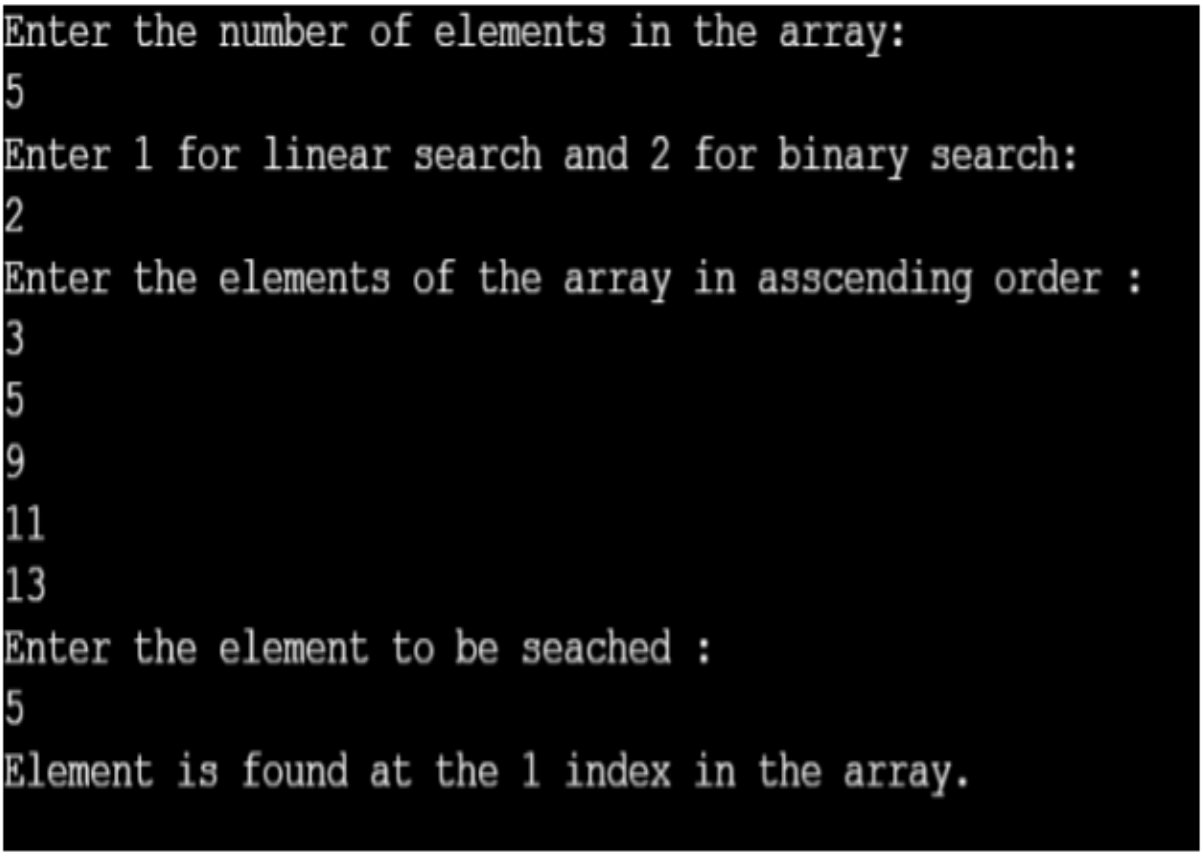
```
}
int main(){
    int n1,n2,i,n,median;
    printf("Enter the number of
elements in array 1\n");
    scanf("%d",&n1);
    printf("Enter the sort array 1\n");
    int b[n1];
    inputArray(b,n1);
    printf("Enter the number of
elements in array 2\n");
    scanf("%d",&n2);
    printf("Enter the sort array 2\n");
    int c[n2];
    inputArray(c,n2);
```

```
    n=n1+n2;
    int a[n];
    merge(b,c,a,n1,n2);
    for(i=0;i<n1+n2;i++){
        printf("%d\t",a[i]);
    }
    printf("\n");
    if(n%2==1){
        median=(n+1)/2;
        printf("Median is %d\n",a[median-
1]);
    }
    else {
        median=(n)/2;
```

```
        printf("Median is
%f\n",(float)(a[median-
1]+a[median])/2);
    }
    return 0;
}
```

OUTPUT:





Question 2: Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N

CODE: a)BINARY SEARCH:

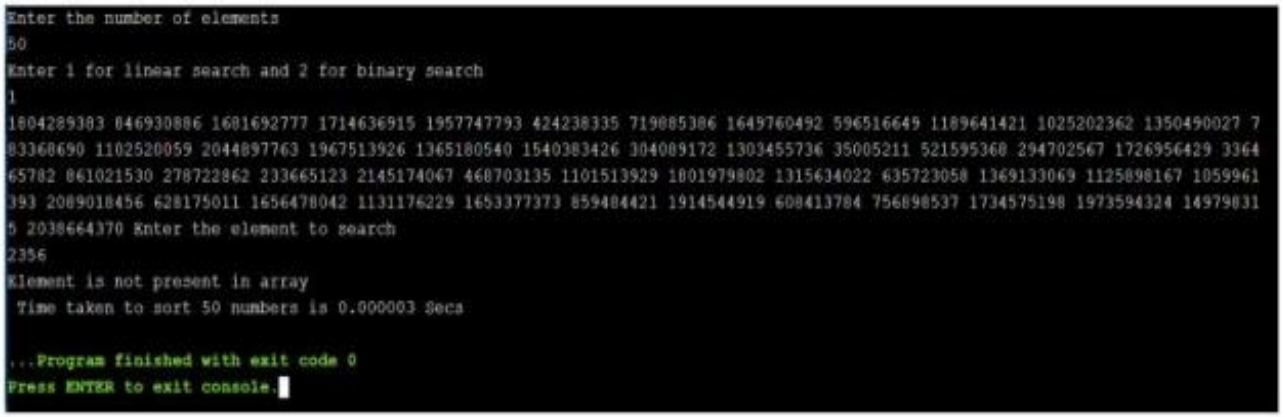
```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>
```

```
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)
{
    int n,i,key;
  float start,end;
    printf("Enter the number of elements\n");
   scanf("%d",&n);
   int arr[n];
   for(i=0;i<n;i++)
      arr[i]=rand();
   printf("Enter the element to search\n");
   scanf("%d",&key);
   start=clock();
```

```
    int result = binarySearch(arr, 0, n - 1, key);
   end=clock();
   if(result==-1)
      printf("Element is not present in array");
   else
      printf("Element is present at index %d",result);
   printf("\n Time taken to sort %d numbers is %f Secs",n, (((double)(end-start))/CLOCKS_PER_SEC));
    return 0;
}
```

OUTPUT:



