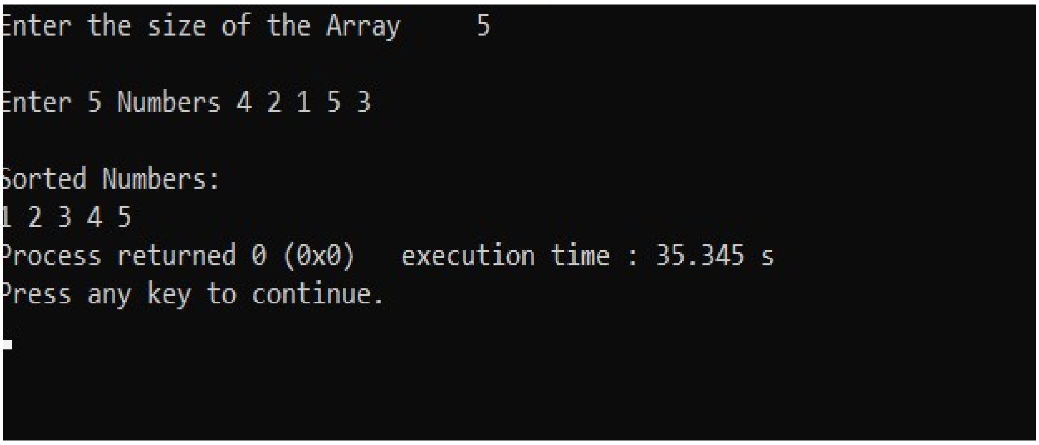
Practical – 1

Write a Program for sorting array in ascending order using bubble sort.

```
#include <stdio.h>
int main()
{
   int  n, j, i, swap;
printf("Enter the size of Array\n");
scanf("%d", &n);
   int array[n];
printf("Enter %d integers\n", n);
   for (i= 0; i< n; i++)
   {
scanf("%d", &array[i]);
   }
   for (i = 0 ; i< n - 1; i++)
   {
      for (j = 0 ; j < n - i- 1; j++)
      {
         if (array[j] > array[j+1])
         {
            swap       = array[j];
            array[j]   = array[j+1];
            array[j+1] = swap;
         }
      }
   }

printf("Sorted Numbers:\n");

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

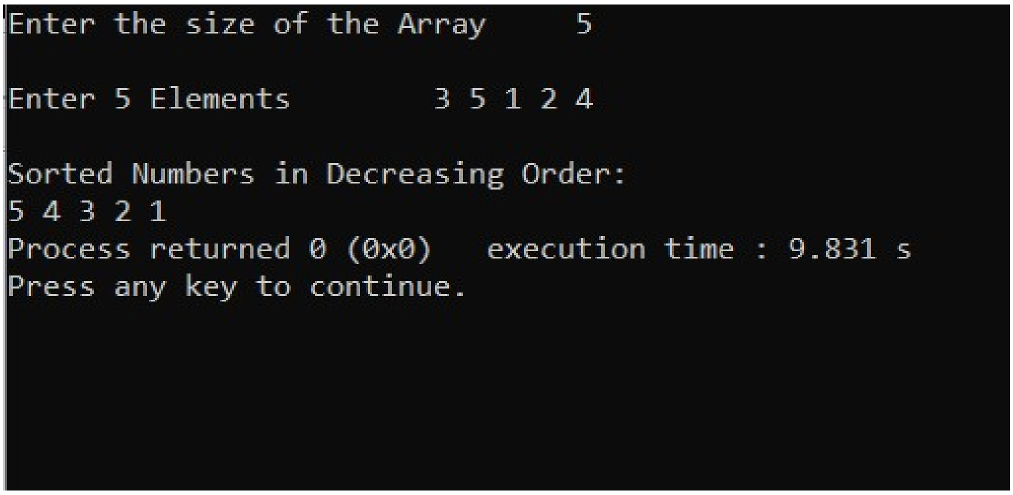


```
Enter the size of the Array     5

Enter 5 Numbers 4 2 1 5 3

Sorted Numbers:
1 2 3 4 5
Process returned 0 (0x0)   execution time : 35.345 s
Press any key to continue.
```

Practical - 2

Write a Program for sorting array in descending order using insertion sort.

```
#include <stdio.h>
#define MAX 100
int main()
{
   int arr[MAX],limit;
   int i,j,temp;
printf("Enter total number of elements: ");
scanf("%d",&limit);
printf("Enter %d  elements: \n",limit);
   for(i=0; i<limit; i++)
   {
printf("%d",i+1);
scanf("%d",&arr[i]);
   }
   for(i=1; i<(limit); i++)
   {
      j=i;
      while(j>0 &&arr[j]<arr[j-1])
      {
         temp=arr[j];
arr[j]=arr[j-1];
arr[j-1]=temp;
         j--;
      }
   }
printf("Array elements in Descending Order:\n");
   for(i=0; i<limit; i++)
printf("%d ",arr[i]);

printf("\n");
   return 0;}
```

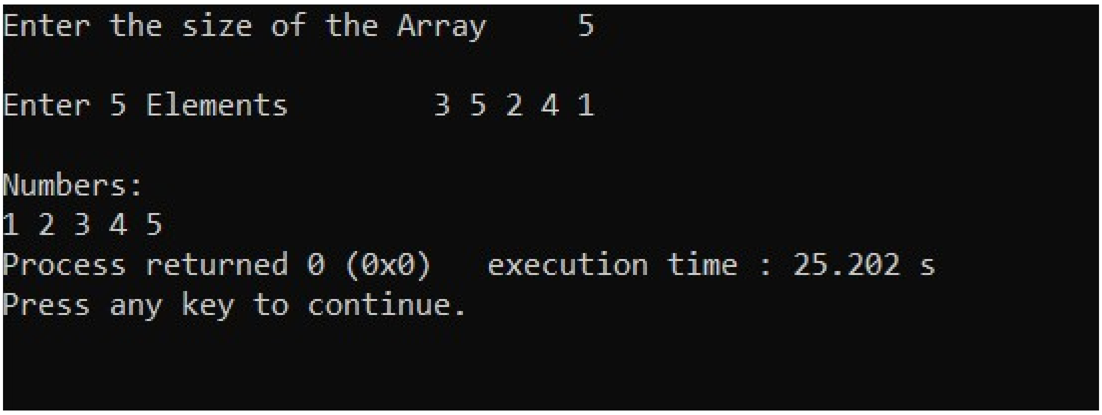


```
Enter the size of the Array     5

Enter 5 Elements        3 5 1 2 4

Sorted Numbers in Decreasing Order:
5 4 3 2 1
Process returned 0 (0x0)   execution time : 9.831 s
Press any key to continue.
```

Practical - 3

Write a Program for sorting array in ascending order using selection sort.

