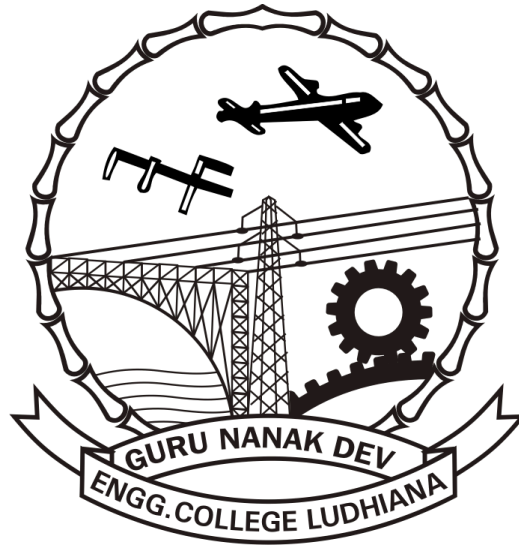


# **INSTRUCTION MANUAL**

## **DESIGN & ANALYSIS OF ALGORITHMS LAB**



**Prepared by**

**Er. Jappreet Kaur**  
**Assistant Professor (CSE)**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**GURU NANAK DEV ENGINEERING COLLEGE**

**LUDHIANA – 141006**

## **DECLARATION**

This Manual of Design & Analysis Of Algorithms Lab (BTCS-508) has been prepared by me as per syllabus of Design & Analysis Of Algorithms Lab (BTCS-508).

**Signature**

# SYLLABUS

## **BTCS 508 Design & Analysis of Algorithms Lab**

**Objective:** To get a firsthand experience of implementing well-known algorithms in a high level language.

To be able to compare the practical performance of different algorithms for the same problem.

1. Code and analyze to compute the greatest common divisor (GCD) of two numbers.
  2. Code and analyze to find the median element in an array of integers.
  3. Code and analyze to find the majority element in an array of integers.
  4. Code and analyze to sort an array of integers using Heap sort.
  5. Code and analyze to sort an array of integers using Merge sort.
  6. Code and analyze to sort an array of integers using Quick sort.
  7. Code and analyze to find the edit distance between two character strings using dynamic programming.
  8. Code and analyze to find an optimal solution to weighted interval scheduling using dynamic programming.
  9. Code and analyze to find an optimal solution to matrix chain multiplication using dynamic programming.
  10. Code and analyze to do a depth-first search (DFS) on an undirected graph. Implementing an application of DFS such as
    - I. To find the topological sort of a directed acyclic graph, OR
    - II. To find a path from source to goal in a maze.
  11. Code and analyze to do a breadth-first search (BFS) on an undirected graph. Implementing an application of BFS such as
    - I. To find connected components of an undirected graph, OR
    - II. To check whether a given graph is bipartite.
  12. Code and analyze to find shortest paths in a graph with positive edge weights using Dijkstra's algorithm.
  13. Code and analyze to find shortest paths in a graph with arbitrary edge weights using Bellman Ford algorithm.
  14. Code and analyze to find the minimum spanning tree in a weighted, undirected graph.
  15. Code and analyze to find all occurrences of a pattern P in a given string S.
  16. Code and analyze to multiply two large integers using Karatsuba algorithm.
  17. Code and analyze to compute the convex hull of a set of points in the plane.
- (Mini-project Topic) Program to multiply two polynomials using Fast Fourier Transform (FFT)

## INDEX

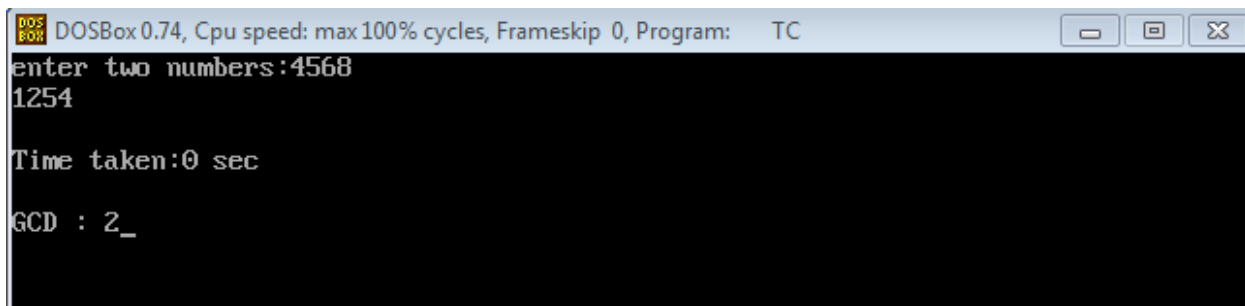
Sr. No.	Contents	Page No
1	Code and analyze to compute the greatest common divisor (gcd) of two numbers.	1
2	Code and analyze to find the median element in an array of integers.	2
3	Code and analyze to find the majority element in an array of integers.	4
4	Code and analyze to sort an array of integers using heap sort.	6
5	Code and analyze to sort an array of integers using merge sort.	9
6.	Code and analyze to sort an array of integers using quick sort.	12
7.	Code and analyze to find the edit distance between two character strings using dynamic programming.	14
8	Code and analyze to find the longest common subsequence between two character strings using dynamic programming.	16
9	Code and analyze to find an optimal solution to matrix chain multiplication using dynamic programming.	18
10	Code and analyze to do a breadth-first search (bfs) on an undirected graph. implementing an application of bfs such as to find connected components of an undirected graph, or to check whether a given graph is bipartite.	20
11	Code and analyze to find shortest paths in a graph with positive edge weights using dijkstra's algorithm.	25
12	Code and analyze to find shortest paths in a graph with arbitrary edge weights using bellman ford algorithm.	28
13	Code and analyze to find the minimum spanning tree in a weighted, undirected graph.	31
14	Code and analyze to find all occurrences of a pattern p in a given string s. (kmp)	36
15	Code and analyze to multiply two large integers using karatsuba algorithm.	39
16	Code and analyze to compute the convex hull of a set of points in the plane.	44
17	(Mini-project Topic) program to multiply two polynomials using fast fouriour transform(fft)	50

## 1.CODE AND ANALYZE TO COMPUTE THE GREATEST COMMON DIVISOR (GCD) OF TWO NUMBERS.

```
#include<iostream.h>
#include<conio.h>
#include<time.h>

long int euclid(long int m,long int n)
{
    clock_t start,end;
    start=clock();
    long int r;
    while(n!=0)
    {
        r=m%n;
        m=n;
        n=r;
    }
    end=clock();
    cout<<endl<<"Time taken:"<<(end-start)/CLK_TCK<<" sec";
    return m;
}

void main()
{
    long int x,y;
    clrscr();
    cout<<"enter two numbers:";
    cin>>x>>y;
    cout<<endl<<endl<<"GCD : "<<euclid(x,y);
    getch();
}
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
enter two numbers:4568
1254
Time taken:0 sec
GCD : 2_
```