CODE: b)LINEAR SEARCH

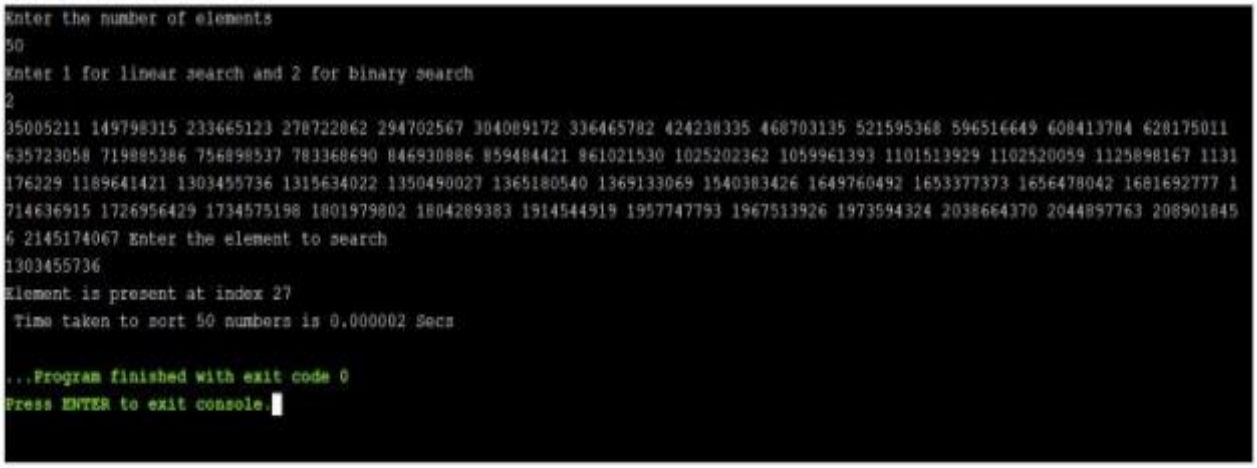
```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>
```

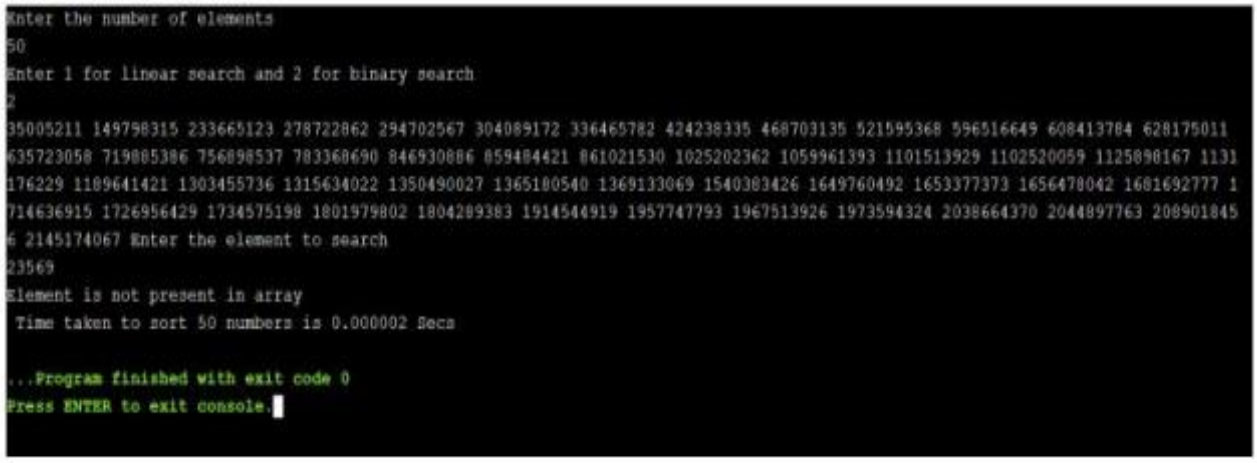
```
int linearSearch(int arr[], int l, int n, int
x)
{
    if (l >= n) {
        if (arr[l] == x)
            return l;
        return linearSearch(arr, l + 1, n,
x);
    }
    return -1;
}

int main(void)
{
    int n,i,key;
```

```
   float start,end;
     printf("Enter the number of elements\n");
    scanf("%d",&n);
    int arr[n];
    for(i=0;i<n;i++)
       arr[i]=rand();
    printf("Enter the element to search\n");
    scanf("%d",&key);
    start=clock();
     int result = linearSearch(arr, 0, n - 1, key);
    end=clock();
    if(result==-1)
```

```
        printf("Element is not present in
array");
    else
        printf("Element is present at index
%d",result);
    printf("\n Time taken to sort %d
numbers is %f Secs",n, (((double)(end-
start))/CLOCKS_PER_SEC));
     return 0;
}
```

OUTPUT:





Question 3: Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

CODE: #include <stdio.h>

#include <time.h>

#include <stdlib.h>

```
void delay(int milli_seconds)
{
    clock_t start_time = clock();
    while (clock() < start_time
+milli_seconds)
        ;
}

void selectionSort(int arr[], int n)
{
    int i, j, min,t;
    for (i = 0; i < n-1; i++)
    {
        min = i;
        for (j = i+1; j < n; j++)
```

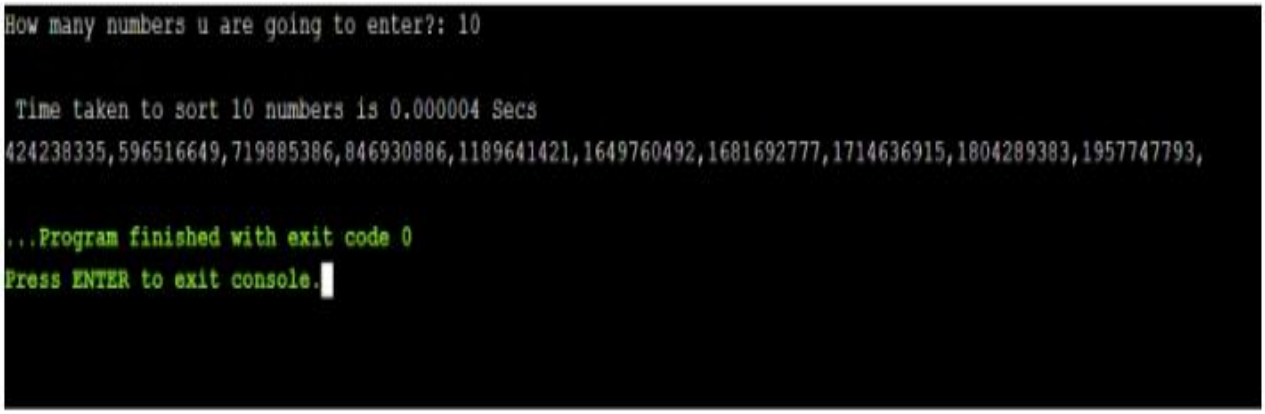
```
        {
                delay(10);
          if (arr[j] < arr[min])
             min = j;
        }
        t=arr[i];
        arr[i]=arr[min];
        arr[min]=t;
    }
}


void printSelection(int arr[], int size)
{
```

