PROGRAM NO -1

OBJECTIVE:

To implement linear search algorithm, Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

PROCEDURE:

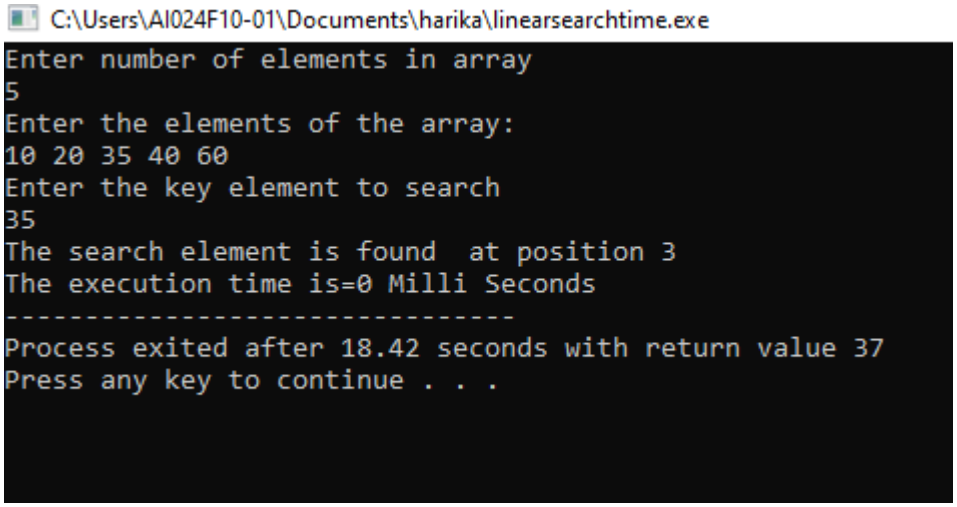
1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include <stdio.h>
#include<time.h>
#include<stdlib.h>
#include<conio.h>
int linearSearch(int a[],int n,int key)
{
int i;
for(i=0;i<n;i++)
{
if(key==a[i])
return i;
}
return -1;
}
void main()
{
 char ch;
 int a[100],n,key,res,i;
 clock_t st,et;
  double time_taken;
  clrscr();
  printf("Enter number of elements in array \n");
  scanf("%d", &n);
  printf("Enter the elements of the array: \n");
  for(i=0;i<n;i++)
  scanf("%d",&a[i]);
  printf("Enter the key element to search\n");
  scanf("%d", &key);
   st=clock();
   res=linearSearch(a,n,key);
   et=clock();
   time_taken=(((double)(et-st))/CLOCKS_PER_SEC)*1000;
```

```
if(res==-1)

{
     printf("The search element is not found \n");
    printf("The execution time is=%.0f Milli Seconds",time_taken);
     getch();
     exit(0);

}
else
     printf("The search element is found  at position %d\n",res+1);
     printf("The execution time is=%.0f Milli Seconds",time_taken);
     getch();
}
```

OUTPUT:



PROGRAM NO -2

OBJECTIVE:

To implement binary search algorithm. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

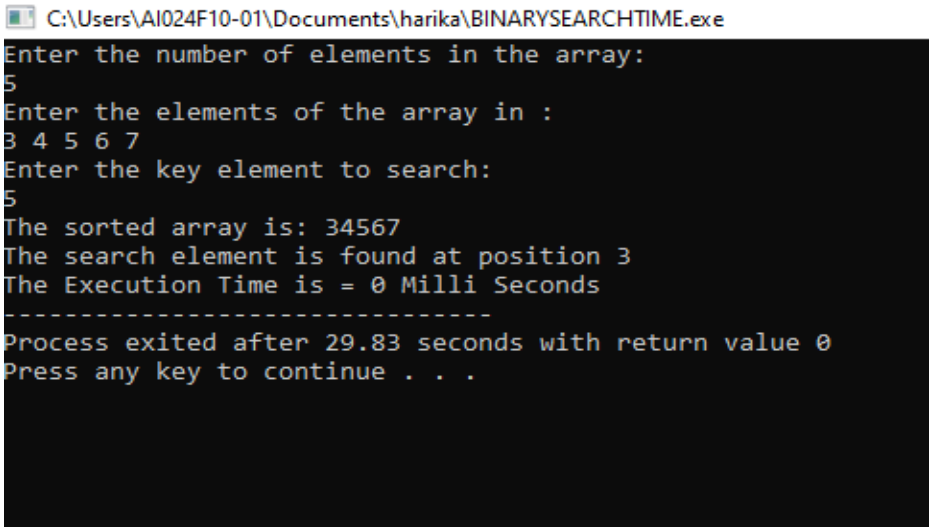
SOURCE CODE:

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
#include<conio.h>
int binarySearch(int a[],int key,int n,int first,int last)
{
int mid,i,j,temp;
if(last<first)
 return -1;
for(i=0;i<=n-2;i++)
{
for(j=0;j<=n-2;j++)
{
if(a[j+1]<a[j])
{
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}
}
}
while(first<=last)
{
mid=(first+last)/2;
```

```
if(key==a[mid])
    return mid+1;
else if(key<a[mid])
     last=mid-1;
else
     first=mid+1;
}
 return -1;
}
int main()
{
char ch;
int a[100],n, key, i, res, first, last;
clock_t st,et;
double time_taken;
clrscr();
printf("Enter the number of elements in the array: \n");
scanf("%d",&n);
printf("Enter the elements of the array in :\n");
 for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("Enter the key element to search: \n");
 scanf("%d",&key);
first=0;
last=n-1;
st=clock();
// Record Start Time
res=binarySearch(a,key,n,first,last);
printf("The sorted array is: ");
for(i=0;i<n;i++)
printf("%d",a[i]);
et=clock();
// Record End time
 time_taken = (((double) (et - st)) / CLOCKS_PER_SEC)*1000;
```

```
if(res ==-1)
{
printf("\nThe search element is not found\n");
printf("The Execution Time is = %.0f Milli Seconds",time_taken);
exit(0);
}
else
printf("\nThe search element is found at position %d\n",res);
printf("The Execution Time is = %.0f Milli Seconds",time_taken);
return 0;
}
```