## 2.CODE AND ANALYZE TO FIND THE MEDIAN ELEMENT IN AN ARRAY OF INTEGERS.

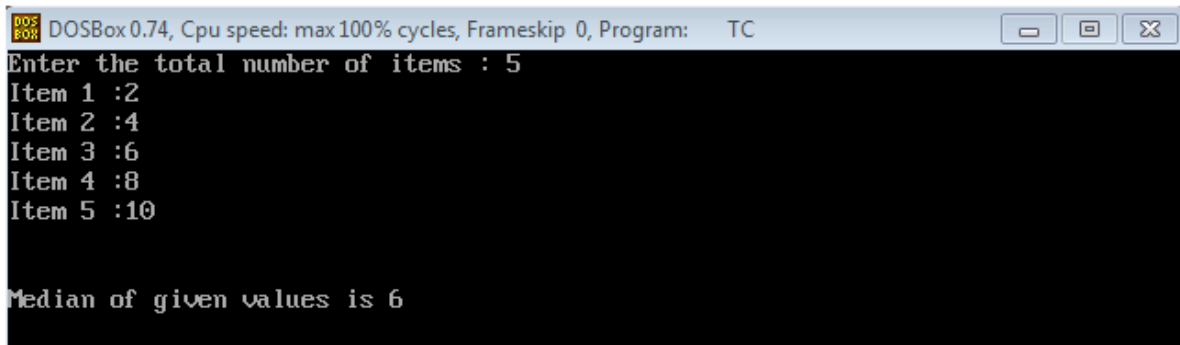
```
#include<iostream.h>
#include<conio.h>
#define s 20

void sort(int *ptr, int size)
{    //Bubble Sort
    int temp;
    for (int i=0; i<size-1; i++)
    {
        for (int j=0; j<size-1-i; j++)
        {
            if (*(ptr+j)>*(ptr+j+1))
            {
                temp=*(ptr+j);
                *(ptr+j)=*(ptr+j+1);
                *(ptr+j+1)=temp;
            }
        }
    }
}

float median(int *ptr, int size)
{
    sort(ptr, size);    //This sorts the array in asscending order
    if (size%2==0)
    {
        //If the total number of numbers are EVEN
        return (*(ptr+(size/2)-1)+ *(ptr+(size/2)))/float(2);
    }
    else
    {
        //If the total number of numbers are ODD
        return (*(ptr+(size/2)));
    }
}

void main ()
{
```

```
clrscr();
int size;
cout << "Enter the total number of items : ";
cin >> size;
int array_items[s];
for (int i=0; i<size; i++)
{
    cout << "Item " << i+1 << " :";
    cin >> array_items[i];
}
cout << "\n\nMedian of given values is " << median(array_items, size);
getch();
}
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter the total number of items : 5
Item 1 :2
Item 2 :4
Item 3 :6
Item 4 :8
Item 5 :10

Median of given values is 6
```

### 3.CODE AND ANALYZE TO FIND THE MAJORITY ELEMENT IN AN ARRAY OF INTEGERS.

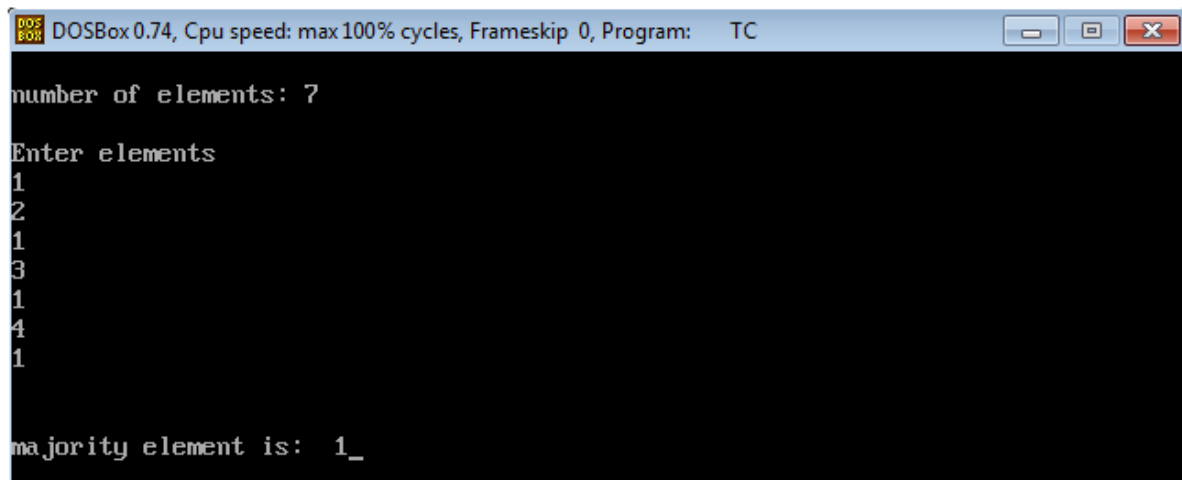
```
#include<iostream.h>
#include<conio.h>
#define N 20
int majority(int [],int );
void main()
{
clrscr();
int a[N],n,m;
cout<<"\nnumber of elements: ";
cin>>n;
cout<<"\nEnter elements\n";
for(int i=0;i<n;i++)
cin>>a[i];
m=majority(a,n);
if(m!=0)
cout<<"\n\nmajority element is: " <<m;
else
cout<<"\n\nNo majority element exists\n";

getch();
}

int majority(int a[],int n)
{
int maj_idx=0,count=1;
for(int i=0;i<n;i++)
{
if(a[maj_idx]==a[i])
count++;
else
count--;
if(count==0)
{
maj_idx=i;
count=1;
}
}
}
```



```
count=0;
for(i=0;i<n;i++)
{
if(a[i]==a[maj_idx])
    count++;
}
if(count>n/2)
return a[maj_idx];
else
return 0;
}
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
number of elements: 7
Enter elements
1
2
1
3
1
4
1
majority element is: 1_
```

#### 4. CODE AND ANALYZE TO SORT AN ARRAY OF INTEGERS USING HEAP SORT.

```
#include<iostream.h>
#include<conio.h>
#define N 10
void heapsort(int [],int);
void heapify(int a[],int n);
void reheapifydownward(int a[],int start,int finish);
```

```
void main()
{
    clrscr();
    int n; int a[N];
    cout<<"\nenter size of array\n";
    cin>>n;
    cout<<"\nenter elements\n";
    for(int i=1;i<=n;i++)
        cin>>a[i];
    heapsort(a,n);
    cout<<"\nSorted array is:\n";
    for(i=1;i<=n;i++)
        cout<<a[i]<<endl;
    getch();
}
```

```
void heapsort(int a[],int n)
{
    int i,temp;
    heapify(a,n);
    for(i=n;i>1;i--)
    {
        temp=a[1];
        a[1]=a[i];
        a[i]=temp;
        reheapifydownward(a,1,i-1);
    }
}
```