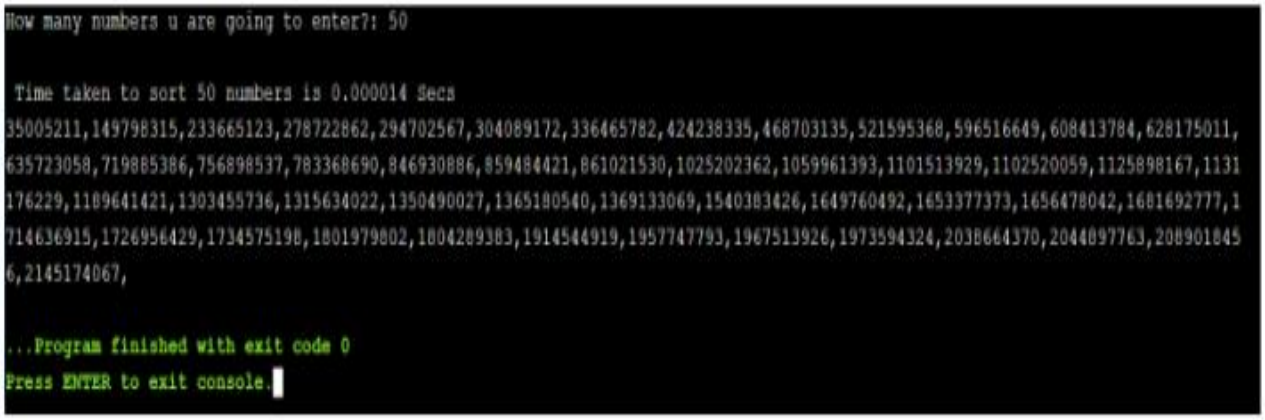
```
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}



int main()
{
     int n;
     clock_t start, end;
    double cpu_time_used;
     printf("Enter the size of the
array\n");
     scanf("%d",&n);
```

```
    int arr[n];
    printf("The elements of the
array:\n");
    for(int i=0;i<n;i++)
    arr[i]=rand();
    printSelection(arr, n);
    printf("\n");
    start = clock();
    selectionSort(arr, n);
    end = clock();
    cpu_time_used = ((double) (end -
start)) / CLOCKS_PER_SEC;
    printf("Sorted array: \n");
    printSelection(arr, n);
    printf("\n");
```

```
    printf("TIME FOR FUNCTION
EXECUTION is %f", cpu_time_used);

    return 0;

}
```

OUTPUT:





Question 4: 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.

CODE:

a)BFS:

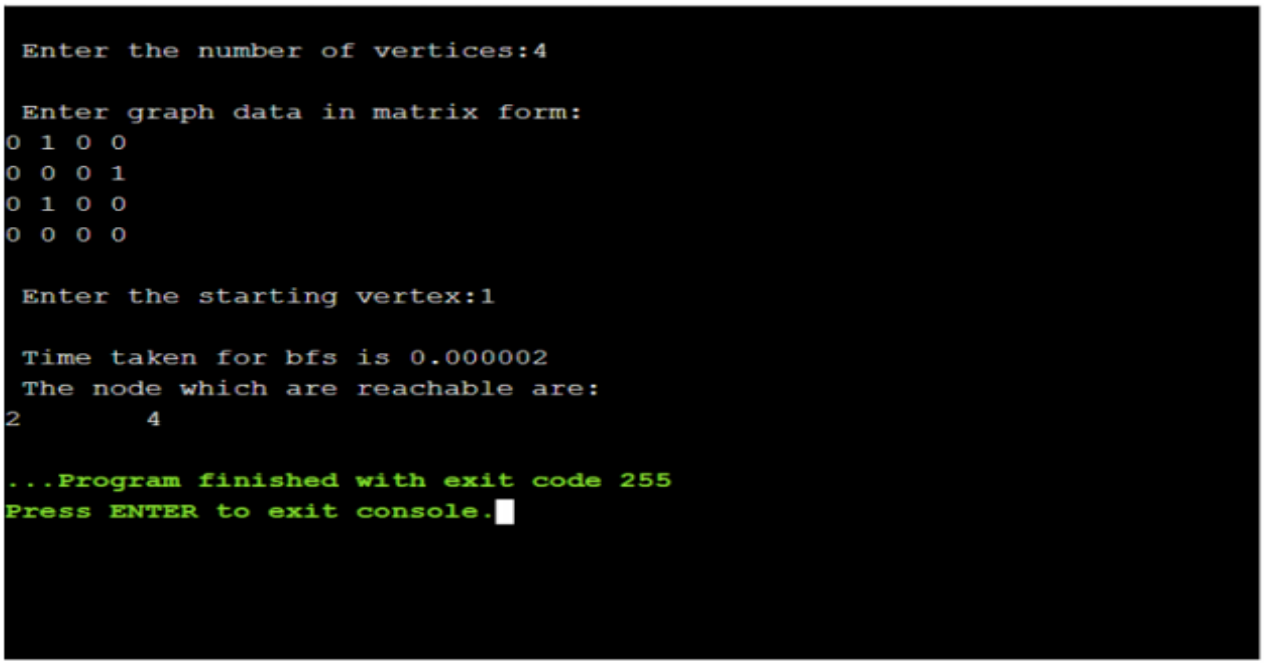
```
#include<stdio.h>
int a[10][10],vis[10],n;

void bfs(int v)
{
    int q[10],f=1,r=1,u,i;
    q[r]=v;
    vis[v]=1;
    while(f<=r)
    {
        u=q[f];
        printf("%d  ",u);
        for(i=1;i<=n;i++)
```

```
        {
            if(a[u][i]==1&&vis[i]==0)
            {
                vis[i]=1;
                r=r+1;
                q[r]=i;



            }
        }
        f=f+1;
    }
}
```

```
void main()
{
    int i,j,src;

    printf("enter the no of vertices\n");
    scanf("%d",&n);
    printf("enter the adjacency\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&a[i][j]);
        }
        vis[i]=0;
```

```
    }


    printf("enter the source vertex\n");
    scanf("%d",&src);
    printf("nodes reachable from source vertex are\n");
    bfs(src);
}
```

OUTPUT:a)



b)DFS:

```
#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 the total number of
vertices:");
    scanf("%d",&n);
    for (i=1;i<=n;i++) {
        reach[i]=0;
        for (j=1;j<=n;j++)
          a[i][j]=0;
    }
    printf("\n Enter the adjacency
matrix:\n");
    for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
       scanf("%d",&a[i][j]);
    dfs(1);
```

```
    printf("\n");
    for (i=1;i<=n;i++) {
        if(reach[i])
          count++;
    }
    if(count==n)
      printf("\n Graph is connected");
else
      printf("\n Graph is not
connected");
}
```

OUTPUT:b)

```
 Enter number of vertices:4

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

 1->2
 2->4
 Time taken for dfs is 0.000018

 Graph is not connected

...Program finished with exit code 255
Press ENTER to exit console.
```

Question 5: Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.

CODE:

```
#include <stdio.h>

#include <time.h>

#include <stdlib.h>
```

```
void delay(int milli_seconds)
{
    clock_t start_time = clock();
    while (clock() < start_time
+milli_seconds)
        ;
}

void insertionSort(int arr[], int n)
{
    int i, j, min,t;
    for (i = 0; i < n-1; i++)
    {
        min = i;
        for (j = i+1; j < n; j++)
```

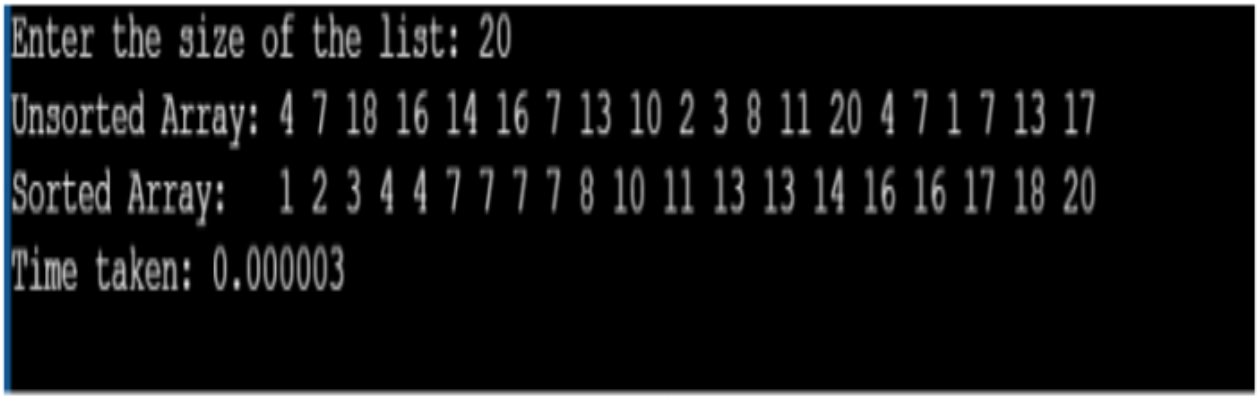
```
        {
                delay(10);

          if (arr[j] < arr[min])
             min = j;
        }
        t=arr[i];
        arr[i]=arr[min];
        arr[min]=t;
    }
}


void printArray(int arr[], int size)
```