OUTPUT:



PROGRAM NO -3

OBJECTIVE:

To solve towers of honai problem and execute it for different number of disks

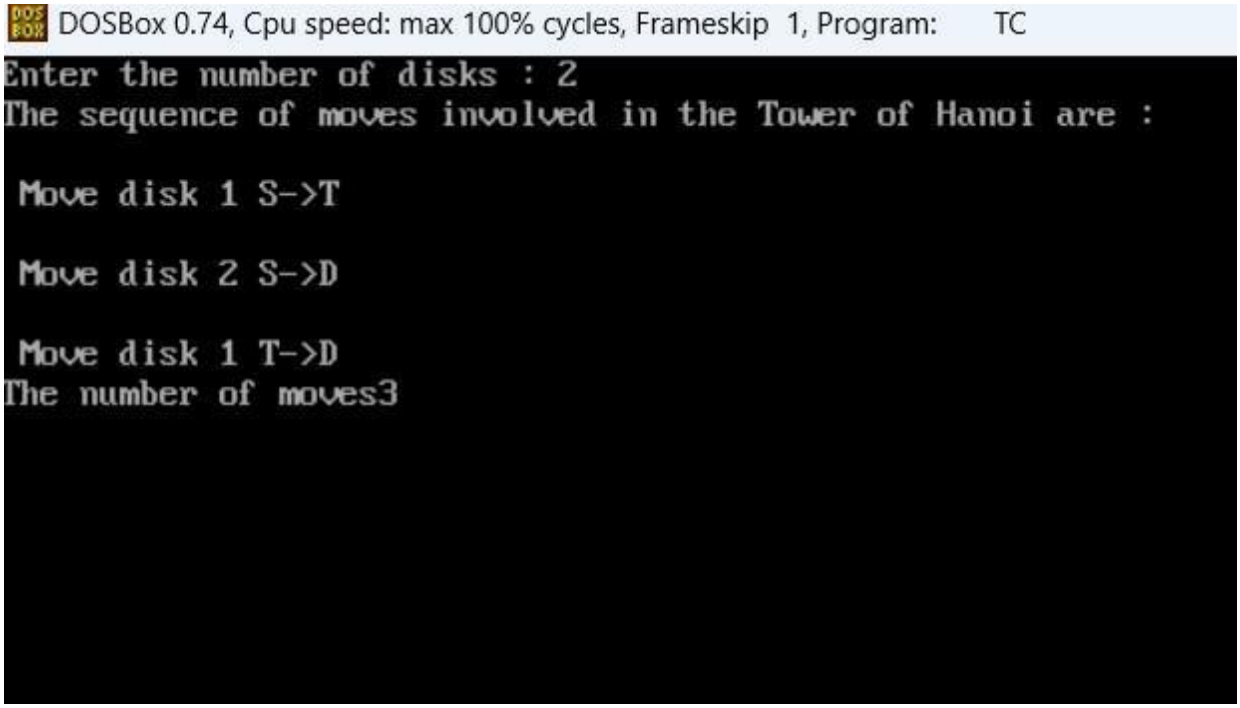
PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include <stdio.h>
#include<conio.h>
void toh(int, char, char, char);
int count=0;
int main()
{
char source='S',temp='T',dest='D';
int n;
clrscr();
printf("Enter the number of disks : ");
scanf("%d", &n);
printf("The sequence of moves involved in the Tower of Hanoi are :\n");
toh(n, source, temp,dest);
printf("The number of moves%d",count);
return 0;
}
void toh(int n, char source, char temp, char dest)
{
// Base Condition if no of disks are
if (n > 0)
{
toh(n-1, source,dest,temp);
printf("\n Move disk %d %c->%c\n", n,source,dest);
count++;
toh(n-1, temp,source,dest);
}
```

OUTPUT:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 1, Program: TC
Enter the number of disks : 2
The sequence of moves involved in the Tower of Hanoi are :