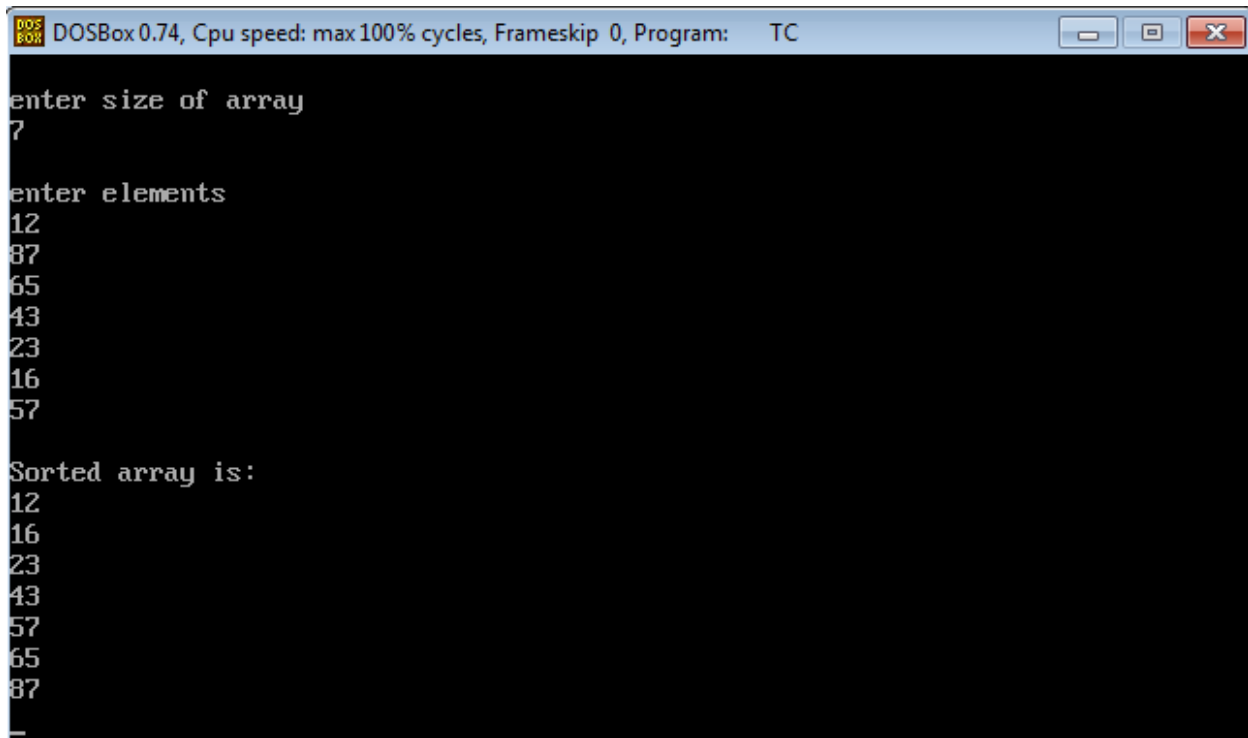
```
void heapify(int a[],int n)
```

```

{
    int i,index;
    index=n/2;
    for(i=index;i>=1;i--)
        reheapifydownward(a,i,n);
}

void reheapifydownward(int a[],int start,int finish)
{
    int index,lchild,rchild,maximum,temp;
    lchild=2*start;
    rchild=lchild+1;
    if(lchild<=finish)
    {
        maximum=a[lchild];
        index=lchild;
        if(rchild<=finish)
        {
            if(a[rchild]>maximum)
            {
                maximum=a[rchild];
                index=rchild;
            }
        }
        if(a[start]<a[index])
        {
            temp=a[start];
            a[start]=a[index];
            a[index]=temp;
            reheapifydownward(a,index,finish);
        }
    }
}

```



DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
enter size of array
7
enter elements
12
87
65
43
23
16
57
Sorted array is:
12
16
23
43
57
65
87
_
```

## 5. CODE AND ANALYZE TO SORT AN ARRAY OF INTEGERS USING MERGE SORT.

```
#include<iostream.h>
#include<conio.h>
#define N 10
void main()
{
clrscr();
int n,a[N],i;
void mergesort(int [],int,int);
cout<<"Insert the number of elements to be inserted: ";
cin>>n;
cout<<"\nInsert "<<n<<" elements\n";
for(i=1;i<=n;i++)
{
cin>>a[i];
}
cout<<"\nSorted array is: \n";
mergesort(a,1,n);
for(i=1;i<=n;i++)
{
cout<<a[i]<<endl;
}

getch();
}

void mergesort(int a[],int beg, int end)
{
int mid;
void mergesubarrays(int [],int,int,int,int);
if(beg<end)
{
mid=(beg+end)/2;
mergesort(a,beg,mid);
mergesort(a,mid+1,end);
mergesubarrays(a,beg,mid,mid+1,end);
}
```

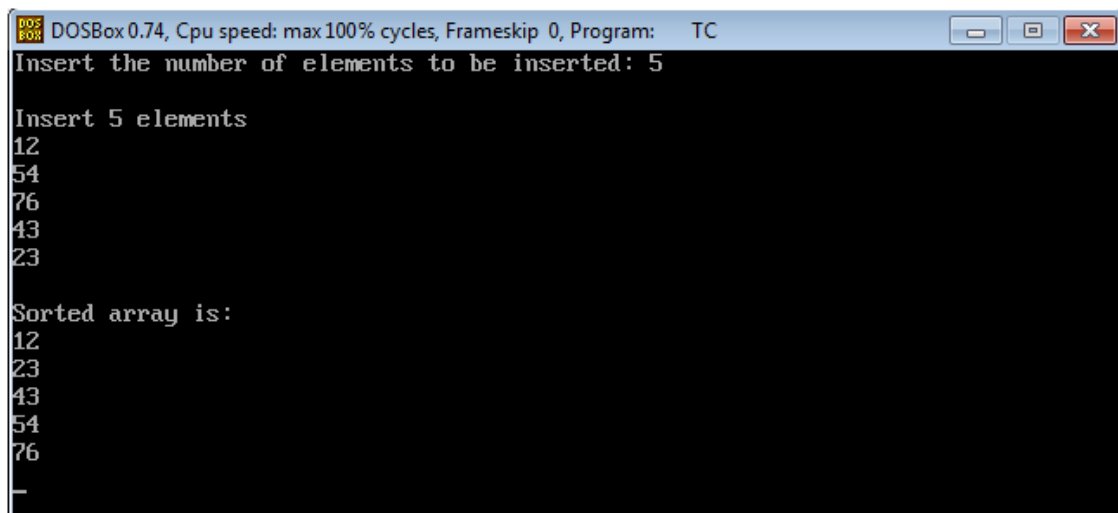
```

}
void mergesubarrays(int a[],int lb,int lr,int rb,int rr)
{
int na,nb,nc,k,c[N];
na=lb;
nb=rb;
nc=lb;
while((na<=lr)&&(nb<=rr))
{
if(a[na]<a[nb])
{
c[nc]=a[na++];
}
else
{
c[nc]=a[nb++];
}
nc++;
}

if(na>lr)
{
while(nb<=rr)
{
c[nc++]=a[nb++];
}
}
else
{
while(na<=lr)
{
c[nc++]=a[na++];
}
}

for(k=lb;k<=rr;k++)
{
a[k]=c[k];
}
}

```



The image shows a DOSBox window titled "DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window contains a C program that implements an insertion sort algorithm. The program prompts the user to "Insert the number of elements to be inserted: 5". It then prompts the user to "Insert 5 elements" and reads five integers: 12, 54, 76, 43, and 23. Finally, it displays the "Sorted array is:" followed by the sorted elements: 12, 23, 43, 54, and 76.