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



int main()
{
     int n;
     clock_t start, end;
    double cpu_time_used;
     printf("Enter the size of the
array\n");
```

```
  scanf("%d",&n);
    int arr[n];
    printf("The elements of the
array:\n");
    for(int i=0;i<n;i++)
    arr[i]=rand();
    printArray(arr, n);
    printf("\n");
    start = clock();
    insertionSort(arr, n);
    end = clock();
    cpu_time_used = ((double) (end -
start)) / CLOCKS_PER_SEC;
    printf("Sorted array: \n");
    printArray(arr, n);
```

```
    printf("\n");
    printf("TIME FOR FUNCTION
EXECUTION is %f", cpu_time_used);
    return 0;
}
```

OUTPUT:



Question 6: Write program to obtain the Topological ordering of vertices in a given digraph

CODE: #include <stdio.h>

```
int main(){
    int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

    printf("Enter the no of vertices:\n");
```

```
    scanf("%d",&n);

    printf("Enter the adjacency matrix:\n");
    for(i=0;i<n;i++){
            printf("Enter row %d\n",i+1);
            for(j=0;j<n;j++)
                    scanf("%d",&a[i][j]);
    }

    for(i=0;i<n;i++){
     indeg[i]=0;
     flag[i]=0;
}

for(i=0;i<n;i++)
     for(j=0;j<n;j++)
         indeg[i]=indeg[i]+a[j][i];
```

```
printf("\nThe topological order is:");

while(count<n){
    for(k=0;k<n;k++){
        if((indeg[k]==0) && (flag[k]==0)){
            printf("%d ",(k+1));
            flag [k]=1;
        }

        for(i=0;i<n;i++){
            if(a[i][k]==1)
                indeg[k]--;
        }
    }

    count++;
}
```

```
    return 0;
}
```

OUTPUT:

Question 7: Implement Johnson Trotter algorithm to generate permutations

CODE:

```
#include<stdio.h>


int LEFT_TO_RIGHT = 1;
int RIGHT_TO_LEFT = 0;


int searchArr(int a[], int n, int mobile)
{
    for (int 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] == 0 && i!=0)
        {
            if (a[i] > a[i-1] && a[i] > mobile_prev)
            {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
```

```
        if (dir[a[i]-1] == 1 && 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 swap(int a[], int pos_1, int pos_2)
```

```
{
    int temp = a[pos_1];
    a[pos_1] = a[pos_2];
    a[pos_2] = temp;
}


int 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] ==  0)
       swap(a, pos-1, pos-2);

    else if (dir[a[pos - 1] - 1] == 1)
       swap(a, pos, pos-1);
```

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

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

```
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] =  0;


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


int main()
```

```
{
  int n;
  printf("Enter the value\n");
  scanf("%d",&n);
    printPermutation(n);
    return 0;
}
```

OUTPUT:

Question 8: 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.

CODE:

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

void merge(int arr[], int l, int m, int r)
{
```

```
int i, j, k;
int n1 = m - l + 1;
int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)
    L[i] = arr[l + i];

for (j = 0; j < n2; j++)
    R[j] = arr[m + 1 + j];

i = 0;
j = 0;
k = l;

while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
```

```
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
```

```
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{

    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void printArray(int A[], int size)
{
```

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

int main()
{
    int arr_size,i;
    clock_t start, end;
    printf("enter the array size\n");
    scanf("%d",&arr_size);
    int arr[arr_size];
    for(i=0;i<arr_size;i++)
    {
        arr[i]=rand()%1000;
    }
    printf("Given array is \n");
```

```
    printArray(arr, arr_size);
    start=clock();
    mergeSort(arr, 0, arr_size - 1);
    for(i=0;i<8000000;i++);
    end=clock();
    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    printf("\n Time taken to sort %d numbers is %f
Secs",arr_size, (((double)(end-
start))/CLOCKS_PER_SEC));
    return 0;
}
```

OUTPUT:

```
Enter the size of the list: 10
Unsorted Array: 4 7 8 6 4 6 7 3 10 2
Sorted Array:   2 3 4 4 6 6 7 7 8 10
Time taken: 0.000004
```

```
Enter the size of the list: 30
Unsorted Array: 14 17 28 26 24 26 17 13 10 2 3 8 21 20 24 17 1 7 23 17 12 9 28 10 3 21 3 14 8 26
Sorted Array:   1 2 3 3 3 7 8 8 9 10 10 12 13 14 14 17 17 17 17 20 21 21 23 24 24 26 26 26 28 28
Time taken: 0.000007
```

```
Enter the size of the list: 50
Unsorted Array: 34 37 28 16 44 36 37 43 50 22 13 28 41 10 14 27 41 27 23 37 12 19 18 30 33 31 13 24 18 36 30 3 23 9 20 18 44
 7 12 43 30 24 22 20 35 38 49 25 16 21
Sorted Array:   3 7 9 10 12 12 13 13 14 16 16 18 18 18 19 20 20 21 22 22 23 23 24 24 25 27 27 28 28 30 30 30 31 33 34 35 36
36 37 37 37 38 41 41 43 43 44 44 49 50
Time taken: 0.000011
```

```
Enter the size of the list: 100
Unsorted Array: 84 87 78 16 94 36 87 93 50 22 63 28 91 60 64 27 41 27 73 37 12 69 68 30 83 31 63 24 68 36 30 3 23 59 70 68 9
4 57 12 43 30 74 22 20 85 38 99 25 16 71 14 27 92 81 57 74 63 71 97 82 6 26 85 28 37 6 47 30 14 58 25 96 83 46 15 68 35 65 4
4 51 88 9 77 79 89 85 4 52 55 100 33 61 77 69 40 13 27 87 95 40
Sorted Array:   3 4 6 6 9 12 12 13 14 14 15 16 16 20 22 22 23 24 25 25 26 27 27 27 27 28 28 30 30 30 30 31 33 35 36 36 37 37
 38 40 40 41 43 44 46 47 50 51 52 55 57 57 58 59 60 61 63 63 63 64 65 68 68 68 68 69 69 70 71 71 73 74 74 77 77 78 79 81 82
83 83 84 85 85 85 87 87 87 88 89 91 92 93 94 94 95 96 97 99 100
Time taken: 0.000017
```

Question 9: Sort a given set of N integer elements using Quick Sort technique and compute its time taken

CODE: #include<stdio.h>

#include<time.h>

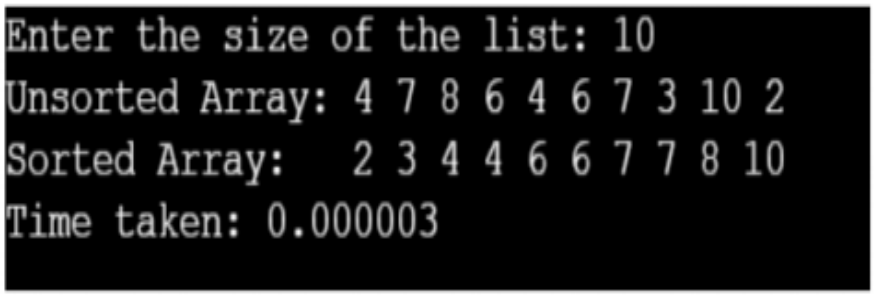
```
void quicksort(int number[25],int
first,int last){
   int i, j, pivot, temp;
   if(first<last){
      pivot=first;
      i=first;
      j=last;

      while(i<j){

while(number[i]<=number[pivot]&&i<l
ast)
            i++;
         while(number[j]>number[pivot])
            j--;
```