Move disk 1 S->T
Move disk 2 S->D
Move disk 1 T->D
The number of moves3

PROGRAM NO -4

OBJECTIVE:

To Sort a given set of numbers using selection sort algorithm. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

PROCEDURE:

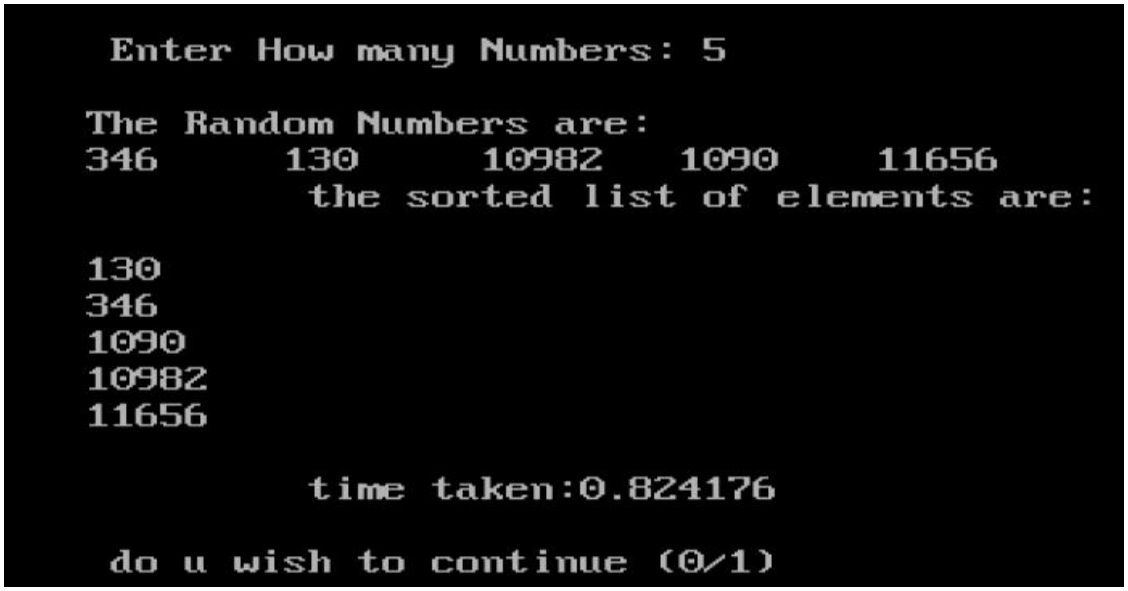
1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<conio.h>
void main()
{
int i,n,j,min,k,a[20],ch=1;
clock_t begin,end;
clrscr();
while(ch)
{
printf("\n Enter How many Numbers: ");
scanf("%d", &n);
printf("\nThe Random Numbers are:\n");
for(k=0; k<n; k++)
{
a[k]=rand();
printf("%d\t",a[k]);
}
begin=clock();
for(k=0;k<n-1;k++)
{
min=k;
```

```
delay(200);
for(j=k+1;j<n;j++)
{
  if(a[j]<a[min])
          min=j;
}
i=a[k];
a[k]=a[min];
a[min]=i;
}
end=clock();
printf("\n\t the sorted list of elements are:\n");
for(k=0;k<n;k++)
printf("\n%d",a[k]);
printf("\n\n\t time taken:%lf",(end-begin)/CLK_TCK);
printf("\n\n do u wish to continue (0/1)\n");
scanf("%d",&ch);
}
getch();
}
```