```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Insert the number of elements to be inserted: 5

Insert 5 elements
12
54
76
43
23

Sorted array is:
12
23
43
54
76
_
```

## 6. CODE AND ANALYZE TO SORT AN ARRAY OF INTEGERS USING QUICK SORT.

```
#include<iostream.h>
#include<conio.h>
#include<time.h>
#define N 20
void reduction(int a[],int beg, int end,int *loc)
{
    int left,right,temp;
    left=*loc=beg;
    right=end;
    int done=0;
    while(!done)
    {
        while((a[*loc]<a[right])&&(*loc!=right))
            right--;
        if(*loc==right)
            done=1;
        else
        {
            temp=a[*loc];
            a[*loc]=a[right];
            a[right]=temp;
            *loc=right;
        }
        if(!done)
        {
            while((a[*loc]>a[left])&&(*loc!=left))
                left++;
            if(*loc==left)
                done=1;
            else
            {
                temp=a[*loc];
                a[*loc]=a[left];
                a[left]=temp;
                *loc=left;
            }
        }
    }
}
```

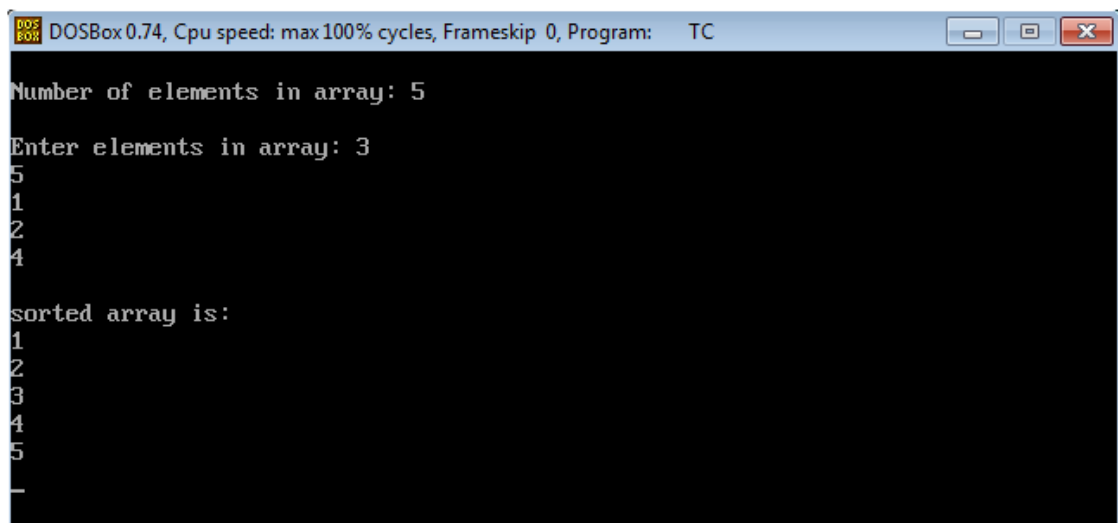


```

void quickr(int a[],int lb,int ub)
{
    int loc;
    if(lb<ub)
    {
        reduction(a,lb,ub,&loc);
        quickr(a,lb,loc-1);
        quickr(a,loc+1,ub);
    }
}

void main()
{
    clrscr();
    int q[N],n;
    cout<<"\nNumber of elements in array: ";
    cin>>n;
    cout<<"\nEnter elements in array: ";
    for(int i=0;i<n;i++)
        cin>>q[i];
    quickr(q,0,n-1);
    cout<<"\nsorted array is:\n";
    for(i=0;i<n;i++)
        cout<<q[i]<<endl;
    getch();
}

```



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Number of elements in array: 5
Enter elements in array: 3
5
1
2
4
1
sorted array is:
1
2
3
4
5
_

```

## 7. CODE AND ANALYZE TO FIND THE EDIT DISTANCE BETWEEN TWO CHARACTER STRINGS USING DYNAMIC PROGRAMMING.

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#define STRING_X "SUN"
#define STRING_Y "SATURN"
#define SENTINEL (-1)
#define EDIT_COST (1)

inline
int min(int a, int b)
{
    return a < b ? a : b;
}

int Minimum(int a, int b, int c)
{
    return min(min(a, b), c);
}

int EditDistanceDP(char X[], char Y[])
{
    int cost = 0;
    int leftCell, topCell, cornerCell;
    int m = strlen(X)+1;
    int n = strlen(Y)+1;
    int *T = (int *)malloc(m * n * sizeof(int));

    for(int i = 0; i < m; i++)
        for(int j = 0; j < n; j++)
            *(T + i * n + j) = SENTINEL;

    for(i = 0; i < m; i++)
        *(T + i * n) = i;

    for(j = 0; j < n; j++)
        *(T + j) = j;
```

```

for(i = 1; i < m; i++)
{
    for(int j = 1; j < n; j++)
    {
        leftCell = *(T + i*n + j-1);
        leftCell += EDIT_COST;
        topCell = *(T + (i-1)*n + j);
        topCell += EDIT_COST;
        cornerCell = *(T + (i-1)*n + (j-1) );
        cornerCell += (X[i-1] != Y[j-1]);
        *(T + (i)*n + (j)) = Minimum(leftCell, topCell, cornerCell);
    }
}

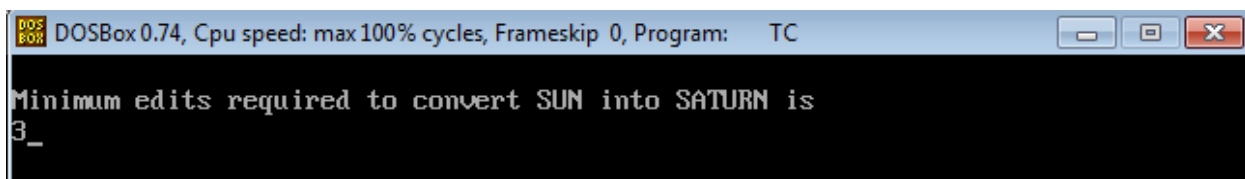
cost = *(T + m*n - 1);
free(T);
return cost;
}

void main()
{
    clrscr();
    char X[] = STRING_X;
    char Y[] = STRING_Y;

    cout<<"\nMinimum edits required to convert "<<X<<" into "<<Y<<" is
\n"<<EditDistanceDP(X, Y);

    getch();
}

```



```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Minimum edits required to convert SUN into SATURN is
3_

```

## **8.CODE AND ANALYZE TO FIND THE LONGEST COMMON SUBSEQUENCE BETWEEN TWO CHARACTER STRINGS USING DYNAMIC PROGRAMMING.**

```
#include<iostream.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
#define M 10
#define N 10

int max(int a, int b);

int lcs( char *X, char *Y, int m, int n )
{
    int L[M][N];
    int i, j;

    for (i=0; i<=m; i++)
    {
        for (j=0; j<=n; j++)
        {
            if (i == 0 || j == 0)
                L[i][j] = 0;

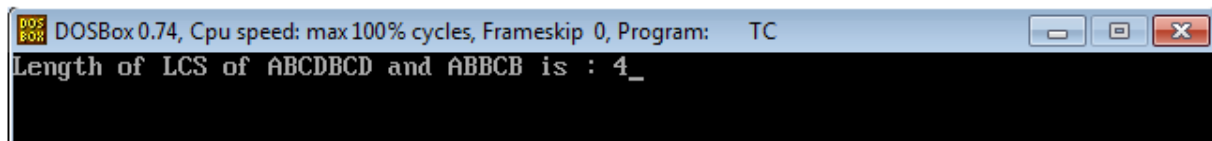
            else if (X[i-1] == Y[j-1])
                L[i][j] = L[i-1][j-1] + 1;

            else
                L[i][j] = max(L[i-1][j], L[i][j-1]);
        }
    }

    return L[m][n];
}

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

```
void main()
{
    clrscr();
    char X[] = "ABCDBCD";
    char Y[] = "ABBCB";
    int m = strlen(X);
    int n = strlen(Y);
    cout<<"Length of LCS of "<<X<<" and "<<Y<<" is : "<< lcs( X, Y, m, n ) ;
    getch();
}
```



## 9. CODE AND ANALYZE TO FIND AN OPTIMAL SOLUTION TO MATRIX CHAIN MULTIPLICATION USING DYNAMIC PROGRAMMING.

```
#include<iostream.h>
#include<conio.h>
#include<limits.h>
#define N 10

unsigned int MatrixChainOrder(int p[], int n)
{
    unsigned int m[N][N];
    int i, j, k, L, q;

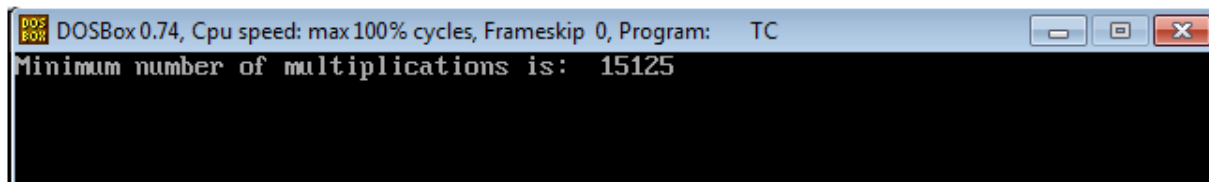
    for (i = 1; i < n; i++)
        m[i][i] = 0;

    for (L=2; L<n; L++)
    {
        for (i=1; i<=n-L+1; i++)
        {
            j = i+L-1;
            m[i][j] = INT_MAX;
            for (k=i; k<=j-1; k++)
            {
                q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }

    return m[1][n-1];
}

void main()
{
    clrscr();
    int arr[] = {30, 35, 15, 5,10,20,25};
    int size = sizeof(arr)/sizeof(arr[0]);
```

```
cout<<"Minimum number of multiplications is: "<<MatrixChainOrder(arr, size);  
getch();  
}
```



**10.CODE AND ANALYZE TO DO A BREADTH-FIRST SEARCH (BFS) ON AN UNDIRECTED GRAPH. IMPLEMENTING AN APPLICATION OF BFS SUCH AS TO FIND CONNECTED COMPONENTS OF AN UNDIRECTED GRAPH, OR TO CHECK WHETHER A GIVEN GRAPH IS BIPARTITE.**

//Breadth First Search - C Program Source Code

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>
/* maxVertices represents maximum number of vertices that can be present in the graph.
*/
#define maxVertices 100
/*Queue has five properties. capacity stands for the maximum number of elements Queue
can hold.
Size stands for the current size of the Queue and elements is the array of elements. front
is the
index of first element (the index at which we remove the element) and rear is the index
of last element
(the index at which we insert the element) */
typedef struct Queue
{
    int capacity;
    int size;
    int front;
    int rear;
    int *elements;
}Queue;
/* crateQueue function takes argument the maximum number of elements the Queue can
hold, creates
a Queue according to it and returns a pointer to the Queue. */
Queue * CreateQueue(int maxElements)
{
    /* Create a Queue */
    Queue *Q;
    Q = (Queue *)malloc(sizeof(Queue));
```



```

    /* Initialise its properties */
    Q->elements = (int *)malloc(sizeof(int)*maxElements);
    Q->size = 0;
    Q->capacity = maxElements;
    Q->front = 0;
    Q->rear = -1;
    /* Return the pointer */
    return Q;
}
void Dequeue(Queue *Q)
{
    /* If Queue size is zero then it is empty. So we cannot pop */
    if(Q->size==0)
    {
        printf("Queue is Empty\n");
        return;
    }
    /* Removing an element is equivalent to incrementing index of front by one */
    else
    {
        Q->size--;
        Q->front++;
        /* As we fill elements in circular fashion */
        if(Q->front==Q->capacity)
        {
            Q->front=0;
        }
    }
    return;
}
int Front(Queue *Q)
{
    if(Q->size==0)
    {
        printf("Queue is Empty\n");
        exit(0);
    }
    /* Return the element which is at the front*/
    return Q->elements[Q->front];
}

```

```

void Enqueue(Queue *Q,int element)
{
    /* If the Queue is full, we cannot push an element into it as there is no space for
it.*/
    if(Q->size == Q->capacity)
    {
        printf("Queue is Full\n");
    }
    else
    {
        Q->size++;
        Q->rear = Q->rear + 1;
        /* As we fill the queue in circular fashion */
        if(Q->rear == Q->capacity)
        {
            Q->rear = 0;
        }
        /* Insert the element in its rear side */
        Q->elements[Q->rear] = element;
    }
    return;
}

```

```

void Bfs(int graph[][maxVertices], int *size, int presentVertex,int *visited)
{
    visited[presentVertex] = 1;
    /* Iterate through all the vertices connected to the presentVertex and perform bfs
on those
vertices if they are not visited before */
    Queue *Q = CreateQueue(maxVertices);
    Enqueue(Q,presentVertex);
    while(Q->size)
    {
        presentVertex = Front(Q);
        printf("Now visiting vertex %d\n",presentVertex);
        Dequeue(Q);
        int iter;
        for(iter=0;iter<size[presentVertex];iter++)

```

```

        {
            if(!visited[graph[presentVertex][iter]])
            {
                visited[graph[presentVertex][iter]] = 1;
                Enqueue(Q,graph[presentVertex][iter]);
            }
        }
    }
    return;
}

```

}  
 /\* Input Format: Graph is directed and unweighted. First two integers must be number of  
 vertices and edges

which must be followed by pairs of vertices which has an edge between them. \*/

```

int main()
{
    clrscr();
    int
graph[maxVertices][maxVertices],size[maxVertices]={0},visited[maxVertices]={0};
    int vertices,edges,iter;
    /* vertices represent number of vertices and edges represent number of edges in the
graph. */
    scanf("%d%d",&vertices,&edges);
    int vertex1,vertex2;
    for(iter=0;iter<edges;iter++)
    {
        scanf("%d%d",&vertex1,&vertex2);
        assert(vertex1>=0 && vertex1<vertices);
        assert(vertex2>=0 && vertex2<vertices);
        graph[vertex1][size[vertex1]++] = vertex2;
    }
    int presentVertex;
    for(presentVertex=0;presentVertex<vertices;presentVertex++)
    {
        if(!visited[presentVertex])
        {
            Bfs(graph,size,presentVertex,visited);
        }
    }
    getch();
}

```

```
return 0;
```

```
}
```

## 11.CODE AND ANALYZE TO FIND SHORTEST PATHS IN A GRAPH WITH POSITIVE EDGE WEIGHTS USING DIJKSTRA'S ALGORITHM.

```
#include<iostream.h>
#include<conio.h>
#define N 7
#define INT_MAX 1024

int c[N][N],s[N],d[N];
void dj(int );
void main()
{
    clrscr();
    int i,j,n,min,u;
    cout<<"\nenter no of vertices in graph\n";
    cin>>n;
    int e,sv,ev;
    cout<<"\nenter no of edges in graph\n";
    cin>>e;

    for( i=1;i<=n;i++)
    {
        for( j=1;j<=n;j++)
        {
            c[i][j]=INT_MAX;
        }
        c[i][i]=0;
    }
    cout<<"\nenter starting vertex, terminal vertex and weight respectively:\n";
    for( i=1;i<=e;i++)
    {
        cout<<"\nEdge "<<i<<endl;
        cin>>sv>>ev;
        cin>>c[sv][ev];
    }
    cout<<"\ncost matrix is: \n";
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
```

```

        cout<<c[i][j]<<"\t";
    }
    cout<<endl;
}
    dj(n);
getch();
}

void dj(int n)
{
// logic starts here
    int v,u,i,j,min;
    cout<<"\nEnter the starting vertex: \n";
    cin>>v;

    for(i=1;i<=n;i++)
    {
        s[i]=0;
        d[i]=c[v][i];
    }
    s[v]=1;
    d[v]=0;

    for(i=2;i<n;i++)
    {
        min= INT_MAX;
        for(j=1;j<=n;j++)
        {
            if((s[j]==0)&&(min>=d[j]))
            {
                min=d[j];
                u=j;
            }
        }

        s[u]=1;
        for(j=1;j<=n;j++)
        {
            if((s[j]==0)&&(d[j]>d[u]+c[u][j]))
            {

```

```

                d[j]=d[u]+c[u][j];
            }
        }
    }

    cout<<"\nShortest paths from source vertex "<<v<<endl;
    for(i=1;i<=n;i++)
    {
        cout<<i<<"\t"<<d[i]<<endl;
    }
}

```

```

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
3
33

Edge 6
1
4
35

cost matrix is:
0      34      1024      35      1024
1024    0       43      1024      41
1024   1024      0       22      1024
1024   1024   1024      0       1024
1024   1024    33      1024      0

enter the starting vertex:
1

Shortest paths from source vertex 1
1      0
2      34
3      77
4      35
5      75

```

## 12. CODE AND ANALYZE TO FIND SHORTEST PATHS IN A GRAPH WITH ARBITRARY EDGE WEIGHTS USING BELLMAN FORD ALGORITHM.

```
#include<iostream.h>
#include<stdlib.h>
#include <string.h>
#include<conio.h>
#include <limits.h>
#define VV 5

struct Edge
{
    int src, dest, weight;
};

struct Graph
{
    int V, E;
    struct Edge* edge;
};

struct Graph* createGraph(int V, int E)
{
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph) );
    graph->V = V;
    graph->E = E;
    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );
    return graph;
}

void printArr(int dist[], int n)
{
    cout<<"\nVertex Distance from Source\n";
    for (int i = 0; i < n; ++i)
        cout<< i<<" "<<dist[i]<<endl;
}

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 (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] + weight < dist[v])
            dist[v] = dist[u] + weight;
    }
}

for (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] + weight < dist[v])
        cout<<"\n\nGraph contains negative weight cycle";
}

printArr(dist, V);
return;
}

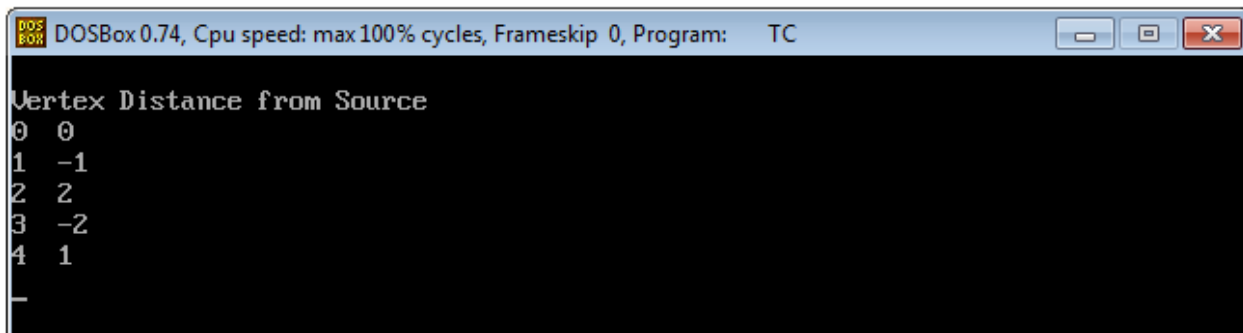
void main()
{
    clrscr();
    int V = 5;
    int E = 8;
    struct Graph* graph = createGraph(V, E);
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;

```

```
graph->edge[0].weight = -1;
graph->edge[1].src = 0;
graph->edge[1].dest = 2;
graph->edge[1].weight = 4;
graph->edge[2].src = 1;
graph->edge[2].dest = 2;
graph->edge[2].weight = 3;
graph->edge[3].src = 1;
graph->edge[3].dest = 3;
graph->edge[3].weight = 2;
graph->edge[4].src = 1;
graph->edge[4].dest = 4;
graph->edge[4].weight = 2;
graph->edge[5].src = 3;
graph->edge[5].dest = 2;
graph->edge[5].weight = 5;
graph->edge[6].src = 3;
graph->edge[6].dest = 1;
graph->edge[6].weight = 1;
graph->edge[7].src = 4;
graph->edge[7].dest = 3;
graph->edge[7].weight = -3;
```

```
BellmanFord(graph, 0);
getch();
```

```
}
```



```
DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

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

### 13. CODE AND ANALYZE TO FIND THE MINIMUM SPANNING TREE IN A WEIGHTED, UNDIRECTED GRAPH.

```
// Kruskal's algorithm to find Minimum Spanning Tree of a given connected,  
// undirected and weighted graph  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
// a structure to represent a weighted edge in graph  
struct Edge  
{  
    int src, dest, weight;  
};  
  
// a structure to represent a connected, undirected and weighted graph  
struct Graph  
{  
    // V-> Number of vertices, E-> Number of edges  
    int V, E;  
  
    // graph is represented as an array of edges. Since the graph is  
    // undirected, the edge from src to dest is also edge from dest  
    // to src. Both are counted as 1 edge here.  
    struct Edge* edge;  
};  
  
// Creates a graph with V vertices and E edges  
struct Graph* createGraph(int V, int E)  
{  
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph) );  
    graph->V = V;  
    graph->E = E;  
  
    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );  
  
    return graph;  
}  
  
// A structure to represent a subset for union-find
```

```

struct subset
{
    int parent;
    int rank;
};

// A utility function to find set of an element i
// (uses path compression technique)
int find(struct subset subsets[], int i)
{
    // find root and make root as parent of i (path compression)
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}

// A function that does union of two sets of x and y
// (uses union by rank)
void Union(struct subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    // Attach smaller rank tree under root of high rank tree
    // (Union by Rank)
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    // If ranks are same, then make one as root and increment
    // its rank by one
    else
    {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

```

```

// Compare two edges according to their weights.
// Used in qsort() for sorting an array of edges
int myComp(const void* a, const void* b)
{
    struct Edge* a1 = (struct Edge*)a;
    struct Edge* b1 = (struct Edge*)b;
    return a1->weight > b1->weight;
}

// The main function to construct MST using Kruskal's algorithm
void KruskalMST(struct Graph* graph)
{
    int V = graph->V;
    struct Edge result[V]; // Tnis will store the resultant MST
    int e = 0; // An index variable, used for result[]
    int i = 0; // An index variable, used for sorted edges

    // Step 1: Sort all the edges in non-decreasing order of their weight
    // If we are not allowed to change the given graph, we can create a copy of
    // array of edges
    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);

    // Allocate memory for creating V subsets
    struct subset *subsets =
        (struct subset*) malloc( V * sizeof(struct subset) );

    // Create V subsets with single elements
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    // Number of edges to be taken is equal to V-1
    while (e < V - 1)
    {
        // Step 2: Pick the smallest edge. And increment the index
        // for next iteration
        struct Edge next_edge = graph->edge[i++];
    }
}

```

```

int x = find(subsets, next_edge.src);
int y = find(subsets, next_edge.dest);

// If including this edge doesn't cause cycle, include it
// in result and increment the index of result for next edge
if (x != y)
{
    result[e++] = next_edge;
    Union(subsets, x, y);
}
// Else discard the next_edge
}

// print the contents of result[] to display the built MST
printf("Following are the edges in the constructed MST\n");
for (i = 0; i < e; ++i)
    printf("%d -- %d == %d\n", result[i].src, result[i].dest,
           result[i].weight);

return;
}

// Driver program to test above functions
int main()
{
    /* Let us create following weighted graph
        10
        0-----1
        | \   |
        6|  5\ |15
        |   \ |
        2-----3
        4      */
    int V = 4; // Number of vertices in graph
    int E = 5; // Number of edges in graph
    struct Graph* graph = createGraph(V, E);

    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;

```

```
graph->edge[0].weight = 10;

// add edge 0-2
graph->edge[1].src = 0;
graph->edge[1].dest = 2;
graph->edge[1].weight = 6;

// add edge 0-3
graph->edge[2].src = 0;
graph->edge[2].dest = 3;
graph->edge[2].weight = 5;

// add edge 1-3
graph->edge[3].src = 1;
graph->edge[3].dest = 3;
graph->edge[3].weight = 15;

// add edge 2-3
graph->edge[4].src = 2;
graph->edge[4].dest = 3;
graph->edge[4].weight = 4;

KruskalMST(graph);

return 0;
}
```

#### 14. CODE AND ANALYZE TO FIND ALL OCCURRENCES OF A PATTERN P IN A GIVEN STRING S. (KMP)

```
#include<iostream.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<conio.h>
void computeLPSArray(char *pat, int M, int *lps);

void KMPSearch(char *pat, char *txt)
{
int M = strlen(pat);
int N = strlen(txt);

int *lps = (int *)malloc(sizeof(int)*M);
int j = 0;

computeLPSArray(pat, M, lsp);

int i = 0;
while(i < N)
{
    if(pat[j] == txt[i])
    {
        j++;
        i++;
    }

    if (j == M)
    {
        cout<<"\nFound pattern at index: "<< i-j<<endl;
        j = lsp[j-1];
    }

    else if(pat[j] != txt[i])
    {
        if(j != 0)
            j = lsp[j-1];
        else

```



```

        i = i+1;
    }
}
free(lps);
}

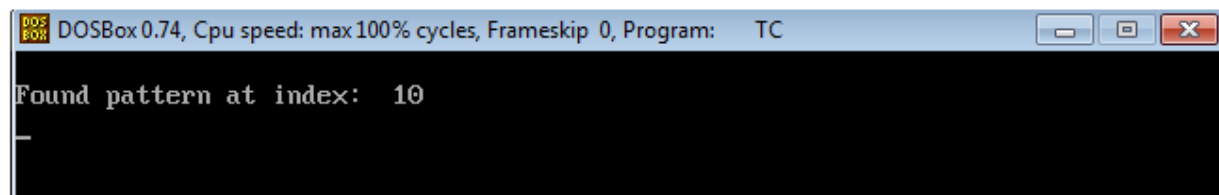
void computeLPSArray(char *pat, int M, int *lps)
{
    int len = 0;
    int i;

    lps[0] = 0;
    i = 1;

    while(i < M)
    {
        if(pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if( len != 0 )
            {
                len = lps[len-1];
            }
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
}

```

```
void main()
{
clrscr();
char *txt = "ABABDABACDABABCABAB";
char *pat = "ABABCABAB";
KMPSearch(pat, txt);
getch();
}
```



## **15. CODE AND ANALYZE TO MULTIPLY TWO LARGE INTEGERS USING KARATSUBA ALGORITHM.**

The above algorithm is called Karatsuba algorithm and it can be used for any base.

Following is C++ implementation of above algorithm.

// C++ implementation of Karatsuba algorithm for bit string multiplication.

```
#include<iostream>
```

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

```
usingnamespacestd;
```

```
// FOLLOWING TWO FUNCTIONS ARE COPIED FROM http://goo.gl/q0OhZ
```

```
// Helper method: given two unequal sized bit strings, converts them to
```

```
// same length by adding leading 0s in the smaller string. Returns the
```

```
// the new length
```

```
intmakeEqualLength(string &str1, string &str2)
```

```
{
```

```
    intlen1 = str1.size();
```

```
    intlen2 = str2.size();
```

```
    if(len1 < len2)
```

```
    {
```

```
        for(inti = 0 ; i < len2 - len1 ; i++)
```

```
            str1 = '0'+ str1;
```

```

        return len2;
    }
    elseif(len1 > len2)
    {
        for(int i = 0 ; i < len1 - len2 ; i++)
            str2 = '0' + str2;
    }
    return len1; // If len1 >= len2
}

// The main function that adds two bit sequences and returns the addition
string addBitStrings( string first, string second )
{
    string result; // To store the sum bits

    // make the lengths same before adding
    int length = makeEqualLength(first, second);
    int carry = 0; // Initialize carry

    // Add all bits one by one
    for(int i = length-1 ; i >= 0 ; i--)
    {

```

```
intfirstBit = first.at(i) - '0';
```

```
intsecondBit = second.at(i) - '0';
```

```
// boolean expression for sum of 3 bits
```

```
intsum = (firstBit ^ secondBit ^ carry)+'0';
```

```
result = (char)sum + result;
```

```
// boolean expression for 3-bit addition
```

```
carry = (firstBit&secondBit) | (secondBit&carry) | (firstBit&carry);
```

```
}
```

```
// if overflow, then add a leading 1
```

```
if(carry) result = '1'+ result;
```

```
returnresult;
```

```
}
```

```
// A utility function to multiply single bits of strings a and b
```

```
intmultiplySingleBit(string a, string b)
```

```
{ return(a[0] - '0')*(b[0] - '0'); }
```

```

// The main function that multiplies two bit strings X and Y and returns
// result as long integer
longintmultiply(string X, string Y)
{
    // Find the maximum of lengths of x and Y and make length
    // of smaller string same as that of larger string
    intn = makeEqualLength(X, Y);

    // Base cases
    if(n == 0) return0;
    if(n == 1) returnmultiplySingleBit(X, Y);

    intfh = n/2; // First half of string, floor(n/2)
    intsh = (n-fh); // Second half of string, ceil(n/2)

    // Find the first half and second half of first string.
    // Refer http://goo.gl/ILmgn for substr method
    string Xl = X.substr(0, fh);
    string Xr = X.substr(fh, sh);

    // Find the first half and second half of second string
    string Yl = Y.substr(0, fh);

```

```

string Yr = Y.substr(fh, sh);

// Recursively calculate the three products of inputs of size n/2
longintP1 = multiply(Xl, Yl);
longintP2 = multiply(Xr, Yr);
longintP3 = multiply(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));

// Combine the three products to get the final result.
return P1*(1<<(2*sh)) + (P3 - P1 - P2)*(1<<sh) + P2;
}

// Driver program to test above functions
int main()
{
    printf("%ld\n", multiply("1100", "1010"));
    printf("%ld\n", multiply("110", "1010"));
    printf("%ld\n", multiply("11", "1010"));
    printf("%ld\n", multiply("1", "1010"));
    printf("%ld\n", multiply("0", "1010"));
    printf("%ld\n", multiply("111", "111"));
    printf("%ld\n", multiply("11", "11"));
}

```

## 16.CODE AND ANALYZE TO COMPUTE THE CONVEX HULL OF A SET OF POINTS IN THE PLANE.

(Graham Scan)

Following is++ C++ implementation of the above algorithm.

```
// A C++ program to find convex hull of a set of points
// Refer http://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/
// for explanation of orientation()

#include <iostream>
#include <stack>
#include <stdlib.h>
using namespace std;

struct Point
{
    int x;
    int y;
};

// A global point needed for sorting points with reference to the first point
// Used in compare function of qsort()
Point p0;
```



```
// A utility function to find next to top in a stack
```

```
Point nextToTop(stack<Point>&S)
```

```
{  
    Point p = S.top();  
    S.pop();  
    Point res = S.top();  
    S.push(p);  
    return res;  
}
```

```
// A utility function to swap two points
```

```
int swap(Point &p1, Point &p2)
```

```
{  
    Point temp = p1;  
    p1 = p2;  
    p2 = temp;  
}
```

```
// A utility function to return square of distance between p1 and p2
```

```
int dist(Point p1, Point p2)
```

```
{  
    return (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y);  
}
```

```

}

// To find orientation of ordered triplet (p, q, r).
// The function returns following values
// 0 --> p, q and r are colinear
// 1 --> Clockwise
// 2 --> Counterclockwise

intorientation(Point p, Point q, Point r)
{
    intval = (q.y - p.y) * (r.x - q.x) -
            (q.x - p.x) * (r.y - q.y);

    if(val == 0) return 0; // colinear
    return (val > 0)? 1: 2; // clock or counterclock wise
}

// A function used by library function qsort() to sort an array of
// points with respect to the first point
intcompare(constvoid*vp1, constvoid*vp2)
{
    Point *p1 = (Point *)vp1;
    Point *p2 = (Point *)vp2;

```

```

// Find orientation
into = orientation(p0, *p1, *p2);
if(o == 0)
    return(dist(p0, *p2) >= dist(p0, *p1))? -1 : 1;

return(o == 2)? -1: 1;
}

// Prints convex hull of a set of n points.
voidconvexHull(Point points[], intn)
{
    // Find the bottommost point
    intymin = points[0].y, min = 0;
    for(inti = 1; i < n; i++)
    {
        inty = points[i].y;

        // Pick the bottom-most or chose the left most point in case of tie
        if((y < ymin) || (ymin == y && points[i].x < points[min].x))
            ymin = points[i].y, min = i;
    }
}

```

```

// Place the bottom-most point at first position
swap(points[0], points[min]);

// Sort n-1 points with respect to the first point. A point p1 comes
// before p2 in sorted output if p2 has larger polar angle (in
// counterclockwise direction) than p1
p0 = points[0];
qsort(&points[1], n-1, sizeof(Point), compare);

// Create an empty stack and push first three points to it.
stack<Point> S;
S.push(points[0]);
S.push(points[1]);
S.push(points[2]);

// Process remaining n-3 points
for(int i = 3; i < n; i++)
{
    // Keep removing top while the angle formed by points next-to-top,
    // top, and points[i] makes a non-left turn
    while(orientation(nextToTop(S), S.top(), points[i]) != 2)

```

```

        S.pop();
    S.push(points[i]);
}

// Now stack has the output points, print contents of stack
while(!S.empty())
{
    Point p = S.top();
    cout << "(" << p.x << ", " << p.y << ")" << endl;
    S.pop();
}
}

// Driver program to test above functions
int main()
{
    Point points[] = { {0, 3}, {1, 1}, {2, 2}, {4, 4},
                       {0, 0}, {1, 2}, {3, 1}, {3, 3} };

    int n = sizeof(points)/sizeof(points[0]);

    convexHull(points, n);

    return 0;
}

```

**17. (Mini-project Topic)** Program to multiply two polynomials using Fast Fourier Transform (FFT).

[illegible]