```
    if(i<j){
      temp=number[i];
      number[i]=number[j];
      number[j]=temp;
    }
  }

  temp=number[pivot];
  number[pivot]=number[j];
  number[j]=temp;
  quicksort(number,first,j-1);
  quicksort(number,j+1,last);

}
```

```
}


int main(){
   int i, count;
   clock_t start, end;
   printf("How many elements are u
going to enter?: ");
   scanf("%d",&count);
   int number[count];
   printf("Given array is \n");
   for(i=0;i<count;i++){
      number[i]=rand()%1000;
      printf("%d ", number[i]);
   }
   start=clock();
```

```
   quicksort(number,0,count-1);
   for(i=0;i<8000000;i++);
   end=clock();
   printf("\nOrder of Sorted elements:
\n");
   for(i=0;i<count;i++)
      printf("%d ",number[i]);
   printf("\n Time taken to sort %d
numbers is %f Secs",count,
(((double)(end-
start))/CLOCKS_PER_SEC));
   return 0;
}
```

OUTPUT:



Question 10: Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

CODE: #include<stdio.h>

void heapSort(int h[],int n){

```
int i,k,v,heap,j;
for(i=n/2;i>=1;i--){
    k=i;
    v=h[k];
    heap=0;
    while(heap==0 && 2*k<=n){
        j=2*k;
        if(j<n){
            if(h[j]<h[j+1])
                j=j+1;
        }
        if(v>=h[j]){
            heap=1;
        }else{
```

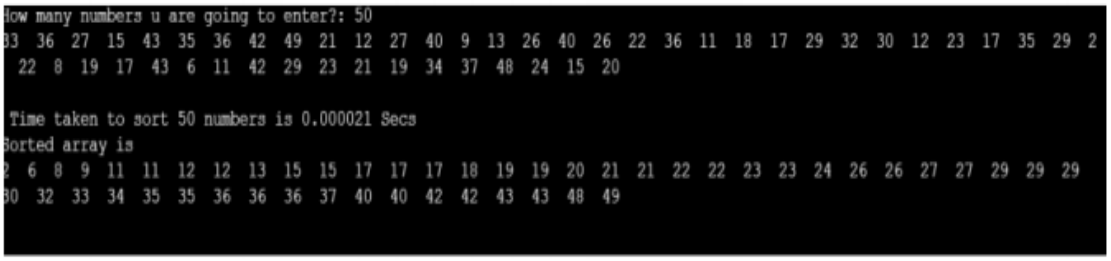
```
            h[k]=h[j];
            k=j;
        }
    }
    h[k]=v;
  }
}
int main(){
  int n,i;
  printf("Enter no of elements\n");
  scanf("%d",&n);
  int h[n];
  printf("Enter the heap in array representation\n");
  for(i=1;i<=n;i++)
```

```
        scanf("%d",&h[i]);
    heapSort(h,n);
    printf("Printing the sorted heap\n");
    for(i=1;i<=n;i++){
        printf("%d\t",h[i]);
        if(i%5==0)
            printf("\n");
    }
    return 0;
}
```

OUTPUT:

```
How many numbers u are going to enter?: 5
3  1  2  0  3

 Time taken to sort 5 numbers is 0.000004 Secs
Sorted array is
0  1  2  3  3
```

```
How many numbers u are going to enter?: 10
3  6  7  5  3  5  6  2  9  1

 Time taken to sort 10 numbers is 0.000003 Secs
Sorted array is
1  2  3  3  5  5  6  6  7  9
```



Question 11: Implement Warshall's algorithm using dynamic programming.

CODE: #include<stdio.h>

#include<stdlib.h>

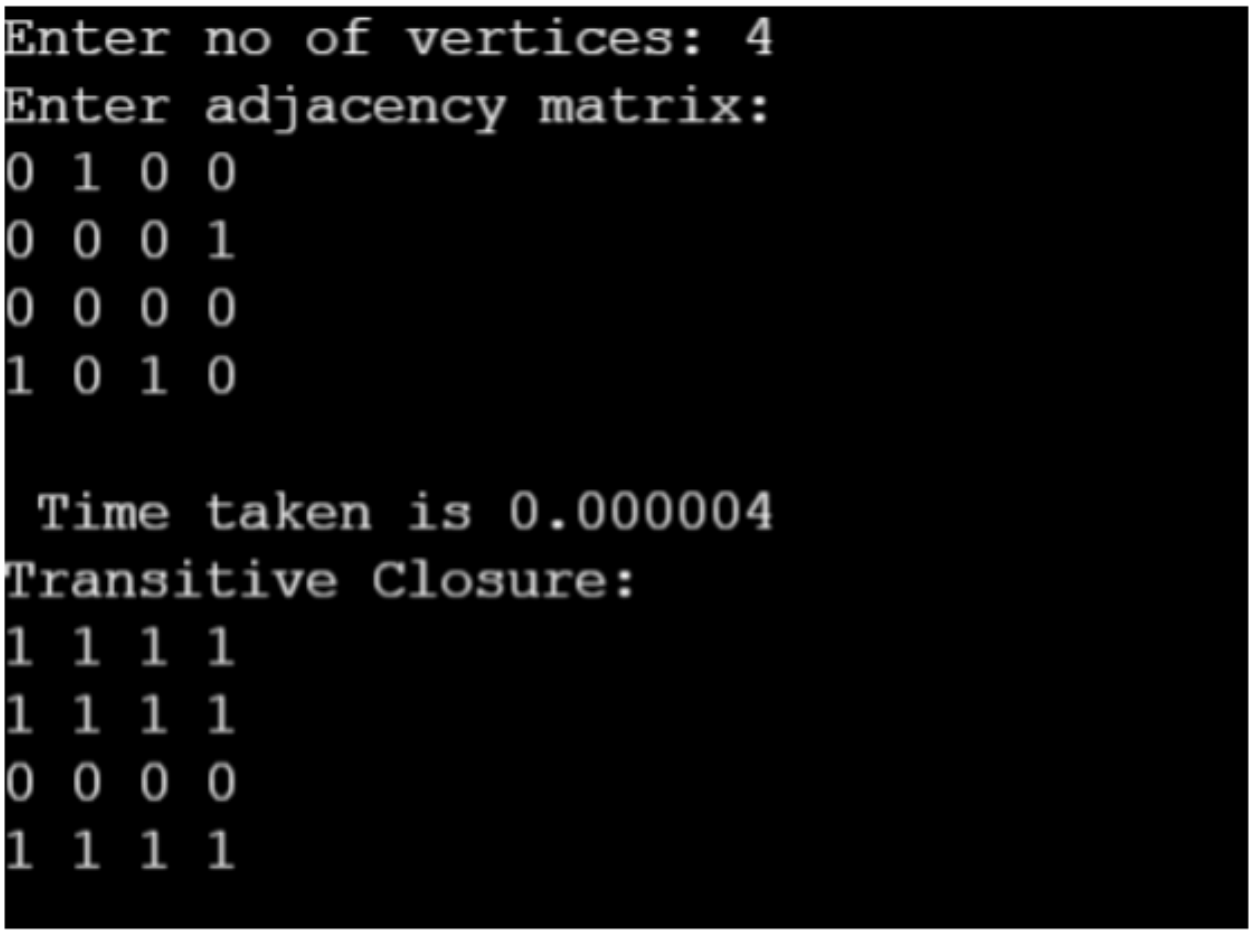
```
void transitiveClosure(int A[10][10],int
n)
{
   int path[n][n];
   int i,j,k;