OUTPUT:



PROGRAM NO -5

OBJECTIVE:

To find the value of an (where a and n are integers) using both brute-force based algorithm and divide and conquer based algorithm.

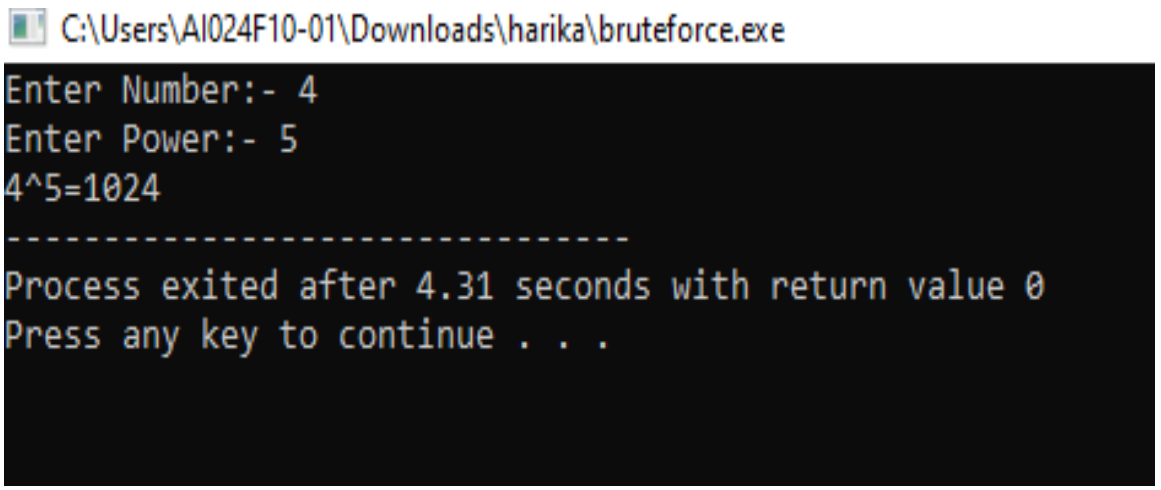
PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
long int power(int x,int n)
   {
   if(n==0)
   {
   return 1;
   }
   else if(n%2==0)
   {
   return power(x,n/2)*power(x,n/2);
   }
   else
   {
   return x*power(x,(n-1)/2)*power(x,(n-1)/2);
   }
   }
   int main()
   {
  int x,y;
  clrscr();
   printf("Enter Number:- “);
   scanf("%d",&x);
   printf("Enter Power:- ");
   scanf("%d",&y);
   printf("%d^%d=%d",x,y,power(x,y);
   return 0;
   }
```

OUTPUT:


C:\Users\AI024F10-01\Downloads\harika\bruteforce.exe
Enter Number:- 4
Enter Power:- 5
4^5=1024

Process exited after 4.31 seconds with return value 0
Press any key to continue . . .

PROGRAM NO -6

OBJECTIVE:

To Sort a given set of elements using quick sort algorithm. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include <time.h>
void quicksort(int A[10], int low, int high)
   {
   int i,j,pivot,temp;
   if (low < high)
   {
pivot=low;
i=low; j=high;
while (i<j)
   {
while(A[i]<=A[pivot]&&i<high)
i++;
   while (A[j]>A[pivot])
   j--;
   if (i<j)
   {
   temp = A[i];
   A[i] = A[j];
   A[j] = temp;
   }
   }
temp =A[pivot];
A[pivot] = A[j];
A[j] = temp;
   quicksort(A,low,j-1);
   quicksort(A,j+1,high);
   }
   }
```

```
int main()
   {
   int i, n,
   A[10];
   clock_t st, et;
   double time_taken;
   clrscr();
   printf("Enter the number of elements of array:\n");
   scanf("%d", &n);
   printf("Enter the elements of the array: \n");
   for (i=0; i< n; i++)
   scanf(" %d", &A[i]);
   st = clock();
   quicksort(A, 0, n 1);
   et = clock();
 time_taken = (((double) (et - st)) /CLOCKS_PER_SEC)*1000;
  printf("Sorted list of elements:");
   for (i=0; i<n; i++)
   printf(" %d ", A[i]);
   printf("The Execution Time is = %.0f Milli  Seconds",time_taken);
   return 0;
   getch();
   }
```

OUTPUT:

```
Enter the number of elements of array:
5
Enter the elements of the array:
3 2 1 6 9
Sorted list of elements: 1  2  3  6  9 The Execution Time is = 0 Milli Seconds
--------------------------------
Process exited after 9.496 seconds with return value 0
Press any key to continue . . .
```

PROGRAM NO -7

OBJECTIVE:

To find the binomial co-efficient C(n, k), [where n and k are integers and n > k] using brute force based algorithm and also dynamic programming based algorithm.

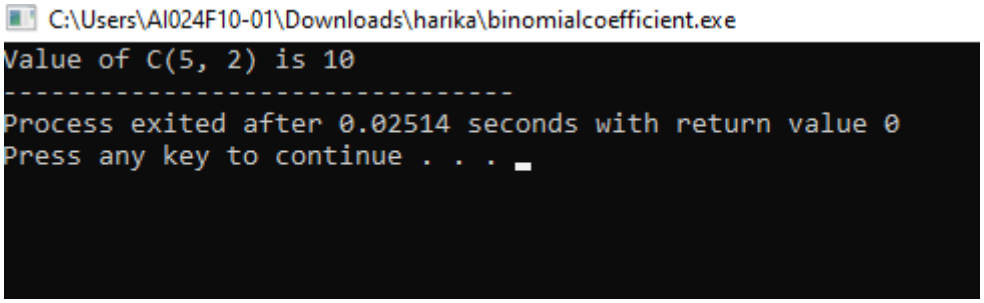
PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include <stdio.h>
#include<conio.h>
 // Returns value of Binomial Coefficient C(n, k)
int binomialCoeff(int n, int k)
{
   // Base Casesif (k > n)
      return 0;
   if (k == 0 || k == n)return 1;