```
#include <stdio.h>
int main()
{
    int a[100], n, i, j, position, swap;
printf("Enter number of elements\n");
scanf("%d", &n);
printf("Enter %d Numbers\n", n);
   for (i = 0; i< n; i++)
scanf("%d", &a[i]);
for(i = 0; i< n - 1; i++)
{
   position=i;
   for(j = i + 1; j < n; j++)
   {
     if(a[position] > a[j])
     position=j;
   }
   if(position != i)
   {
   swap=a[i];
   a[i]=a[position];
   a[position]=swap;
   }
}
printf("Sorted Array:\n");
for(i = 0; i< n; i++)
printf("%dn", a[i]);
return 0;
}
```



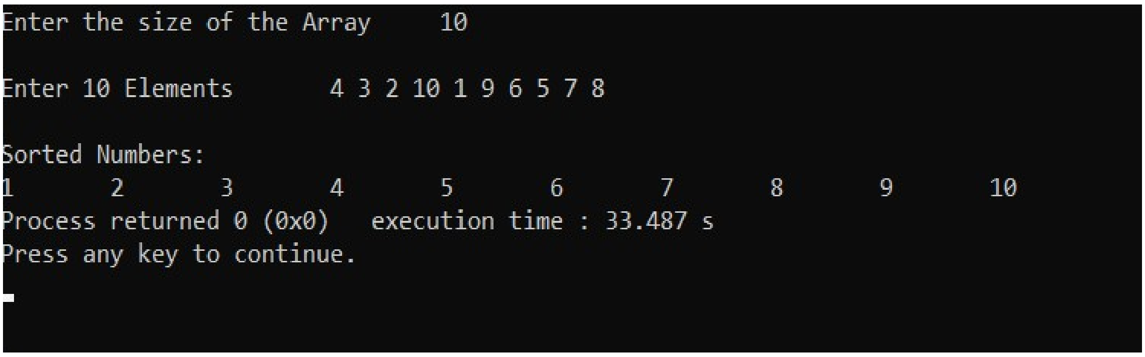
```
Enter the size of the Array     5

Enter 5 Elements        3 5 2 4 1

Numbers:
1 2 3 4 5
Process returned 0 (0x0)   execution time : 25.202 s
Press any key to continue.
```

Practical - 4

Write a C Program for Merge Sort.

```
#include<stdio.h>
void merge_sort(int [],int,int);
void merge(int [], int, int, int);
int main()
{
   int i,n,A[15];
printf("Enter the size of the Array\t");
scanf("%d",&n);
printf("\nEnter %d Elements\t",n);
   for(i=0; i<n; i++)
   {
scanf("%d",&A[i]);
   }
merge_sort(A,0,n-1);
printf("\nSorted Numbers:\n");
   for(i=0; i<n; i++)
   {
printf("%d\t",A[i]);
   }
}
void merge_sort(int A[], int beg, int end)
{
   int mid;
   if(beg<end)
   {
      mid=(beg+end)/2;
merge_sort(A,beg,mid);
merge_sort(A,mid+1,end);
      merge(A,beg,mid,end);
   }
}
void merge(int A[],int BEG, int MID, int END)
{
   int B[15],i,j,k;
i=BEG;
   j=MID+1;
   k=BEG;
   while(i<=MID && j<=END)
   {
      if(A[i]<=A[j])
      {
         B[k]=A[i];
i++;
      }
      else
```

```
        {
            B[k]=A[j];
j++;
        }
        k++;
    }
    if(i>MID)
    {
        while(j<=END)
        {
            B[k]=A[j];
j++;
            k++;
        }
    }
    else if(j>END)
    {
        while(i<=MID)
        {
            B[k]=A[i];
i++;
            k++;
        }
    }
    for(i=BEG; i<=END; i++)
        A[i]=B[i];
}
```



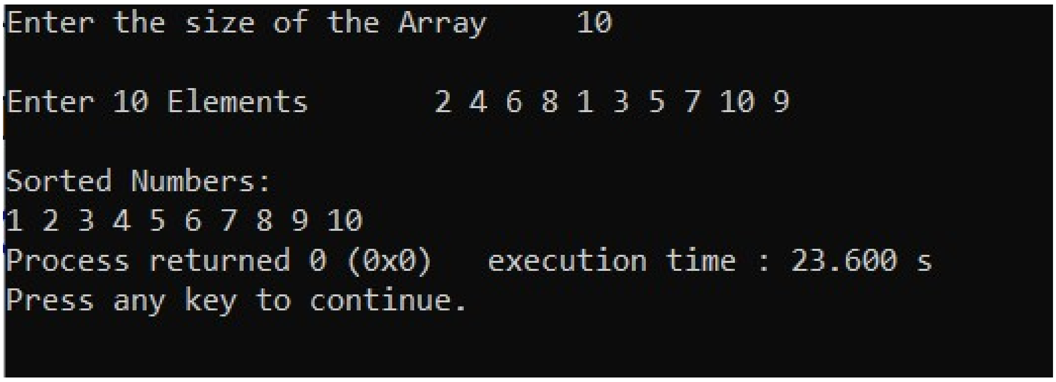
```
Enter the size of the Array     10

Enter 10 Elements       4 3 2 10 1 9 6 5 7 8

Sorted Numbers:
1       2       3       4       5       6       7       8       9       10
Process returned 0 (0x0)   execution time : 33.487 s
Press any key to continue.
```

Practical - 5

Write a C Program for Quick Sort.

```
#include<stdio.h>
int partitioning(int [], int, int);
void Quick_Sort(int [], int, int);
int main()
{
   int A[15],i,n;
printf("Enter the size of the Array\t");
scanf("%d",&n);
printf("\nEnter %d Elements\t",n);
   for(i=0; i<n; i++)
   {
scanf("%d",&A[i]);
   }
Quick_Sort(A,0,n-1);
printf("\nSorted Numbers:\n");
   for(i=0; i<n; i++)
   {
printf("%d ",A[i]);
   }
   return 0;
}
int partitioning(int A[], int beg,int end)
{
   int pivot,left,right,temp;
   pivot=A[beg];
   left=beg;
   right=end;
   while(left<right)
   {
      while(A[left]<=pivot)
      {
         left++;
      }
      while(A[right]>pivot)
      {
         right--;
      }
      if(left<right)
      {
         temp=A[left];
         A[left]=A[right];
         A[right]=temp;
      }
   }
   temp=A[beg];
```

```
    A[beg]=A[right];
    A[right]=temp;
    return right;
}

void Quick_Sort(int A[],int beg, int end)
{
    int loc;
    if(beg<end)
    {   loc=partitioning(A,beg,end);
Quick_Sort(A,beg,loc-1);
Quick_Sort(A,loc+1,end);    }}
```

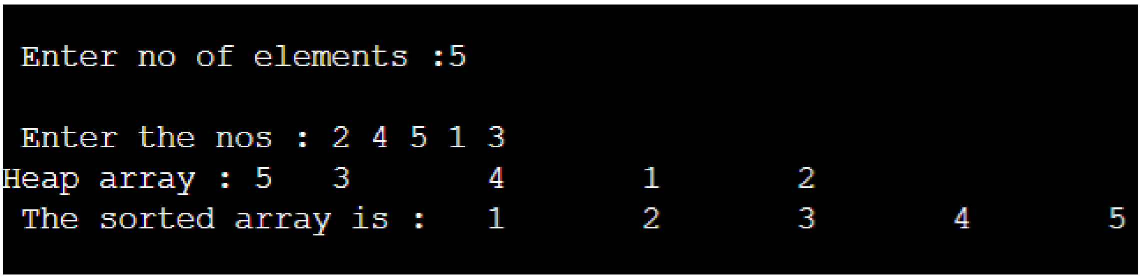


```
Enter the size of the Array     10

Enter 10 Elements       2 4 6 8 1 3 5 7 10 9

Sorted Numbers:
1 2 3 4 5 6 7 8 9 10
Process returned 0 (0x0)   execution time : 23.600 s
Press any key to continue.
```

Practical –6

Write a C program for heap sort.