   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         path[i][j] = A[i][j];
      }
   }

   for(k=0;k<n;k++)
```

```
    {
        for(i=0;i<n;i++)
        {
            for(j=0;j<n;j++)
            {
                if(path[i][j] != 1 && (path[i][k]
&& path[k][j]))
                    path[i][j] = 1;
            }
        }
    }

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
```

```
        {
            printf("\t%d",path[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int A[10][10];
    int i,j;
    int n;
    printf("\nEnter The Number Of Vertices : ");
    scanf("%d",&n);
```

```
    printf("\nEnter The Graph Data in
the form of Adacency Matrix : \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            scanf("%d",&A[i][j]);
        }
    }
    transitiveClosure(A,n);
    return 0;
}
```

OUTPUT:



Question 12: Implement 0/1 Knapsack problem using dynamic programming.

CODE: #include<stdio.h>

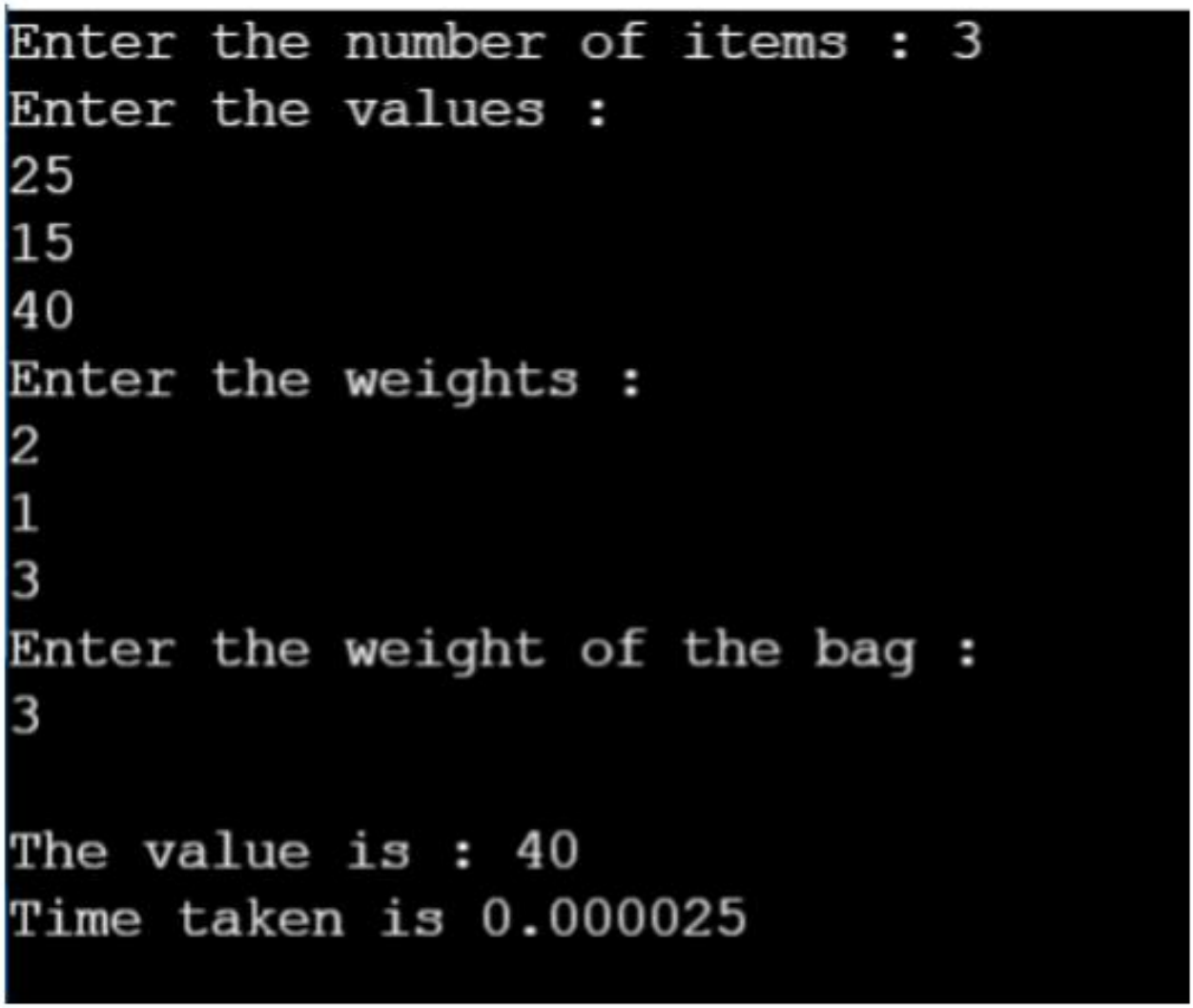
int max(int a, int b) { return (a > b)? a : b; }

```
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];
        }
    }

    return K[n][W];
}

int main()
{
    int i, n, val[20], wt[20], W;

    printf("Enter number of items:");
    scanf("%d", &n);
```

```
    printf("Enter value and weight of
items:\n");
    for(i = 0;i < n; ++i){
      scanf("%d%d", &val[i], &wt[i]);
    }

    printf("Enter size of knapsack:");
    scanf("%d", &W);

    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

OUTPUT:



Question 13: Implement All Pair Shortest paths problem using Floyd's algorithm.

CODE: #include<stdio.h>

int a[10][10],d[10][10],n;

```
void floyds();
int min(int ,int);
int main()
{
    printf("Enter no of vertices\n");
    scanf("%d",&n);
    printf("Enter cost adjacency matrix
:\n");
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
    }
    floyds();
    printf("Distance matrix :\n");
```

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

void floyds()
{
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
            d[i][j]=a[i][j];
```

```
    }
    for(int k=1;k<=n;k++)
    {
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=n;j++)
            {

    d[i][j]=min(d[i][j],d[i][k]+d[k][j]);
            }
        }
    }
}
int min(int a,int b)
{
```

```
    if(a<b)
        return a;
    else
        return b;
}
```

OUTPUT:

```
No of vertices: 4
Enter Weight matrix(-1 if there is no edge):
0 2 3 7
-1 0 1 -1
-1 -1 0 5
-1 -1 -1 0

 Time taken is 0.000004Distance matrix:
0   2   3   7
INF 0   1   6
INF INF 0   5
INF INF INF 0
```

Question 14: Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm

CODE:

```
#include<stdio.h>

int minKey(int key[], int mstSet[],int n)

{

    int min = 100, min_index;

  int v;

    for ( v = 0; v < n; v++)

        if (mstSet[v] == 0 && key[v] < min)

            min = key[v], min_index = v;


    return min_index;