   // Recur
   return binomialCoeff(n - 1, k - 1)+ binomialCoeff(n - 1, k);
}
 /* Driver program to test above
function*/int main()
{
   int n = 5, k= 2;
   clrscr();
   printf("Value of C(%d, %d) is %d ", n, k,
   binomialCoeff(n, k));
   return 0;
}
```

OUTPUT:


C:\Users\AI024F10-01\Downloads\harika\binomialcoefficient.exe
Value of C(5, 2) is 10

Process exited after 0.02514 seconds with return value 0
Press any key to continue . . .

PROGRAM NO -8

OBJECTIVE:

To implement Floyd's algorithm and find the lengths of the shortest paths fromevery pairs of vertices in a given weighted graph.

PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
  int cost[10][10],a[10][10];
  void all_paths(int [10][10],int [10][10],int);
  int min1(int,int);
  int main()
  {
  int i,j,n;
  printf("\n enter the number of vertices\n");
  scanf("%d",&n);
  printf("\n enter the adjacency matrix\n");
  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  scanf("%d",&cost[i][j]);
  all_paths(cost,a,n);
  printf("\n\t the shortest path obtained is\n");
  for(i=1;i<=n;i++)
  {
  for(j=1;j<=n;j++)
  printf("\t %d",a[i][j]);
  printf("\n");
  }
```

```
return 0;
}
void all_paths(int cost[10][10],int a[10][10],int n)
{
int i,j,k; for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
a[i][j]=cost[i][j];
for(k=1;k<=n;k++)
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
a[i][j]=min1(a[i][j],a[i][k]+a[k][j]);
}
int min1(int a,int b)
{
return(a<b)?a:b;
}
```

OUTPUT:

```
 enter the number of vertices
4

 enter the adjacency matrix
999 999 3 999
2 999 999 999
999 7 999 1
6 999 999 999

         the shortest path obtained is
         10      10      3       4
         2       12      5       6
         7       7       10      1
         6       16      9       10

--------------------------------
Process exited after 33.26 seconds with return value 0
Press any key to continue . . .
```

PROGRAM NO -9

OBJECTIVE:

To evaluate a polynomial using brute-force based algorithm and using Horner's rule and compare their performances.

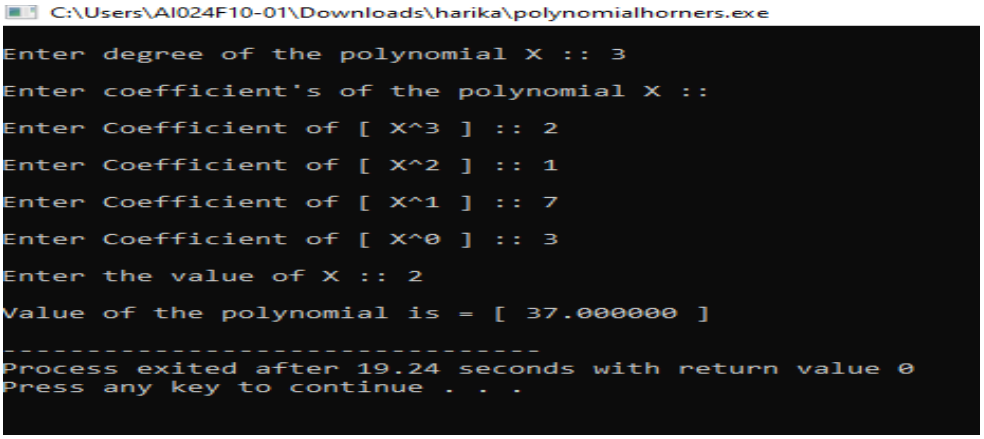
PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

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

   int main()
   {
       float a[100],sum=0,x;
       int n,i;
       clrscr();
       printf("\nEnter degree of the polynomial X :: ");
       scanf("%d",&n);
       printf("\nEnter coefficient's of the polynomial X :: \n");
       for(i=n;i>=0;i--)
       {
            printf("\nEnter Coefficient of [ X^%d ] :: ",i);
            scanf("%f",&a[i]);
       }
            printf("\nEnter the value of X :: ");
            scanf("%f",&x);

       for(i=n;i>0;i--)
       {
         sum=(sum+a[i])*x;
       }
   sum=sum+a[0];
   printf("\nValue of the polynomial is = [ %f ]\n",sum);
     return 0;
}
```

OUTPUT:


C:\Users\AI024F10-01\Downloads\harika\polynomialhorners.exe
Enter degree of the polynomial X :: 3
Enter coefficient's of the polynomial X ::
Enter Coefficient of [X^3] :: 2
Enter Coefficient of [X^2] :: 1
Enter Coefficient of [X^1] :: 7
Enter Coefficient of [X^0] :: 3
Enter the value of X :: 2
Value of the polynomial is = [37.000000]
Process exited after 19.24 seconds with return value 0
Press any key to continue . . .

PROGRAM NO -10

OBJECTIVE:

To solve the string-matching problem using Boyer-Moore approach.

PROCEDURE:

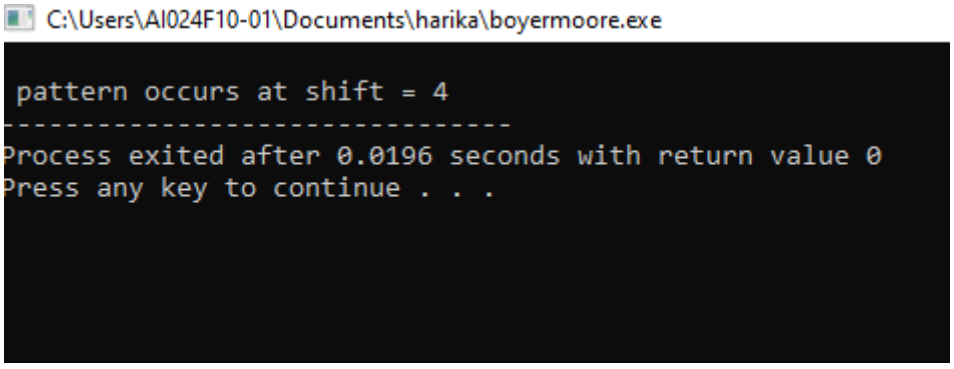
1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
# include <limits.h>
# include <string.h>
# include <stdio.h>
#include<conio.h>
# define NO_OF_CHARS 256

// A utility function to get maximum of two integers
int max(int a, int b)
{
   return (a > b) ? a : b;
}

// The preprocessing function for Boyer Moore's bad character heuristic
void badCharHeuristic(char *str, int size, int badchar[NO_OF_CHARS])
{
   int i;

   // Initialize all occurrences as -1
   for (i = 0; i < NO_OF_CHARS; i++)
      badchar[i] = -1;

   // Fill the actual value of last occurrence of a character
   for (i = 0; i < size; i++)
      badchar[(int) str[i]] = i;
}

void search(char *txt, char *pat)
 {int m = strlen(pat);
   int n = strlen(txt);

   int badchar[NO_OF_CHARS];

   int s = 0; // s is shift of the pattern with respect to text

   badCharHeuristic(pat, m, badchar);
```

```
    while (s <= (n - m))
    {
      int j = m - 1;


while (j >= 0 && pat[j] == txt[s + j])j--;

      if (j < 0) { printf("\n pattern occurs at shift = %d", s);

        s += (s + m < n) ? m - badchar[txt[s + m]] : 1;

      }

      else
        s += max(1, j - badchar[txt[s + j]]);
    }
  }

  int main() {
    char txt[] = "ABAAABCD";
    char pat[] = "ABC";
    search(txt, pat);
    return 0;
  }
```

OUTPUT:



PROGRAM NO -11

OBJECTIVE:

To solve the string-matching problem using KMP algorithm.

PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
void computeLPSArray(char *pat, int M, int *lps);
void KMPSearch(char *pat, char *txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    // create lps[] that will hold the longest prefix suffix values for pattern
    int *lps = (int *) malloc(sizeof(int) * M);
    int j = 0; // index for pat[]
    int i = 0; // index for txt[]

    // Preprocess the pattern (calculate lps[] array)
    computeLPSArray(pat, M, lps);
    while (i < N)
  {
      if (pat[j] == txt[i])
  {
        j++;
        i++;
      }

      if (j == M)
  {
```

```
            printf("Found pattern at index %d \n", i - j);
            j = lps[j - 1];
        }

        // mismatch after j matches
        else if (i < N && pat[j] != txt[i])
 {
            // Do not match lps[0..lps[j-1]] characters,
            // they will match anyway
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
    free(lps); // to avoid memory leak
}

void computeLPSArray(char *pat, int M, int *lps)
 {
    int len = 0; // lenght of the previous longest prefix suffix
    int i;

    lps[0] = 0; // lps[0] is always 0
    i = 1;

    // the loop calculates lps[i] for i = 1 to M-1
    while (i < M)
 {
        if (pat[i] == pat[len])
{
            len++;
            lps[i] = len;
            i++;
        } else // (pat[i] != pat[len])
        {
            if (len != 0)
```

```
{
            // This is tricky. Consider the example AAACAAAA and i = 7.
            len = lps[len - 1];

            // Also, note that we do not increment i here
          } else // if (len == 0)
          {
            lps[i] = 0;
            i++;
          }
        }
    }
}

// Driver program to test above function
int main()
 {
    char *txt = "ABABDABACDABABCABAB";
    char *pat = "ABABCABAB";
    KMPSearch(pat, txt);
    return 0;
getch();
}
```

OUTPUT:

PROGRAM NO -12

OBJECTIVE:

To implement BFS traversal algorithm

PROCEDURE:

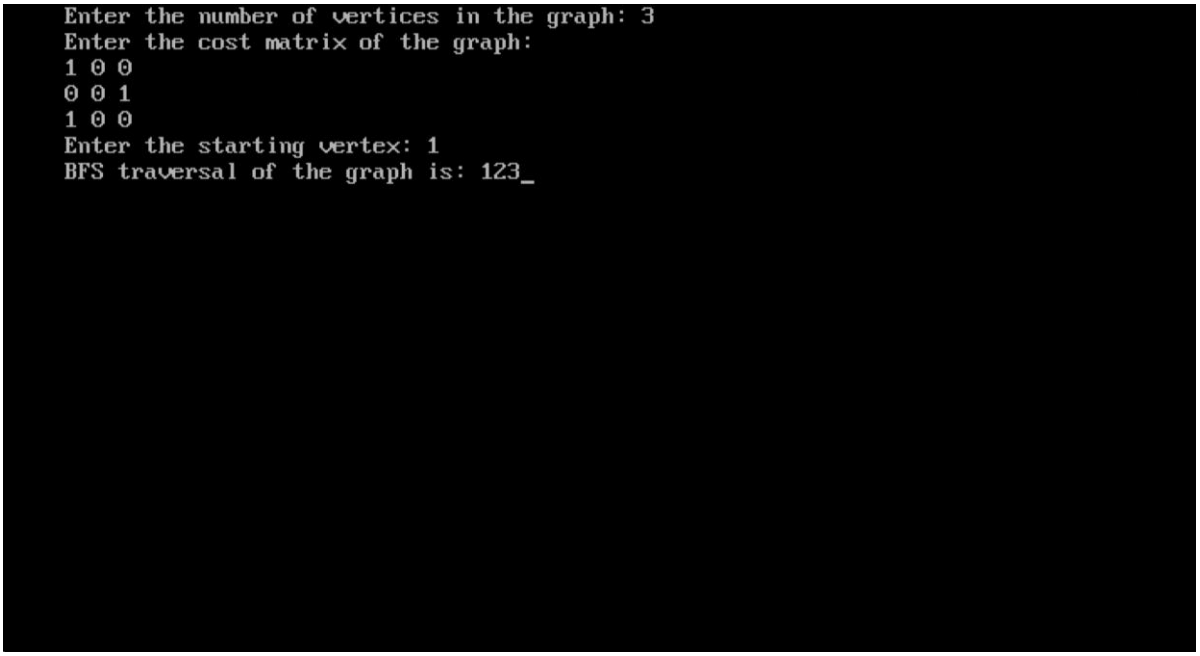
1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#include<conio.h>
int c[MAX][MAX];
int visited [MAX];
int queue [MAX];
int n; // Number of vertices
void BFS (int v)
{
int front = 0, rear=-1, i;
visited[v] = 1;
queue[++rear] = v;
while (front <= rear)
{
v = queue[front++];
printf("%d", v);
for ( i=1 ; i <= n ;i++)
{
if( c[v][1] ==1 && visited [i] ==0)
{
queue[++rear]= i;
visited [i] = 1;
}
}
}
}
int main()
{
int i, j, v;
clrscr();
printf("Enter the number of vertices in the graph: ");
scanf("%d", &n);
printf("Enter the cost matrix of the graph: \n");
for( i = 1; i <= n ;i++)
```

```
{
for ( j = 1;j <= n;j++)
scanf("%d",&c[i][j]);
}
for(i = 1; i <= n; i++)
visited[i] = 0;
printf("Enter the starting vertex: ");
scanf("%d", &v);
printf("BFS traversal of the graph is: ");
BFS(v);
return 0;
}
```

OUTPUT:



PROGRAM NO -13

OBJECTIVE:

To find the minimum spanning tree of a given graph using Prim‟s algorithm

PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]= {0},min,mincost=0,cost[10][10];
void main()
{
clrscr();
printf("\n Enter the number of nodes:");
scanf("%d",&n);
printf("\n Enter the 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;
}
   visited[1]=1;
   printf("\n");
   while(ne<n)
   {
   for(i=1,min=999;i<=n;i++)
   for(j=1;j<=n;j++)
   if(cost[i][j]<min)
```

```
if(visited[i]!=0)
{
min=cost[i][j];a=u=i;
b=v=j;
}
if(visited[u]==0 || visited[v]==0)
{
printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
mincost+=min;
visited[b]=1;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n Minimun cost=%d",mincost);
getch();
}
```

OUTPUT:

```
 Enter the number of nodes:4

 Enter the adjacency matrix:
0 2 9 1
2 5 1 11
7 4 1 22
11 4 7 2


 Edge 1:(1 4) cost:1
 Edge 2:(1 2) cost:2
 Edge 3:(2 3) cost:1
 Minimun cost=4
```

PROGRAM NO -14

OBJECTIVE:

To obtain the topological ordering of vertices in a given digraph. Compute the transitive closure of a given directed graph using Warshall's algorithm.

PROCEDURE:

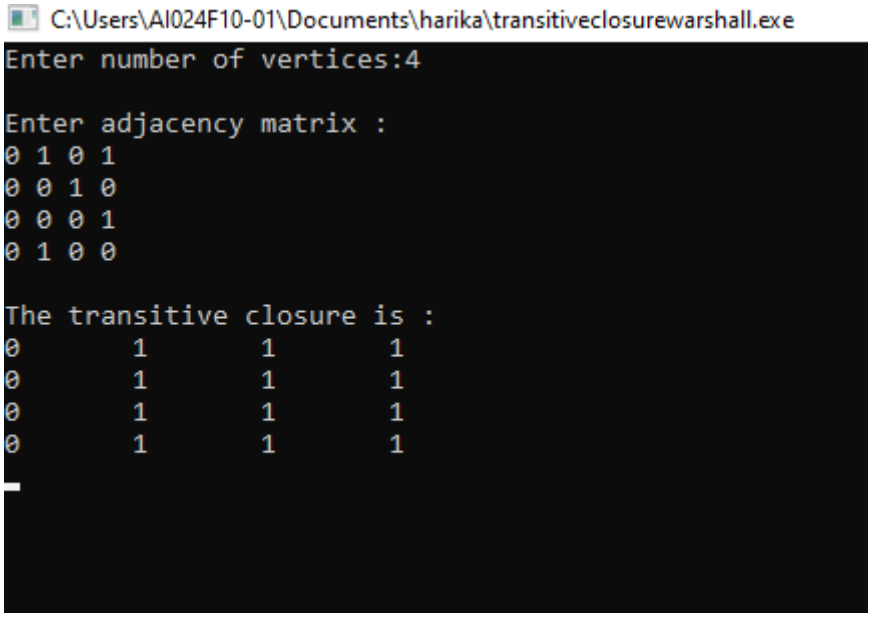
1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
   void warshall(int a[10][10], int n)
   {
   int i, j, k;
    for(k=0;k<n;k++)
    for(i=0;i<n;i++)
    if(a[i][k]==1)
    for(j=0;j<n;j++)
    a[i][j] = a[i][j] || a[k][j];
    }
    void main()
   {

    int n, i, j, a[10][10];
   clrscr();
    printf("Enter number of vertices:");
   scanf("%d",&n);
   printf("\nEnter adjacency matrix :\n");
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
   scanf("%d",&a[i][j]);
   warshall(a,n);
```

```
 printf("\nThe transitive closure is :\n");
for(i=0;i<n;i++)
 {
 for(j=0;j<n;j++)
printf("%d\t",a[i][j]);
printf("\n");
}
getch();
 }
```

OUTPUT:



PROGRAM NO -15

OBJECTIVE:

To 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.

PROCEDURE:

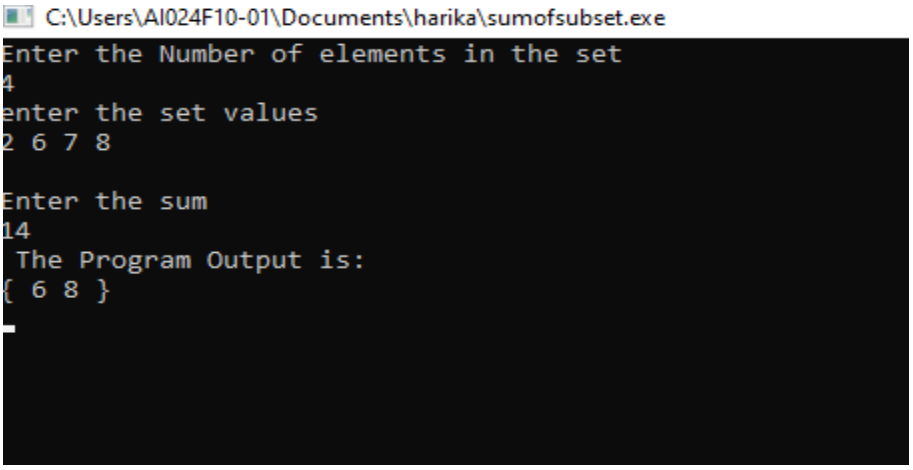
1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
int s[10], d,n,set[10],count=0;
void display(int);
int flag =0;
void main()
  {
      int subset(int,int);
      int i;
      printf("Enter the Number of elements in the set\n");
      scanf("%d",&n);
      printf("enter the set values\n");
      for(i=0;i<n;++i)
      scanf("%d",&s[i]);
      printf("\nEnter the sum\n");
      scanf("%d",&d);
      printf(" The Program Output is:\n");
      subset(0,0);
      if(flag == 0)
      printf(" There is no solution \n");
```

```
getch();
}
int subset(int sum,int i)
{
if(sum == d)
        {
                flag = 1;
                display(count);

                return;
        }
        if(sum>d || i>=n)
        return;else
        {
                set[count]=s[i];
                count++;
                subset(sum+s[i],i+1);
                count--;
                subset(sum,i+1);
        }
}
void display(int count)
{
int i;

clrscr();
printf("{ ");
for(i=0;i<count;i++)
printf("%d ",set[i]);
printf("}\n");
}
```

OUTPUT:


C:\Users\AI024F10-01\Documents\harika\sumofsubset.exe
Enter the Number of elements in the set
4
enter the set values
2 6 7 8
Enter the sum
14
The Program Output is:
{ 6 8 }