```
#include <stdio.h>
void main()
{ int heap[20], no, i, j, c, root, temp;
printf("\n Enter no of elements :");
scanf("%d", &no);
printf("\n Enter the nos : ");
   for (i = 0; i< no; i++)
scanf("%d", &heap[i]);
   for (i = 1; i< no; i++)
   {    c = i;
      do
      { root = (c - 1) / 2;
         if (heap[root] < heap[c])
         { temp = heap[root];
            heap[root] = heap[c];
            heap[c] = temp;  }
         c = root;
      } while (c != 0);    }
printf("Heap array : ");
   for (i = 0; i< no; i++)
printf("%d\t ", heap[i]);
   for (j = no - 1; j >= 0; j--)
   {    temp = heap[0];
      heap[0] = heap[j];
      heap[j] = temp;
      root = 0;
      do
      {  c = 2 * root + 1;
         if ((heap[c] < heap[c + 1]) && c < j-1)
c++;
         if (heap[root]<heap[c] && c<j)
         {  temp = heap[root];
            heap[root] = heap[c];
            heap[c] = temp;   }
         root = c;
      } while (c < j);    }
printf("\n The sorted array is : ");
   for (i = 0; i< no; i++)
printf("\t %d", heap[i]);}
```

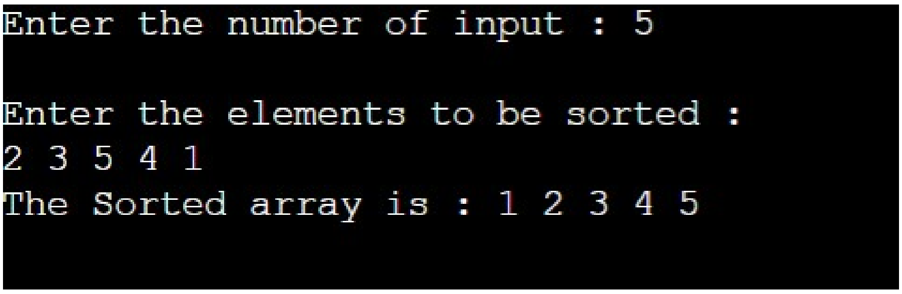


```
 Enter no of elements :5

 Enter the nos : 2 4 5 1 3
Heap array : 5   3       4       1       2
 The sorted array is :   1       2       3       4       5
```

Practical –7

WAP to implement Counting sort.

```
#include <stdio.h>
void counting_sort(int A[], int k, int n)
{ int i, j;
   int B[15], C[100];
   for (i = 0; i<= k; i++)
      C[i] = 0;
   for (j = 1; j <= n; j++)
      C[A[j]] = C[A[j]] + 1;
   for (i = 1; i<= k; i++)
      C[i] = C[i] + C[i-1];
   for (j = n; j >= 1; j--)
   {  B[C[A[j]]] = A[j];
      C[A[j]] = C[A[j]] - 1;   }
printf("The Sorted array is : ");
   for (i = 1; i<= n; i++)
printf("%d ", B[i]);}
int main()
{   int n, k = 0, A[15], i;
printf("Enter the number of input : ");
scanf("%d", &n);
printf("\nEnter the elements to be sorted :\n");
   for (i = 1; i<= n; i++)
   {      scanf("%d", &A[i]);
      if (A[i] > k) {
         k = A[i];       }    }
counting_sort(A, k, n);
printf("\n");
   return 0;
}
```

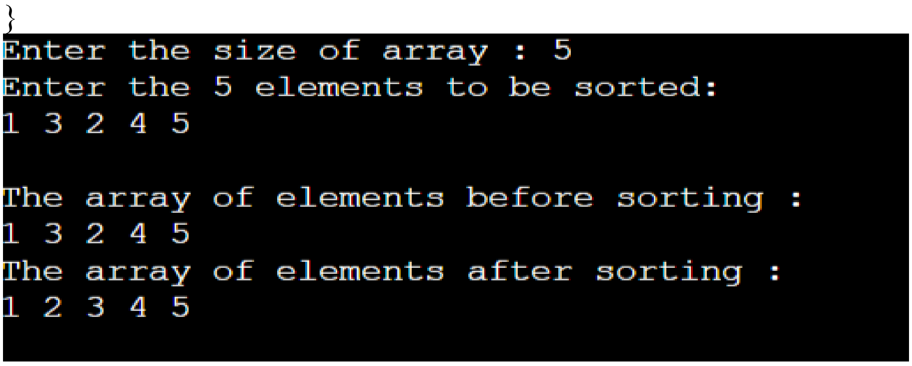


```
Enter the number of input : 5

Enter the elements to be sorted :
2 3 5 4 1
The Sorted array is : 1 2 3 4 5
```

Practical - 8

Wap to implement Bucket sort.

```
#include <stdio.h>
void Bucket_Sort(int array[], int n)
{
    int i, j;
    int count[n];
    for (i = 0; i< n; i++)
        count[i] = 0;

    for (i = 0; i< n; i++)
        (count[array[i]])++;

    for (i = 0, j = 0; i< n; i++)
        for(; count[i] > 0; (count[i])--)
            array[j++] = i;
}
int main()
{
    int array[100], i, num;

printf("Enter the size of array : ");
scanf("%d", &num);
printf("Enter the %d elements to be sorted:\n",num);
    for (i = 0; i< num; i++)
scanf("%d", &array[i]);
printf("\nThe array of elements before sorting : \n");
    for (i = 0; i< num; i++)
printf("%d ", array[i]);
printf("\nThe array of elements after sorting : \n");
Bucket_Sort(array, num);
    for (i = 0; i< num; i++)
printf("%d ", array[i]);
printf("\n");
    return 0;
}
```

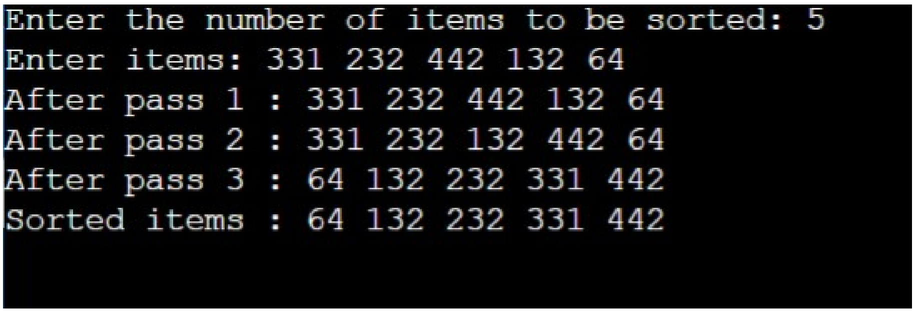


```
Enter the size of array : 5
Enter the 5 elements to be sorted:
1 3 2 4 5

The array of elements before sorting :
1 3 2 4 5
The array of elements after sorting :
1 2 3 4 5
```

Practical - 9

Write a program to implement radix sort in c