}
```

```
int printMST(int parent[10], int
graph[10][10],int n)
{
    int i;
    printf("Edge \tWeight\n");
    for ( i = 1; i < n; i++)
        printf("%d - %d \t%d \n", parent[i],
i, graph[i][parent[i]]);
}

void primMST(int graph[10][10],int n)
{

    int parent[n];
    int key[n];
```

```
    int mstSet[n];
    int i,count,v,u;


    for (i = 0; i < n; i++)
        key[i] = 100, mstSet[i] = 0;


    key[0] = 0;
    parent[0] = -1;


    for ( count = 0; count < n - 1;
count++) {
```

```
        u = minKey(key, mstSet,n);


        mstSet[u] = 1;


        for (v = 0; v < n; v++)


            if (graph[u][v] && mstSet[v] == 0
&& graph[u][v] < key[v])

                parent[v] = u, key[v] =
graph[u][v];
    }


    printMST(parent, graph,n);
}
```

```
int main()
{
    int graph[10][10];
    int i,j,n;
    printf("Enter number of nodes\n");
    scanf("%d",&n);
    printf("Enter adjacency matrix\n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        scanf("%d",&graph[i][j]);
    }
    primMST(graph,n);
```

```
    return 0;
}
```

OUTPUT:

```
Enter number of nodes
5
Enter adjacency matrix
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
Edge     Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

Question 15: Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.

```
CODE: #include <stdio.h>
 #include <stdlib.h>
    int i,j,k,a,b,u,v,n,ne=1;
    int
min,mincost=0,cost[9][9],parent[9];
  int find(int i)
    {
     while(parent[i])
     i=parent[i];
     return i;
    }
    int uni(int i,int j)
    {
     if(i!=j)
     {
```

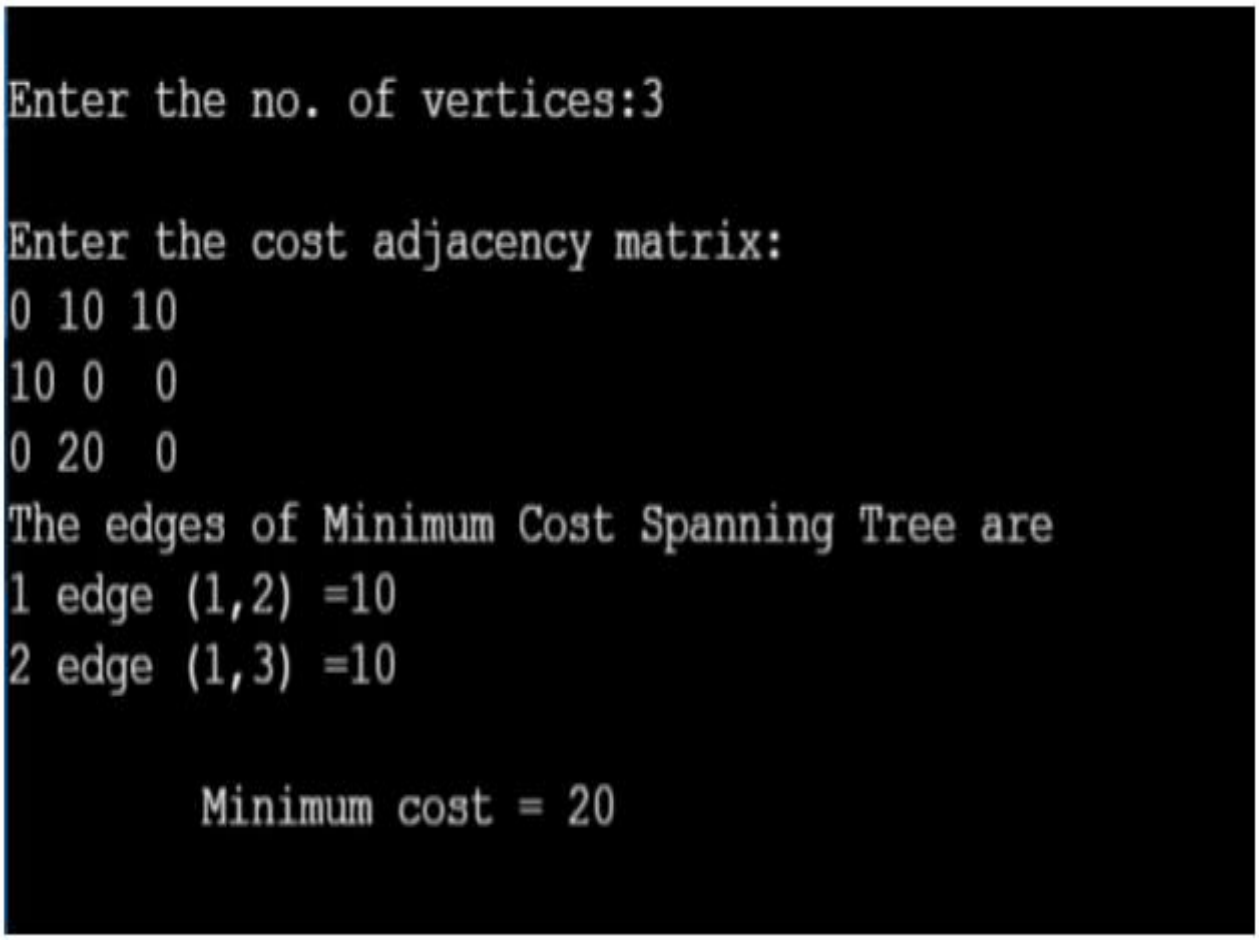
```
    parent[j]=i;
    return 1;
    }
    return 0;
   }
   void main()
   {
    printf("\nEnter the no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the cost adjacency
matrix:\n");
    for(i=1;i<=n;i++)
    {
    for(j=1;j<=n;j++)
    {
```

```
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
for(i=1,min=999;i<=n;i++)
{
for(j=1;j <= n;j++)
{
if(cost[i][j] < min)
{
```

```
    min=cost[i][j];
    a=u=i;
    b=v=j;
    }
    }
    }
    u=find(u);
    v=find(v);
    if(uni(u,v))
    {
    printf("%d edge (%d,%d)
=%d\n",ne++,a,b,min);
    mincost +=min;
    }
    cost[a][b]=cost[b][a]=999;
```

```
    }
    printf("\n\tMinimum cost =
%d\n",mincost);


   }
```

OUTPUT:



Question 16: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm

CODE:

```
#include <stdio.h>


int minDistance(int dist[], int
sptSet[],int V)
{
   int min = 999, min_index;
  int v;
   for ( v = 0; v < V; v++)
       if (sptSet[v] == 0 && dist[v] <= min)
           min = dist[v], min_index = v;


   return min_index;
}
```

```
int printSolution(int dist[],int V )
{ int i;
    printf("Vertex \t\t Distance from Source\n");
    for (i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[10][10], int src,int V)
{
    int dist[V];
  int i,count,u,v;
    int sptSet[V];
    for ( i = 0; i < V; i++)
        dist[i] = 999, sptSet[i] = 0;
```

```
    dist[src] = 0;

    for ( count = 0; count < V - 1;
count++) {

        u = minDistance(dist, sptSet,V);
        sptSet[u] = 1;

        for (v = 0; v < V; v++)

            if (!sptSet[v] && graph[u][v] &&
dist[u] != 999
                && dist[u] + graph[u][v] <
dist[v])
```

```
                dist[v] = dist[u] + graph[u][v];
    }

    printSolution(dist,V);
}

int main()
{
    int i,j,V;

    int graph[10][10];
    printf("Enter number of vertices\n");
    scanf("%d",&V);
    printf("Enter adjacency matrix\n");
```

```
    for(i=0;i<V;i++)
    {
        for(j=0;j<V;j++)
    scanf("%d",&graph[i][j]);
    }

    dijkstra(graph, 0,V);

    return 0;
}
```

OUTPUT:

```
Enter the number of vertices : 5
Enter the matrix :
0 0 0 7 0
3 0 4 0 0
0 0 0 0 6
0 2 5 0 0
0 0 0 4 0
Vertex   Distance        Path
0 -> 1           9               0 3 1
0 -> 2           12              0 3 2
0 -> 3           7               0 3
0 -> 4           18              0 3 2 4
```

Question 17: Implement “Sum of Subsets” using Backtracking. “Sum of Subsets” problem: Find a subset of a given set S = {s1,s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn’t have a solution.