```
#include<stdio.h>
int get_max (int a[], int n){
  int max = a[0];
  for (int i = 1; i< n; i++)
    if (a[i] > max)
      max = a[i];
  return max;}
void radix_sort (int a[], int n){
  int bucket[10][10], bucket_cnt[10];
  int i, j, k, r, NOP = 0, divisor = 1, lar, pass;
  lar = get_max (a, n);
  while (lar > 0){
    NOP++;
    lar /= 10; }
  for (pass = 0; pass < NOP; pass++){
    for (i = 0; i< 10; i++){
bucket_cnt[i] = 0;    }
    for (i = 0; i< n; i++){
      r = (a[i] / divisor) % 10;
      bucket[r][bucket_cnt[r]] = a[i];
bucket_cnt[r] += 1;
    }
i = 0;
    for (k = 0; k < 10; k++){
      for (j = 0; j <bucket_cnt[k]; j++){
        a[i] = bucket[k][j];
i++;
      }
    }
    divisor *= 10;
printf ("After pass %d : ", pass + 1);
    for (i = 0; i< n; i++)
printf ("%d ", a[i]);
printf ("\n");
  }
}
int main (){
  int i, n, a[10];
printf ("Enter the number of items to be sorted: ");
scanf ("%d", &n);
printf ("Enter items: ");
  for (i = 0; i< n; i++){
scanf ("%d", &a[i]);
  }
radix_sort (a, n);
printf ("Sorted items : ");
   for (i = 0; i< n; i++)
printf ("%d ", a[i]);
printf ("\n");
   return 0;
}
```



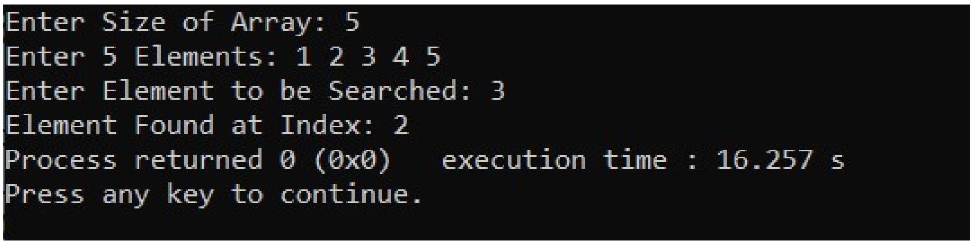
```
Enter the number of items to be sorted: 5
Enter items: 331 232 442 132 64
After pass 1 : 331 232 442 132 64
After pass 2 : 331 232 132 442 64
After pass 3 : 64 132 232 331 442
Sorted items : 64 132 232 331 442
```

Practical - 10

Write a C Program for Linear Search And Binary Search.

Linear Search:-

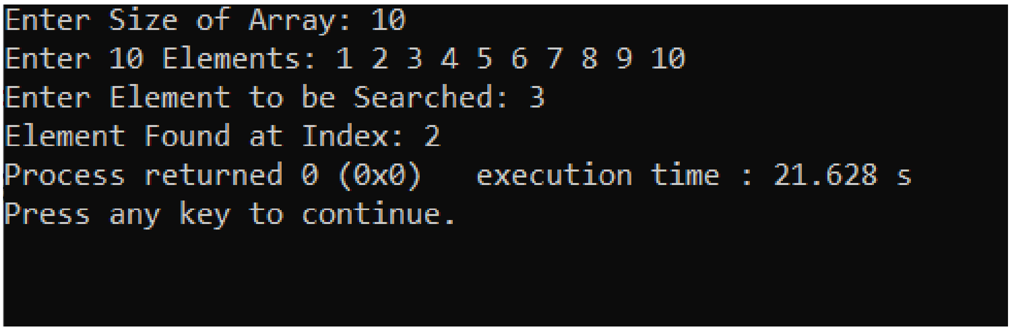
```
#include<stdio.h>
#include<conio.h>
int main()
{    int n,i,element,flag=0;
    int *a;
printf("Enter Size of Array: ");
scanf("%d",&n);
    a=(int*)malloc(sizeof(int)*n);
printf("Enter %d Elements: ",n);
    for(i=0; i<n; i++)
    {       scanf("%d",&a[i]);    }
printf("Enter Element to be Searched: ");
scanf("%d",&element);
    for(i=0; i<n; i++)
    {   if(a[i]==element)
        {   printf("Element Found at Index: %d",i);
            flag=1;
            break;
        } }
    if(flag==0)
    { printf("Element is Not Founded");    }
    return 0;}
```



```
Enter Size of Array: 5
Enter 5 Elements: 1 2 3 4 5
Enter Element to be Searched: 3
Element Found at Index: 2
Process returned 0 (0x0)   execution time : 16.257 s
Press any key to continue.
```

Binary Search:-

```
#include<stdio.h>
#include<conio.h>
int binary_searchRecurr(int *a,intstart,intend,int k)
{
    int mid;
    if(start<=end)
    {
        mid=(start+end)/2;
        if(a[mid]==k)
        {
            return mid;
        }
        else if(a[mid]>k)
binary_searchRecurr(a,start,mid,k);
        else
binary_searchRecurr(a,mid+1,end,k);
    }
    else
        return -1;
}
int main()
{
    int i,n,*a,k,ans;
printf("Enter Size of Array: ");
scanf("%d",&n);
    a=(int*)malloc(sizeof(int)*n);
printf("Enter %d Elements: ",n);
    for(i=0; i<n; i++)
    {
scanf("%d",&a[i]);
    }
printf("Enter Element to be Searched: ");
scanf("%d",&k);
ans=binary_searchRecurr(a,0,n-1,k);
    if(ans!=-1)
    {
printf("Element Found at Index: %d",ans);
    }
    else
    {
printf("Element is Not Founded");
    }
    return 0;}
```



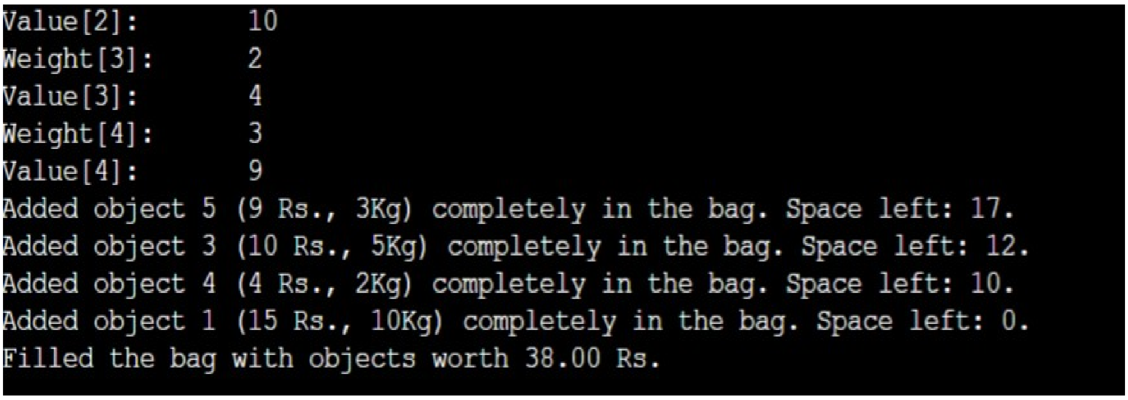
```
Enter Size of Array: 10
Enter 10 Elements: 1 2 3 4 5 6 7 8 9 10
Enter Element to be Searched: 3
Element Found at Index: 2
Process returned 0 (0x0)   execution time : 21.628 s
Press any key to continue.
```

Practical – 11

Write a program to find maximum profit with the help of Fractional Knapsack.

```
#include <stdio.h>
void main()
{
  int capacity, no_items, cur_weight, item;
    int used[10];
    float total_profit;
    int i;
    int weight[10];
    int value[10];
printf("Enter the capacity of knapsack:\n");
scanf("%d", &capacity);
printf("Enter the number of items:\n");
scanf("%d", &no_items);
printf("Enter the weight and value of %d item:\n", no_items);
    for (i = 0; i<no_items; i++)
    {
printf("Weight[%d]:\t", i);
scanf("%d", &weight[i]);
printf("Value[%d]:\t", i);
scanf("%d", &value[i]);
}
    for (i = 0; i<no_items; ++i)
        used[i] = 0;
cur_weight = capacity;
    while (cur_weight> 0)
    {
   item = -1;
        for (i = 0; i<no_items; ++i)
            if ((used[i] == 0) &&
                ((item == -1) || ((float) value[i] / weight[i] > (float) value[item] / weight[item])))
                item = i;
        used[item] = 1;
cur_weight -= weight[item];
total_profit += value[item];
        if (cur_weight>= 0)
            printf("Added object %d (%d Rs., %dKg) completely in the bag. Space left: %d.\n",
            item + 1, value[item], weight[item], cur_weight);
        else
        {
    int item_percent = (int) ((1 + (float) cur_weight / weight[item]) * 100);
printf("Added %d%% (%d Rs., %dKg) of object %d in the bag.\n", item_percent,
value[item], weight[item], item + 1);
total_profit -= value[item];
total_profit += (1 + (float)cur_weight / weight[item]) * value[item];
```

```
   }
  }
printf("Filled the bag with objects worth %.2f Rs.\n", total_profit);
}
```



```
Value[2]:       10
Weight[3]:      2
Value[3]:       4
Weight[4]:      3
Value[4]:       9
Added object 5 (9 Rs., 3Kg) completely in the bag. Space left: 17.
Added object 3 (10 Rs., 5Kg) completely in the bag. Space left: 12.
Added object 4 (4 Rs., 2Kg) completely in the bag. Space left: 10.
Added object 1 (15 Rs., 10Kg) completely in the bag. Space left: 0.
Filled the bag with objects worth 38.00 Rs.
```

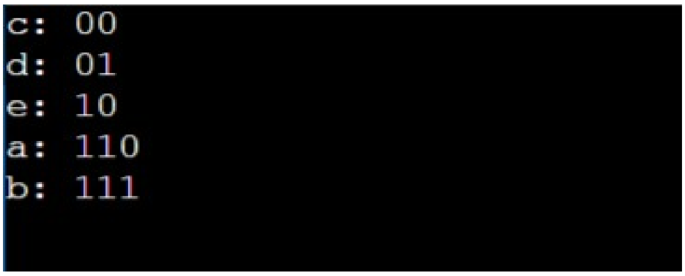
Practical - 12

Write a program to find Huffman code for frequency of a=50, b=30, c=20, d=10, e=10. The total character in the file are 120 without any space.

```
#include <bits/stdc++.h>
using namespace std;
#define MAX_TREE_HT 120
class QueueNode {
public:
   char data;
   unsigned freq;
QueueNode *left, *right;
};
class Queue {
public:
   int front, rear;
   int capacity;
QueueNode** array;
};
QueueNode* newNode(char data, unsigned freq)
{
QueueNode* temp = new QueueNode[(sizeof(QueueNode))];
   temp->left = temp->right = NULL;
   temp->data = data;
   temp->freq = freq;
   return temp;
}
Queue* createQueue(int capacity)
{
   Queue* queue = new Queue[(sizeof(Queue))];
   queue->front = queue->rear = -1;
   queue->capacity = capacity;
   queue->array = new QueueNode*[(queue->capacity
                              * sizeof(QueueNode*))];
   return queue;
}
int isSizeOne(Queue* queue)
{
   return queue->front == queue->rear
&& queue->front != -1;
}
int isEmpty(Queue* queue) { return queue->front == -1; }
int isFull(Queue* queue)
{
   return queue->rear == queue->capacity - 1;
}
void enQueue(Queue* queue, QueueNode* item)
{    if (isFull(queue))
```

```
        return;
    queue->array[++queue->rear] = item;
    if (queue->front == -1)
        ++queue->front;
}

QueueNode* deQueue(Queue* queue)
{
    if (isEmpty(queue))
        return NULL;
QueueNode* temp = queue->array[queue->front];
    if (queue->front
        == queue
               ->rear)
        queue->front = queue->rear = -1;
    else
        ++queue->front;
    return temp;
}
QueueNode* getFront(Queue* queue)
{
    if (isEmpty(queue))
        return NULL;
    return queue->array[queue->front];
}
QueueNode* findMin(Queue* firstQueue, Queue* secondQueue)
{
    if (isEmpty(firstQueue))
        return deQueue(secondQueue);
    if (isEmpty(secondQueue))
        return deQueue(firstQueue);
    if (getFront(firstQueue)->freq
<getFront(secondQueue)->freq)
        return deQueue(firstQueue);
    return deQueue(secondQueue);
}
int isLeaf(QueueNode* root)
{
    return !(root->left) && !(root->right);
}
void printArr(int arr[], int n)
{
    int i;
    for (i = 0; i< n; ++i)
cout<<arr[i];
cout<<endl;
}
QueueNode* buildHuffmanTree(char data[], int freq[],
                            int size)
{QueueNode *left, *right, *top;
```

```
   Queue* firstQueue = createQueue(size);
   Queue* secondQueue = createQueue(size);
   for (int i = 0; i< size; ++i)
enQueue(firstQueue, newNode(data[i], freq[i]));
   while (
      !(isEmpty(firstQueue) &&isSizeOne(secondQueue))) {
      left = findMin(firstQueue, secondQueue);
      right = findMin(firstQueue, secondQueue);
      top = newNode('$', left->freq + right->freq);
      top->left = left;
      top->right = right;
enQueue(secondQueue, top);
   }
   return deQueue(secondQueue);
}
void printCodes(QueueNode* root, int arr[], int top)
{
   if (root->left) {
arr[top] = 0;
printCodes(root->left, arr, top + 1);
   }
   if (root->right) {
arr[top] = 1;
printCodes(root->right, arr, top + 1);
   }
   if (isLeaf(root)) {
cout<< root->data << ": ";
printArr(arr, top);
   }}
void HuffmanCodes(char data[], int freq[], int size)
{  QueueNode* root = buildHuffmanTree(data, freq, size);
  int arr[MAX_TREE_HT], top = 0;
printCodes(root, arr, top);
}
 int main()
{
   char arr[] = { 'a', 'b', 'c', 'd', 'e'};
   int freq[] = { 50, 30, 20, 10,10 };
   int size = sizeof(arr) / sizeof(arr[0]);
HuffmanCodes(arr, freq, size);
   return 0;
}
```

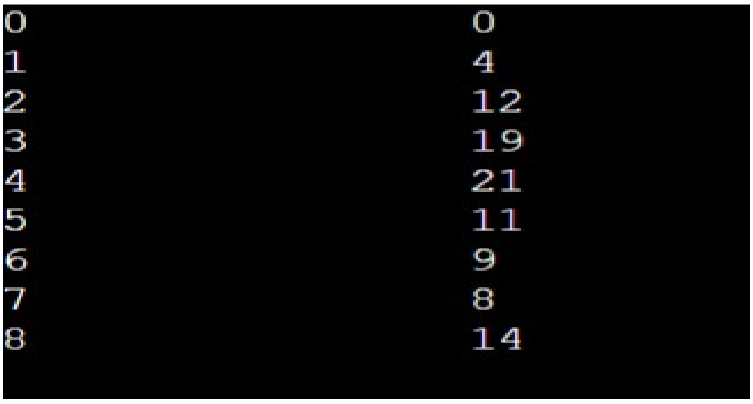


```
c: 00
d: 01
e: 10
a: 110
b: 111
```

Practical - 13

Write a program to find shortest path in a graph. Consider one node as a source node and all remaining nodes are destination nodes.

```
#include <limits.h>
#include <stdio.h>
#include <stdbool.h>
#define V 9
int minDistance(int dist[], bool sptSet[])
{
        int min = INT_MAX, min_index;
        for (int v = 0; v < V; v++)
                if (sptSet[v] == false &&dist[v] <= min)
                        min = dist[v], min_index = v;
        return min_index;
}
void printSolution(int dist[])
{
        printf("Vertex \t\t Distance from Source\n");
        for (int i = 0; i< V; i++)
                printf("%d \t\t %d\n", i, dist[i]);
}
void dijkstra(int graph[V][V], int src)
{
        int dist[V];
        bool sptSet[V];
        for (int i = 0; i< V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;
        dist[src] = 0;
        for (int count = 0; count < V - 1; count++) {
        int u = minDistance(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];
        }
        printSolution(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;
}
```

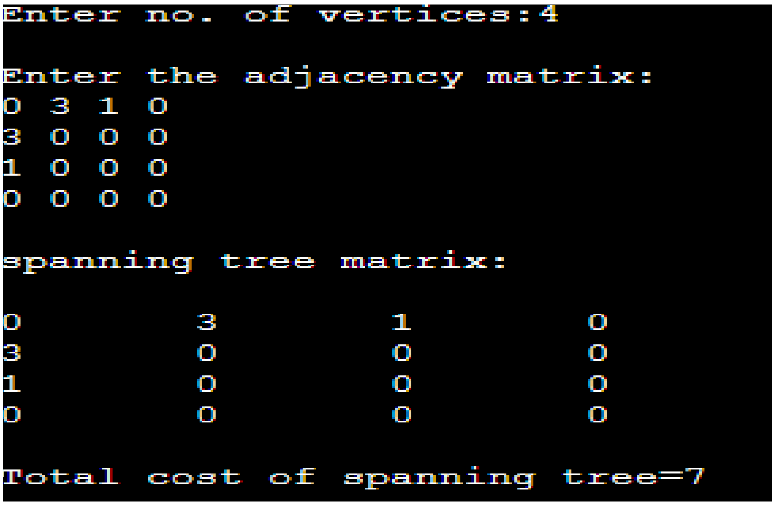


```
0                                  0
1                                  4
2                                  12
3                                  19
4                                  21
5                                  11
6                                  9
7                                  8
8                                  14
```

Practical - 14

Write a program to find minimum cost spanning tree in a graph having N vertices and M edges. *Using prim’s algorithm*

```
#include<stdio.h>
#include<stdlib.h>
#define infinity 9999
#define MAX 20
int G[MAX][MAX],spanning[MAX][MAX],n;
int prims();
int main()
{
int i,j,total_cost;
printf("Enter no. of vertices:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
total_cost=prims();
printf("\nspanning tree matrix:\n");
for(i=0;i<n;i++)
{
printf("\n");
for(j=0;j<n;j++)
printf("%d\t",spanning[i][j]);
}
printf("\n\nTotal cost of spanning tree=%d",total_cost);
return 0;
}
int prims()
{
int cost[MAX][MAX];
int u,v,min_distance,distance[MAX],from[MAX];
int visited[MAX],no_of_edges,i,min_cost,j;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
{
if(G[i][j]==0)
cost[i][j]=infinity;
else
cost[i][j]=G[i][j];
spanning[i][j]=0;
}
distance[0]=0;
visited[0]=1;
for(i=1;i<n;i++) {
distance[i]=cost[0][i];
from[i]=0;
visited[i]=0;
}
min_cost=0;
no_of_edges=n-1;
while(no_of_edges>0)  {
min_distance=infinity;
for(i=1;i<n;i++)
if(visited[i]==0&&distance[i]<min_distance) {
v=i;
min_distance=distance[i];
}
u=from[v];
spanning[u][v]=distance[v];
spanning[v][u]=distance[v];
no_of_edges--;
visited[v]=1;
for(i=1;i<n;i++)
if(visited[i]==0&&cost[i][v]<distance[i])
{
distance[i]=cost[i][v];
from[i]=v;}
min_cost=min_cost+cost[u][v];}
return(min_cost);}
```

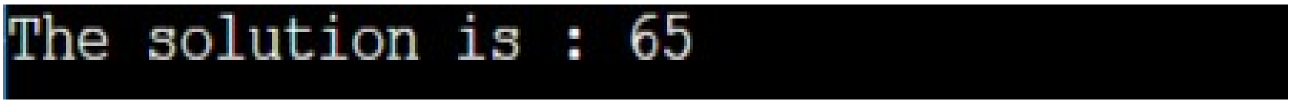


```
Enter no. of vertices:4

Enter the adjacency matrix:
0 3 1 0
3 0 0 0
1 0 0 0
0 0 0 0

spanning tree matrix:

0       3       1       0
3       0       0       0
1       0       0       0
0       0       0       0

Total cost of spanning tree=7
```

Practical - 15

Write a program to find maximum profit in a 0-1 Knapsack.

```
#include<stdio.h>
int max(int a, int b) {
  if(a>b){
    return a;
  } else {
    return b;
  }
}
int knapsack(int W, int wt[], int val[], int n) {
  int i, w;
  int knap[n+1][W+1];
  for (i = 0; i<= n; i++) {
    for (w = 0; w <= W; w++) {
      if (i==0 || w==0)
        knap[i][w] = 0;
      else if (wt[i-1] <= w)
        knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);
      else
        knap[i][w] = knap[i-1][w];
    }
  }
  return knap[n][W];
}
int main() {
  int val[] = {20, 25, 40};
  int wt[] = {25, 20, 30};
  int W = 50;
  int n = sizeof(val)/sizeof(val[0]);
printf("The solution is : %d", knapsack(W, wt, val, n));
  return 0;
}
```

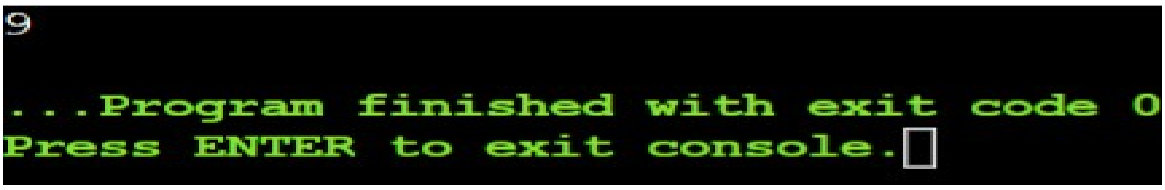


```
The solution is : 65
```

Practical - 16

Write a program to find shortest path with the help of multi-stage graph.

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

#define N 8
#define INF INT_MAX
int shortestDist(int graph[N][N])
   int dist[N];
dist[N-1] = 0;
   for (int i = N-2 ; i>= 0 ; i--)
   {
dist[i] = INF;
      for (int j = i ; j < N ; j++)
      {
         if (graph[i][j] == INF)
            continue;
dist[i] = min(dist[i], graph[i][j] +
dist[j]);
      }  }
   return dist[0];
}
int main()
{
   int graph[N][N] =
     {{INF, 1, 2, 5, INF, INF, INF, INF},
      {INF, INF, INF, INF, 4, 11, INF, INF},
      {INF, INF, INF, INF, 9, 5, 16, INF},
      {INF, INF, INF, INF, INF, INF, 2, INF},
      {INF, INF, INF, INF, INF, INF, INF, 18},
      {INF, INF, INF, INF, INF, INF, INF, 13},
      {INF, INF, INF, INF, INF, INF, INF, 2},
     {INF, INF, INF, INF, INF, INF, INF, INF}};
cout<<shortestDist(graph);
   return 0;
}
```

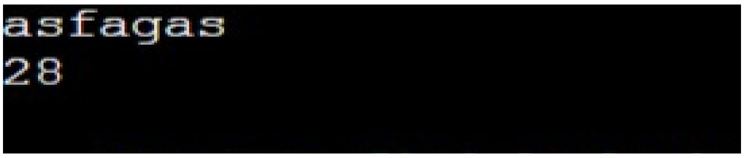


```
9

...Program finished with exit code 0
Press ENTER to exit console.
```

Practical - 17

Write a program to find how many substrings are possible if length of string is N.

```
#include <bits/stdc++.h>

using namespace std;

int countNonEmptySubstr(string str)
{
  int n = str.length();
  return n*(n+1)/2;
}
int main()
{
   string s;
cin>>s;
cout<<countNonEmptySubstr(s);
   return 0;
}
```

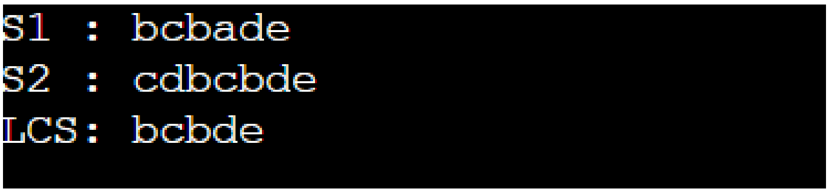


```
asfagas
28
```

Practical - 18

Write a program to find longest common subsequence between 2 strings.

```
#include <stdio.h>
#include <string.h>
int i, j, m, n, LCS_table[20][20];
char S1[20] = "bcbade", S2[20] = "cdbcbde", b[20][20];
void lcsAlgo() {
  m = strlen(S1);
  n = strlen(S2);
  for (i = 0; i<= m; i++)
LCS_table[i][0] = 0;
  for (i = 0; i<= n; i++)
LCS_table[0][i] = 0;
  for (i = 1; i<= m; i++)
    for (j = 1; j <= n; j++) {
      if (S1[i - 1] == S2[j - 1]) {
LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
      } else if (LCS_table[i - 1][j] >= LCS_table[i][j - 1]) {
LCS_table[i][j] = LCS_table[i - 1][j];
      } else {
LCS_table[i][j] = LCS_table[i][j - 1];     }    }
  int index = LCS_table[m][n];
  char lcsAlgo[index + 1];
lcsAlgo[index] = '\0';
  int i = m, j = n;
  while (i> 0 && j > 0) {
    if (S1[i - 1] == S2[j - 1]) {
lcsAlgo[index - 1] = S1[i - 1];
i--;
      j--;
      index--;
    }     else if (LCS_table[i - 1][j] >LCS_table[i][j - 1])
i--;
    else
      j--;    }
printf("S1 : %s \nS2 : %s \n", S1, S2);
printf("LCS: %s", lcsAlgo);
}    int main() {
lcsAlgo();
printf("\n");  }
```



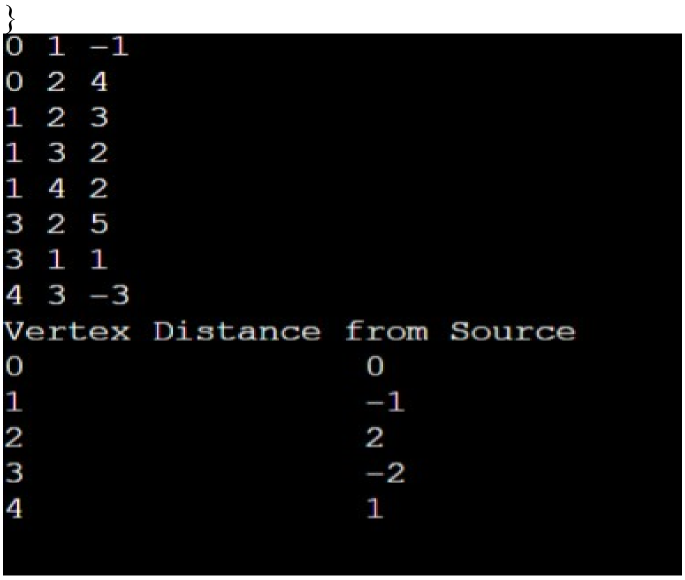
```
S1  : bcbade
S2  : cdbcbde
LCS: bcbde
```

Practical - 19

Write a program to find shortest path with the help of Bellman-Ford algorithm.

```
#include <bits/stdc++.h>
using namespace std;
struct Edge {
        int src, dest, weight;
};
struct Graph {
        int V, E;
        struct Edge* edge;
};
struct Graph* createGraph(int V, int E)
{
        struct Graph* graph = new Graph;
        graph->V = V;
        graph->E = E;
        graph->edge = new Edge[E];
        return graph;
}
void printArr(int dist[], int n)
{
        printf("Vertex Distance from Source\n");
        for (int i = 0; i< n; ++i)
                printf("%d \t\t %d\n", i, dist[i]);
}
void BellmanFord(struct Graph* graph, int src)
{
        int V = graph->V;
        int E = graph->E;
        int dist[V];
        for (int i = 0; i< V; i++)
                dist[i] = INT_MAX;
        dist[src] = 0;
        for (int i = 1; i<= V - 1; i++) {
                for (int j = 0; j < E; j++) {
                        int u = graph->edge[j].src;
                        int v = graph->edge[j].dest;
                        int weight = graph->edge[j].weight;
                        if (dist[u] != INT_MAX
                                &&dist[u] + weight <dist[v])
                                dist[v] = dist[u] + weight;
                }
}
        for (int i = 0; i< E; i++) {
                int u = graph->edge[i].src;
```

```
                    int v = graph->edge[i].dest;
                    int weight = graph->edge[i].weight;
                    if (dist[u] != INT_MAX
                            &&dist[u] + weight <dist[v]) {
                            printf("Graph contains negative weight cycle");
                            return;
                                                        }        }
          printArr(dist, V);
          return;
}
int main()
{
          int V = 5;
          int E = 8;
          struct Graph* graph = createGraph(V, E);
   for(int i=0;i<E;i++)
   {
      //Enter Source, Enter destination, Enter weight
cin>>graph->edge[i].src;
cin>>graph->edge[i].dest;
cin>>graph->edge[i].weight;
   }      BellmanFord(graph, 0);
return 0;
}
```

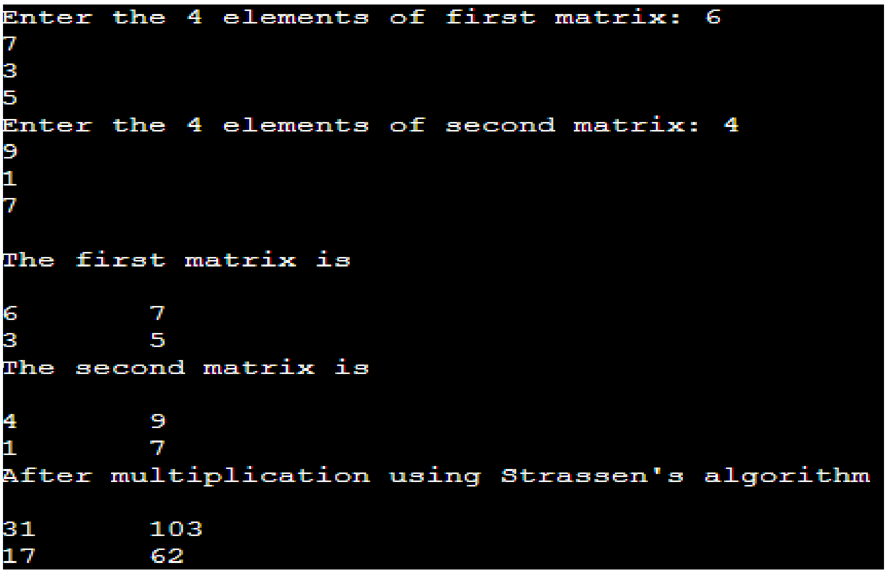


```
0 1 -1
0 2 4
1 2 3
1 3 2
1 4 2
3 2 5
3 1 1
4 3 -3
Vertex Distance from Source
0               0
1               -1
2               2
3               -2
4               1
```

Practical –20

Write a program to find minimum scalar multiplication in matrix chain multiplication.

```
#include<stdio.h>
int main(){
  int a[2][2], b[2][2], c[2][2], i, j;
  int m1, m2, m3, m4 , m5, m6, m7;
printf("Enter the 4 elements of first matrix: ");
  for(i = 0;i < 2; i++)
    for(j = 0;j < 2; j++)
scanf("%d", &a[i][j]);
printf("Enter the 4 elements of second matrix: ");
  for(i = 0; i< 2; i++)
      for(j = 0;j < 2; j++)
scanf("%d", &b[i][j]);
printf("\nThe first matrix is\n");
  for(i = 0; i< 2; i++){
printf("\n");
      for(j = 0; j < 2; j++)
printf("%d\t", a[i][j]);
 }
printf("\nThe second matrix is\n");
  for(i = 0;i < 2; i++){
printf("\n");
      for(j = 0;j < 2; j++)
printf("%d\t", b[i][j]);
  }
  m1= (a[0][0] + a[1][1]) * (b[0][0] + b[1][1]);
  m2= (a[1][0] + a[1][1]) * b[0][0];
  m3= a[0][0] * (b[0][1] - b[1][1]);
  m4= a[1][1] * (b[1][0] - b[0][0]);
  m5= (a[0][0] + a[0][1]) * b[1][1];
  m6= (a[1][0] - a[0][0]) * (b[0][0]+b[0][1]);
  m7= (a[0][1] - a[1][1]) * (b[1][0]+b[1][1]);
  c[0][0] = m1 + m4- m5 + m7;
  c[0][1] = m3 + m5;
  c[1][0] = m2 + m4;
  c[1][1] = m1 - m2 + m3 + m6;
printf("\nAfter multiplication using Strassen's algorithm \n");
   for(i = 0; i< 2 ; i++)
{
printf("\n");
      for(j = 0;j < 2; j++)
printf("%d\t", c[i][j]);
   }
   return 0;
}
```

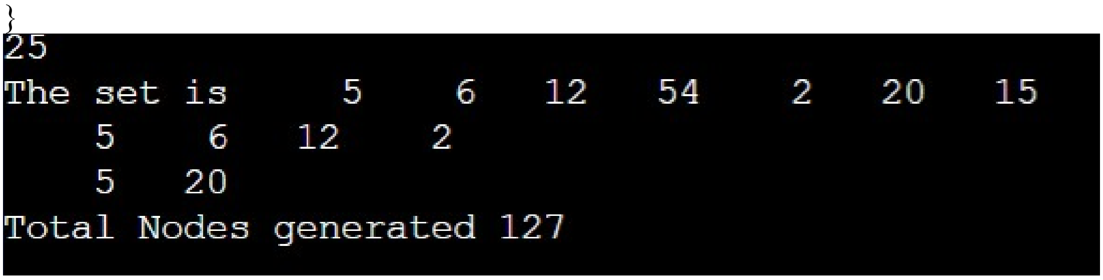


```
Enter the 4 elements of first matrix: 6
7
3
5
Enter the 4 elements of second matrix: 4
9
1
7

The first matrix is

6       7
3       5
The second matrix is

4       9
1       7
After multiplication using Strassen's algorithm

31      103
17      62
```

Practical - 21

Write a program to find sum of subset.

```
#include <stdio.h>
#include <stdlib.h>
static int total_nodes;
void printValues(int A[], int size){
   for (int i = 0; i< size; i++) {
printf("%*d", 5, A[i]);
   }
printf("\n");
}
void subset_sum(int s[], int t[], int s_size, int t_size, int sum, int ite, int const target_sum){
total_nodes++;
   if (target_sum == sum) {
printValues(t, t_size);
subset_sum(s, t, s_size, t_size - 1, sum - s[ite], ite + 1, target_sum);
      return;
   }
   else {
      for (int i = ite; i<s_size; i++) {
         t[t_size] = s[i];
subset_sum(s, t, s_size, t_size + 1, sum + s[i], i + 1, target_sum);
      } }}
void generateSubsets(int s[], int size, int target_sum){
   int* tuplet_vector = (int*)malloc(size * sizeof(int));
subset_sum(s, tuplet_vector, size, 0, 0, 0, target_sum);
   free(tuplet_vector);
}
int main(){
    int target_sum;
scanf("%d",&target_sum);

   int set[] = { 5, 6, 12 , 54, 2 , 20 , 15 };
   int size = sizeof(set) / sizeof(set[0]);
printf("The set is ");
printValues(set , size);
generateSubsets(set, size, target_sum);
printf("Total Nodes generated %d\n", total_nodes);
   return 0;
}
```

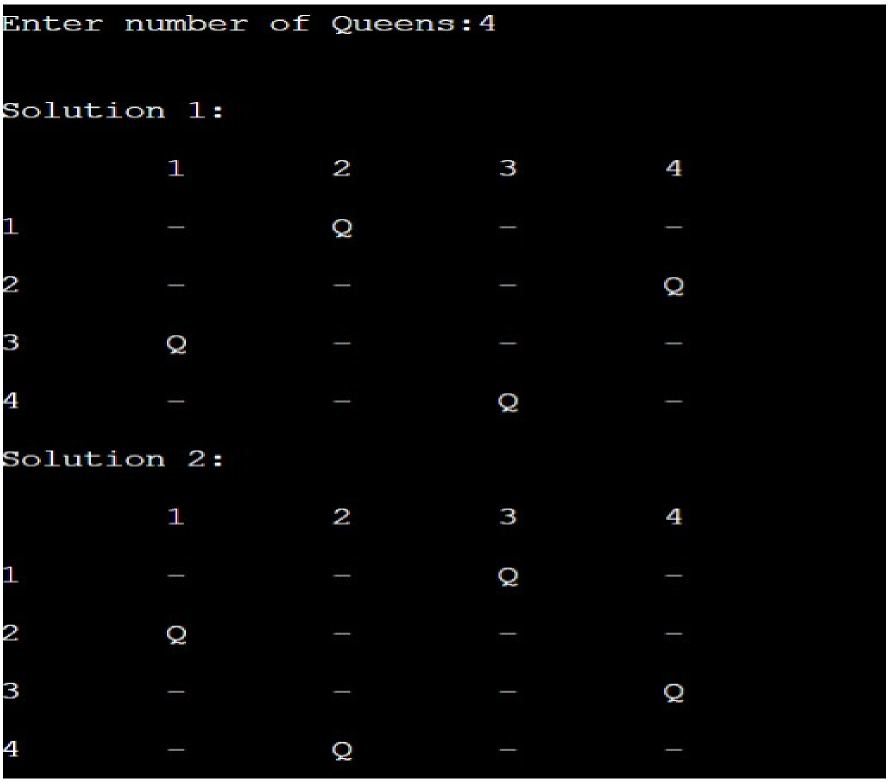


```
25
The set is     5    6   12   54    2   20   15
    5    6   12    2
    5   20
Total Nodes generated 127
```

Practical –22

Write a program to implement N-Queens problem.

```
#include<stdio.h>
#include<math.h>
int board[20],count;
int main()
{int n,i,j;
void queen(int row,int n);
printf(" - N Queens Problem Using Backtracking -");
printf("\n\nEnter 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);
  }}}
```



```
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       -       -
```