CODE: #include<stdio.h>

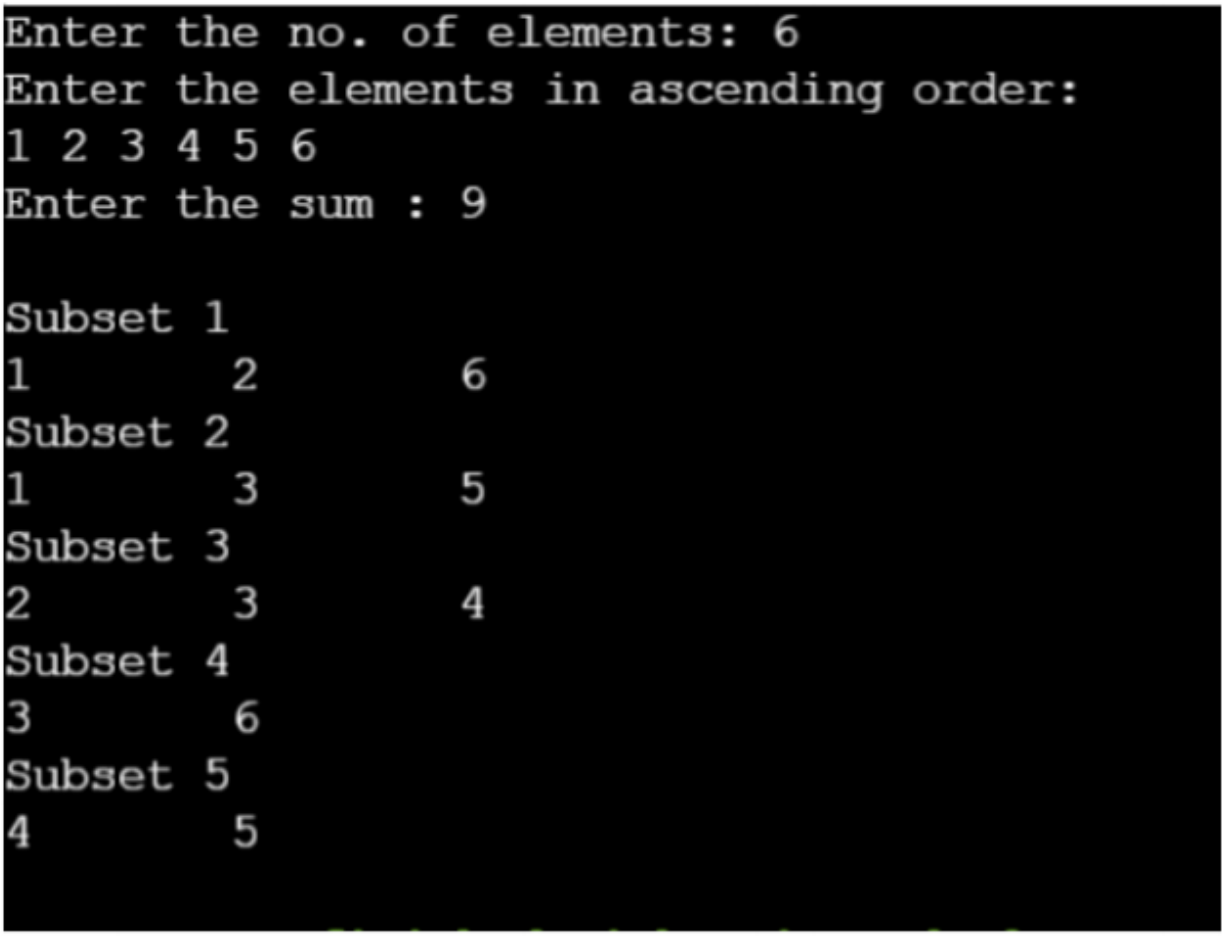
```
#include<stdlib.h>

int x[1000],w[1000],d,count=0;

void subset(int cs,int k,int r)
{
    int i;
    x[k]=1;
    if(cs+w[k]==d)
    {
        printf("\nSubset %d\n",++count);
        for(i=0;i<=k;i++)
        if(x[i]==1)
        printf("%d\t",w[i]);
```

```
    }
    else if(cs+w[k]+w[k+1]<=d)
        subset(cs+w[k],k+1,r-w[k]);
    if(cs+r-w[k]>=d && cs+w[k]<=d)
    {
        x[k]=0;
        subset(cs,k+1,r-w[k]);
    }
}

int main()
{
    int i,n,sum=0;
```

```
    printf("Enter the no. of elements: ");
    scanf("%d",&n);
    printf("Enter the elements in
ascending order:\n");
    for(i=0;i<n;i++)
        scanf("%d",&w[i]);
    printf("Enter the sum : ");
    scanf("%d",&d);
    for(i=0;i<n;i++)
        sum=sum+w[i];
    if(sum<d)
    {
        printf("No solution\n");
        exit(0);
    }
```

```
    subset(0,0,sum);
    if(count==0)
    {
        printf("No solution\n");
        exit(0);
    }
    return 0;
}
```

OUTPUT:



Question 18: Implement "N-Queens Problem" using Backtracking.

CODE:

```
#include<stdio.h>

#include<stdlib.h>

#include<math.h>


void nqueens(int);
```

```
int place(int[],int);
void printsolution(int,int[]);

void main()
{
   int n;
   printf("Enter the no.of queens: ");
   scanf("%d",&n);
   nqueens(n);
}

void nqueens(int n)
{
   int x[10],count=0,k=1;
```

```
    x[k]=0;
    while(k!=0)
    {
        x[k]=x[k]+1;
        while(x[k]<=n&&(!place(x,k)))
        x[k]=x[k]+1;
        if(x[k]<=n)
        {
            if(k==n)
            {
                count++;
                printf("\nSolution
%d\n",count);
                printsolution(n,x);
            }
```

```
            else
            {
                k++;
                x[k]=0;
            }
        }
        else
        {
            k--; //backtracking
        }
    }
    return;
}
```

```
int place(int x[],int k)
{
    int i;
    for(i=1;i<k;i++)
        if(x[i]==x[k]||(abs(x[i]-
x[k]))==abs(i-k))
            return 0;
    return 1;
}

void printsolution(int n,int x[])
{
    int i,j;
    char c[10][10];
    for(i=1;i<=n;i++)
```

```
{
    for(j=1;j<=n;j++)
    c[i][j]='X';
}
for(i=1;i<=n;i++)
    c[i][x[i]]='Q';
for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++)
    {
        printf("%c\t",c[i][j]);
    }
    printf("\n");
}
```

}

OUTPUT: