BANARSIDAS CHANDIWALA INSTITUTE OF INFORMATION TECHNOLOGY

Affiliated To Guru Gobind Singh Indraprastha University

SECTOR 16-C, DWARKA, NEW DELHI

PRACTICAL FILE

SUBJECT: DESIGN AND ANALYSIS OF ALGORITHMS

Submitted by:

Tanish Sharma
MCA Sem-III
70311104422

Submitted to:

Ms. Sonia Batra
Assistant Professor
BCIIT, GGSIPU

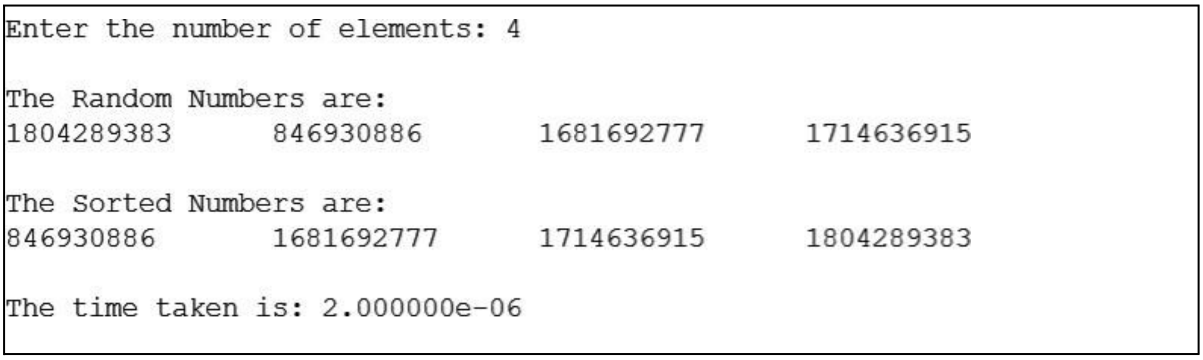
INDEX

Q1. Implement a quick sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, i.e., the number of elements in the list to be sorted. The elements can be generated using a random number generator.

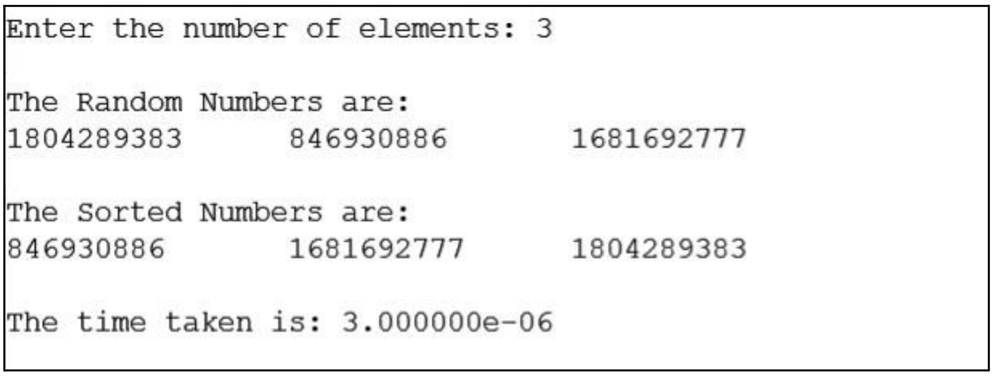
```
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include<stdlib.h>

void Exch(int *p, int *q){
   int temp = *p;
   *p = *q;
   *q = temp;
}

void Quicksort(int a[], int low, int high){
   int i, j, k, key;

   if(low>=high)
      return;

   key = low;
   i = low+1;
   j = high;


   while(i<=j){
      while(a[i]<= a[key])
      i=i+1;
      while(a[j]>a[key])
      j=j+1;
      if(i<j)
      Exch(&a[i], &a[j]);
   }

   Exch(&a[j], &a[key]);
   Quicksort(a, low, j-1);
   Quicksort(a, j+1, high);
}

void main(){
   int n, a[1000], k;
   clock_t st, et;
   double ts;

   printf("\nEnter the number of elements: ");
   scanf("%d", &n);
```

```
    printf("\nThe Random Numbers are: \n");
    for(k=1;k<=n;k++){
        a[k]=rand();
        printf("%d\t", a[k]);

}

    st = clock();
    Quicksort(a,1,n);
    et = clock();
    ts = (double) (et-st)/CLOCKS_PER_SEC;

    printf("\n\nThe Sorted Numbers are: \n");
        for(k=1;k<=n;k++){
            printf("%d\t", a[k]);
        }
    printf("\n\nThe time taken is: %e", ts);
    getch();

}
```

Output



```
Enter the number of elements: 4

The Random Numbers are:
1804289383      846930886       1681692777      1714636915

The Sorted Numbers are:
846930886       1681692777      1714636915      1804289383

The time taken is: 2.000000e-06
```

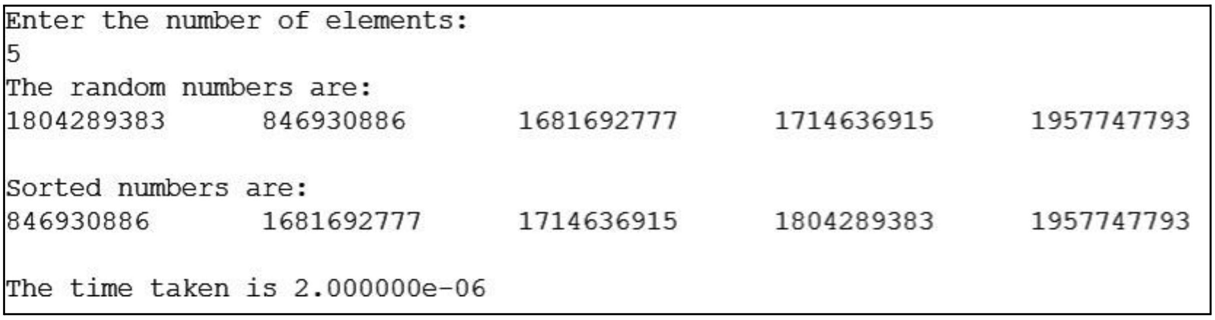


```
Enter the number of elements: 3

The Random Numbers are:
1804289383      846930886       1681692777

The Sorted Numbers are:
846930886       1681692777      1804289383

The time taken is: 3.000000e-06
```

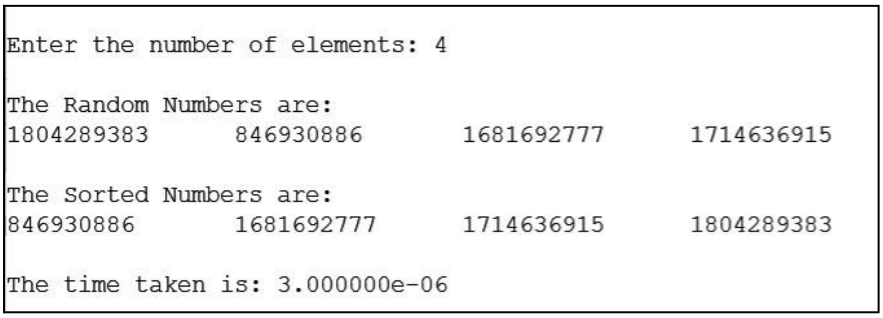
Q2. Implement a merge sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, i.e., the number of elements in the list to be sorted. The elements can be generated using a random number generator.

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

void Merge(int a[],int low, int mid, int high){
    int i,j,k,b[20];
    i=low, j=mid+1, k = low;

    while(i<=mid && j<=high){
        if(a[i]<=a[j]){
            b[k++] = a[i++];
        }
        else{
            b[k++]= a[j++];
        }
    }
        while(i<=mid){
            b[k++] = a[i++];
    }
        while(j<=high){
            b[k++]=a[j++];
    }
        for(k=low;k<=high;k++){
            a[k]=b[k];
        }
}

void MergeSort(int a[],int low, int high){
    int mid;
    if(low>=high){
        return;
    }

    mid=(low+high)/2;
    MergeSort(a, low, mid);
    MergeSort(a, mid+1, high);
    Merge(a, low, mid, high);
}

void main(){
    int n, a[2000], k;
    clock_t st, et;
    double ts;
    clrscr();

    printf("\nEnter the number of elements: ");
    scanf("%d", &n);
    printf("\nThe Random Numbers are: \n");
    for(k=1;k<=n;k++){
        a[k]=rand();
        printf("%d\t", a[k]);
    }

    st = clock();
    MergeSort(a, 1 , n);
    et = clock();
    ts= (double)(et-st)/CLOCKS_PER_SEC;

    printf("\n\nThe Sorted Numbers are: \n");
    for(k=1;k<=n;k++){
        printf("%d\t", a[k]);
    }
    printf("\n\nThe time taken is: %e", ts);
    getch();

}
```

Output



```
Enter the number of elements:
5
The random numbers are:
1804289383      846930886       1681692777      1714636915      1957747793

Sorted numbers are:
846930886       1681692777      1714636915      1804289383      1957747793

The time taken is 2.000000e-06
```

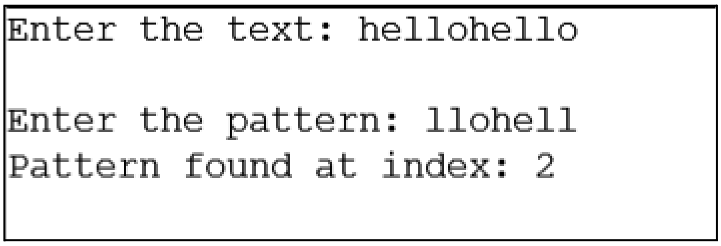


```
Enter the number of elements: 4

The Random Numbers are:
1804289383      846930886       1681692777      1714636915

The Sorted Numbers are:
846930886       1681692777      1714636915      1804289383

The time taken is: 3.000000e-06
```

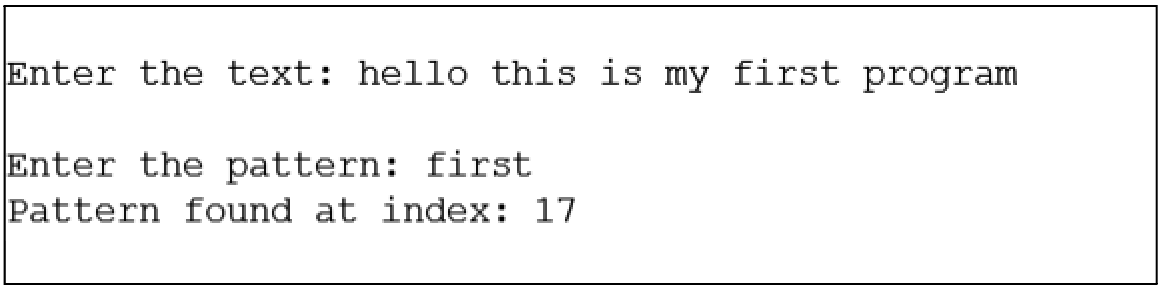
Q3. Write a program to find a substring in a string using Naive String Matching Algorithm.

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

int search(char *pat, char *text) {
    int M = strlen(pat);
    int N = strlen(text);
    int i, j;

    for (i = 0; i <= N - M; i++) {
        for (j = 0; j < M; j++) {
            if (text[i + j] != pat[j])
                break;
        }

        if (j == M)
            return i;
    }
    return -1;
}

int main() {
    char text[100], pat[100];
    int index;

    printf("\nEnter the text: ");
    fgets(text, 100, stdin);
    printf("\nEnter the pattern: ");
    fgets(pat, 100, stdin);

    // Remove newline characters from input
    text[strcspn(text, "\n")] = '\0';
    pat[strcspn(pat, "\n")] = '\0';

    index = search(pat, text);

    if (index >= 0) {
        printf("Pattern found at index: %d\n", index);
    } else {
        printf("Pattern not found\n");
    }

    return 0;
}
```

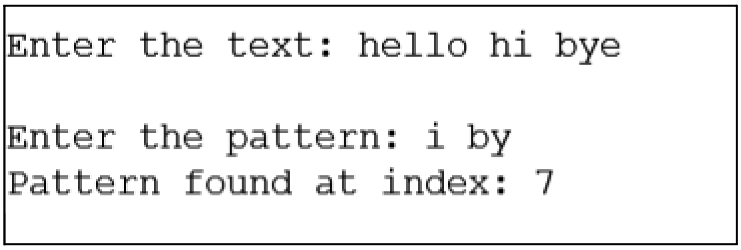
Output



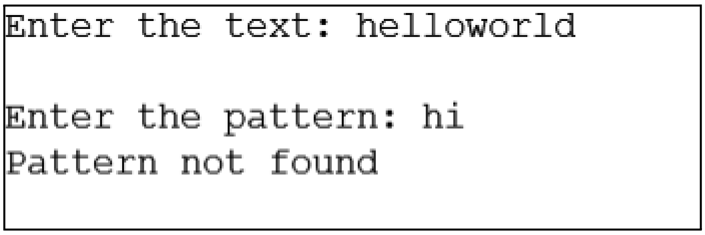
```
Enter the text: hellohello

Enter the pattern: llohell
Pattern found at index: 2
```



```

Enter the text: hello this is my first program

Enter the pattern: first
Pattern found at index: 17
```



```
Enter the text: hello hi bye

Enter the pattern: i by
Pattern found at index: 7
```



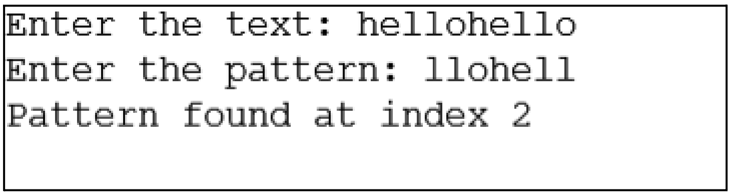
```
Enter the text: helloworld

Enter the pattern: hi
Pattern not found
```

Q4. Write a program to find a substring in a string using the Rabin Karp Algorithm.

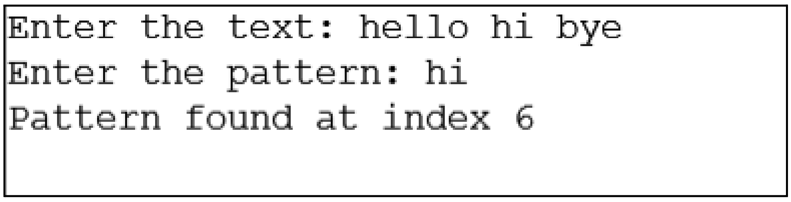
```
#include <stdio.h>
#include <string.h>
#define d 256

void search(char *pat, char *text, int q) {
    int M = strlen(pat);
    int N = strlen(text);
    int i, j;

    for (i = 0; i <= N - M; i++) {
        int p = 0;
        int t = 0;

        for (j = 0; j < M; j++) {
            p = (d * p + pat[j]) % q;
            t = (d * t + text[i + j]) % q;
        }

        if (p == t) {
            for (j = 0; j < M; j++) {
                if (text[i + j] != pat[j])
                    break;
            }

            if (j == M) {
                printf("Pattern found at index %d\n", i);
            }
            else{
                printf("Pattern not found!");
            }
        }
    }
}


int main() {
    char text[100], pat[100];
    int q = 101;

    printf("Enter the text: ");
    fgets(text, 100, stdin);
    printf("Enter the pattern: ");
    fgets(pat, 100, stdin);

    // Remove newline characters from input
    text[strcspn(text, "\n")] = '\0';
```

```
   pat[strcspn(pat, "\n")] = '\0';

   search(pat, text, q);

   return 0;
}
```

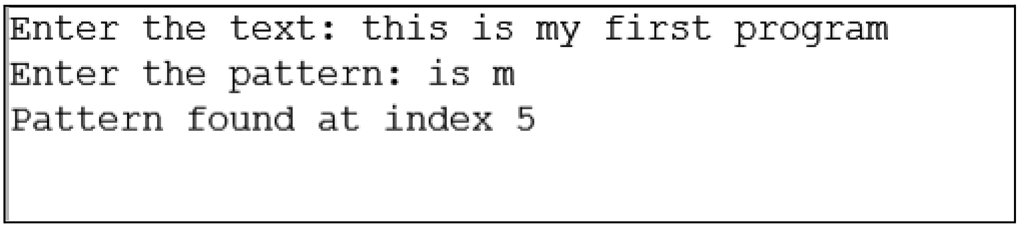
Output



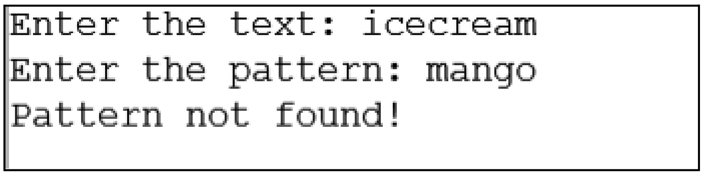
```
Enter the text: hellohello
Enter the pattern: llohell
Pattern found at index 2
```



```
Enter the text: hello hi bye
Enter the pattern: hi
Pattern found at index 6
```



```
Enter the text: this is my first program
Enter the pattern: is m
Pattern found at index 5
```



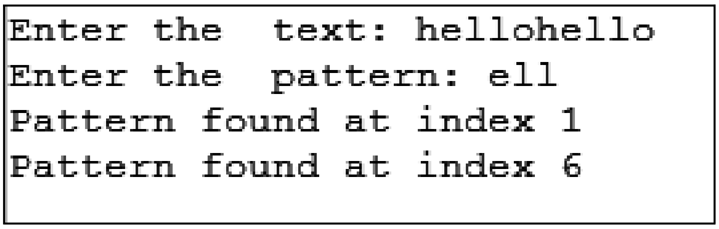
```
Enter the text: icecream
Enter the pattern: mango
Pattern not found!
```

Q5. Write a program to find a substring in a string using KMP Algorithm for String Matching.

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

void computeLPSArray(char *pat,int M,char *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 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,lps);
   int i=0;
   while(i<N)
   {
      if(pat[j]==txt[i])
```

```
        {
            j++;
            i++;
        }
        if(j==M)
        {
            printf("Pattern found at index %d \n",i-j);
            j=lps[j-1];
        }
        else if(i<N && pat[j]!=txt[i])
        {
            if(j!=0)
            {
                j=lps[j-1];
            }
            else
            {
                i=i+1;
            }
        }

    }
     free(lps);

}

int main()
{
    char txt[20];
    char pat[10];
    printf("Enter the  text: ");
    scanf("%s",&txt);

    printf("Enter the  pattern: ");
    scanf("%s",&pat);

    KMPSearch(pat,txt);
    return 0;
}
```

Output



```
Enter the  text: hellohello
Enter the  pattern: ell
Pattern found at index 1
Pattern found at index 6
```

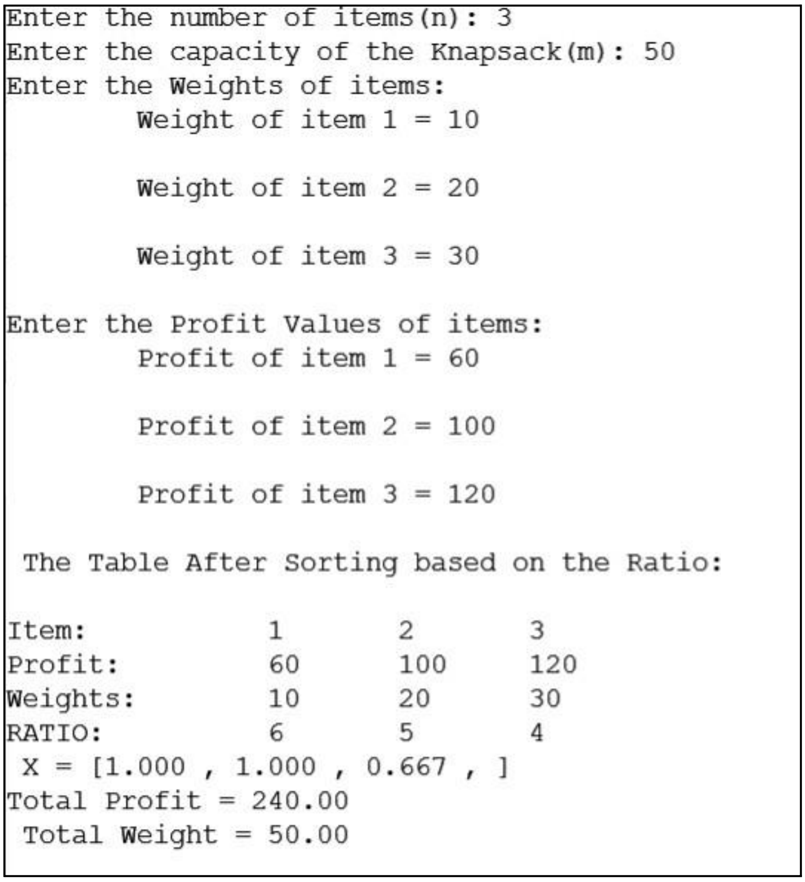
Q6. Write a program for the Fractional Knapsack problem.

```
#include<stdio.h>

void main (){
  int n, m, w[100], p[100], ratio[100] , i, j, u, temp;
  float xr, x[100], total_profit=0, total_weight=0;

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

  printf ("Enter the capacity of the Knapsack(m): ");
  scanf ("%d", &m);

  //Initializing remaining capacity of Knapsack (u)
  u = m;

  //Initializing Solution Array x[]
  for(i=0;i<n;i++){
      x[i]=0;
  }

  //Reading the Weights
  printf ("Enter the Weights of items: ");
  for (i = 0; i < n; i++){
      printf ("\n\tWeight of item %d = ", i + 1);
      scanf ("%d", &w[i]);
    }

  //Reading the Profit values
  printf ("\nEnter the Profit Values of items: ");
  for (i = 0; i < n; i++){
      printf ("\n\tProfit of item %d = ", i + 1);
      scanf ("%d", &p[i]);
    }

  for (i = 0; i < n; i++){
      ratio[i] = p[i] / w[i];
    }

  for (i = 0; i < n; i++){
      for (j = 0; j < n - 1; j++){
            if (ratio[j] < ratio[i]){
                temp = ratio[i];
                ratio[i] = ratio[j];
                ratio[j] = temp;

                temp = w[i];
```

```
              w[i] = w[j];
              w[j] = temp;

              temp = p[i];
              p[i] = p[j];
              p[j] = temp;
            }
        }
}

printf("\n The Table After Sorting based on the Ratio: \n");

//Printing Item numbers
printf("\nItem:\t\t");
for(i=0;i<n;i++){
        printf("%d\t",i+1);
}

printf("\nProfit:\t\t");
for(i=0;i<n;i++){
        printf("%d\t",p[i]);
}

printf("\nWeights:\t");
for(i=0;i<n;i++){
        printf("%d\t",w[i]);
}

printf ("\nRATIO:\t\t");
for (i = 0; i < n; i++){
printf ("%d\t", ratio[i]);
}

//Calculating Solution Array x
for(i=0;i<n;i++){
        if(w[i]<=u){
            x[i]=1;
            u=u-w[i];
        }
        else if(w[i]>u){
            break;
        }
}

if(i<=n){
        xr = (float)u/w[i];
        x[i] = xr;
}
```

```
    //Printing Solution Array x
    printf("\n X = [");
    for(i=0;i<n;i++){
            printf("%.3f , ",x[i]);
    }
    printf("]");

    for(i=0;i<n;i++){
        total_profit += x[i]*p[i];
        total_weight += x[i]*w[i];
    }
            printf("\nTotal Profit = %.2f \n Total Weight = %.2f ",total_profit,total_weight);

}
```

Output

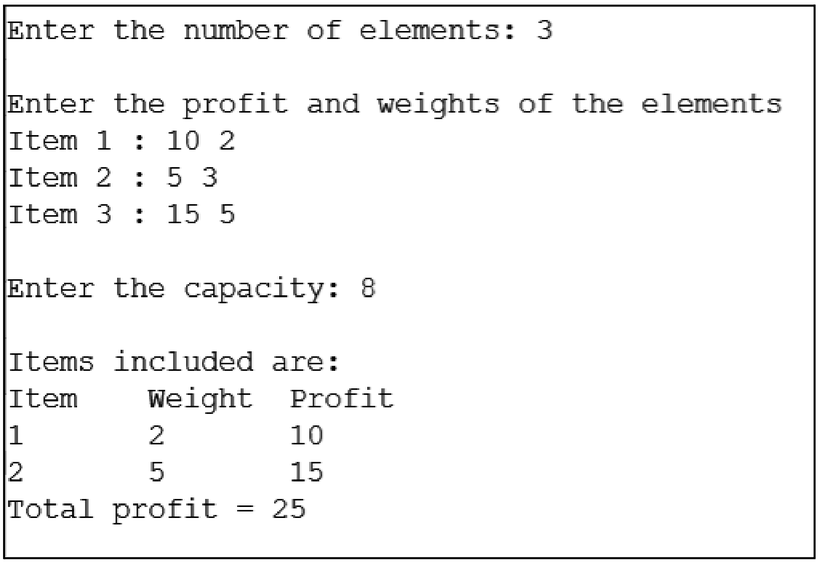


```
Enter the number of items(n): 3
Enter the capacity of the Knapsack(m): 50
Enter the Weights of items:
        Weight of item 1 = 10

        Weight of item 2 = 20

        Weight of item 3 = 30

Enter the Profit Values of items:
        Profit of item 1 = 60

        Profit of item 2 = 100

        Profit of item 3 = 120

 The Table After Sorting based on the Ratio:

Item:           1       2       3
Profit:         60      100     120
Weights:        10      20      30
RATIO:          6       5       4
 X = [1.000 , 1.000 , 0.667 , ]
Total Profit = 240.00
 Total Weight = 50.00
```

Q7. Write a program for the 0/1 Knapsack Problem.

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

int w[10], p[10], v[10][10], n, i, j, cap, x[10] = {0};

int max(int i, int j) {
   return ((i > j) ? i : j);
}

int knap(int i, int j) {
   int value;
   if (v[i][j] < 0) {
      if (j < w[i])
         value = knap(i - 1, j);
      else
         value = max(knap(i - 1, j), p[i] + knap(i - 1, j - w[i]));
          v[i][j] = value;
   }
   return (v[i][j]);
}

void main() {
   int profit, count = 0;

   printf("\nEnter the number of elements: ");
   scanf("%d", &n);
   printf("\nEnter the profit and weights of the elements\n");
   for (i = 1; i <= n; i++) {
      printf("Item %d : ", i);
      scanf("%d%d", &p[i], &w[i]);
   }
   printf("\nEnter the capacity: ");
   scanf("%d", &cap);

   for (i = 0; i <= n; i++)
      for (j = 0; j <= cap; j++)
         if ((i == 0) || (j == 0))
            v[i][j] = 0;
         else
            v[i][j] = -1;

   profit = knap(n, cap);
   i = n;
   j = cap;
   while (j != 0 && i != 0) {
      if (v[i][j] != v[i - 1][j]) {
            x[i] = 1;
            j = j - w[i];
            i--;
        } else
            i--;
    }
    printf("\nItems included are: \n");
    printf("Item\tWeight\tProfit\n");
    for (i = 1; i <= n; i++)
        if (x[i])
            printf("%d\t%d\t%d\n", ++count, w[i], p[i]);

    printf("Total profit = %d\n", profit);
    getch();
}
```

Output

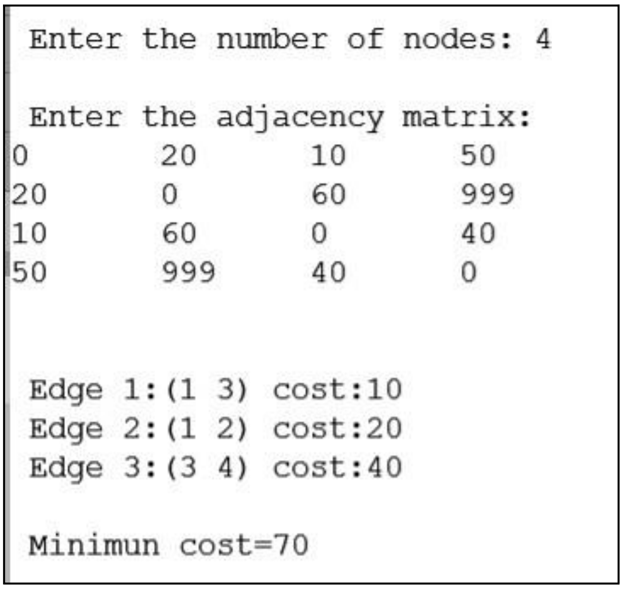


```
Enter the number of elements: 3

Enter the profit and weights of the elements
Item 1 : 10 2
Item 2 : 5 3
Item 3 : 15 5

Enter the capacity: 8

Items included are:
Item    Weight  Profit
1       2       10
2       5       15
Total profit = 25
```

Q8. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

```
#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() {
    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\n Minimum cost=%d", mincost);
    getch();
}
```

Output

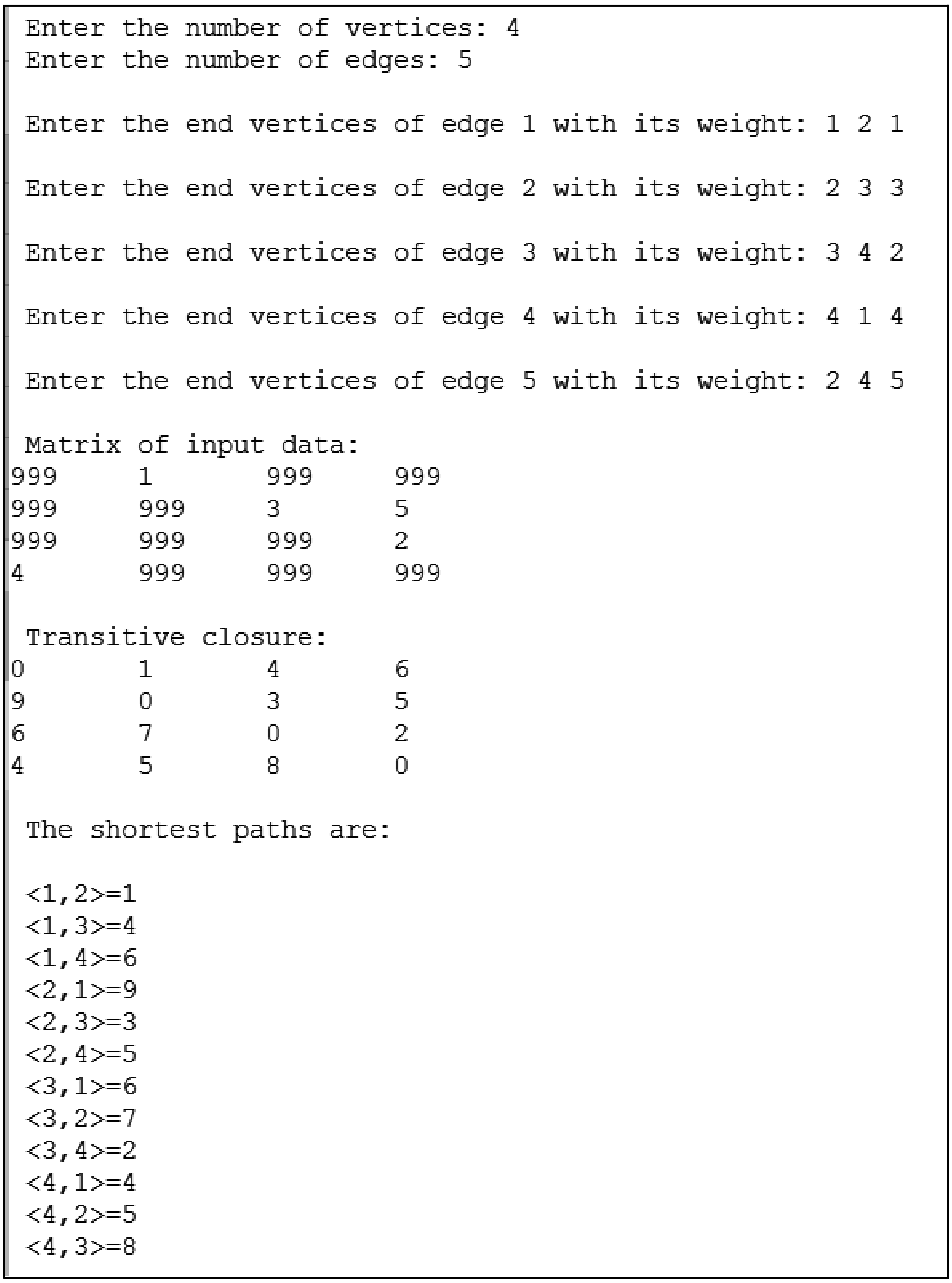


```
 Enter the number of nodes: 4

 Enter the adjacency matrix:
0       20      10      50
20      0       60      999
10      60      0       40
50      999     40      0


 Edge 1:(1 3) cost:10
 Edge 2:(1 2) cost:20
 Edge 3:(3 4) cost:40

 Minimun cost=70
```

Q9. Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.

```
#include <stdio.h>

int min(int, int);
void floyds(int p[10][10], int n);

int min(int a, int b)
{
   if (a < b)
      return (a);
   else
      return (b);
}

void floyds(int p[10][10], int n)
{
   int i, j, k;
   for (k = 1; k <= n; k++)
      for (i = 1; i <= n; i++)
         for (j = 1; j <= n; j++)
            if (i == j)
               p[i][j] = 0;
            else
               p[i][j] = min(p[i][j], p[i][k] + p[k][j]);
}

void main()
{
   int p[10][10], w, n, e, u, v, i, j;
   printf("\n Enter the number of vertices: ");
   scanf("%d", &n);
   printf(" Enter the number of edges: ");
   scanf("%d", &e);
   for (i = 1; i <= n; i++)
   {
      for (j = 1; j <= n; j++)
         p[i][j] = 999;
   }
   for (i = 1; i <= e; i++)
   {
      printf("\n Enter the end vertices of edge %d with its weight: ", i);
      scanf("%d%d%d", &u, &v, &w);
      p[u][v] = w;
   }
   printf("\n Matrix of input data:\n");
   for (i = 1; i <= n; i++)
   {
        for (j = 1; j <= n; j++)
            printf("%d \t", p[i][j]);
        printf("\n");
    }
    floyds(p, n);
    printf("\n Transitive closure:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
            printf("%d \t", p[i][j]);
        printf("\n");
    }
    printf("\n The shortest paths are:\n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
        {
            if (i != j)
                printf("\n <%d,%d>=%d", i, j, p[i][j]);
        }
}
```

Output

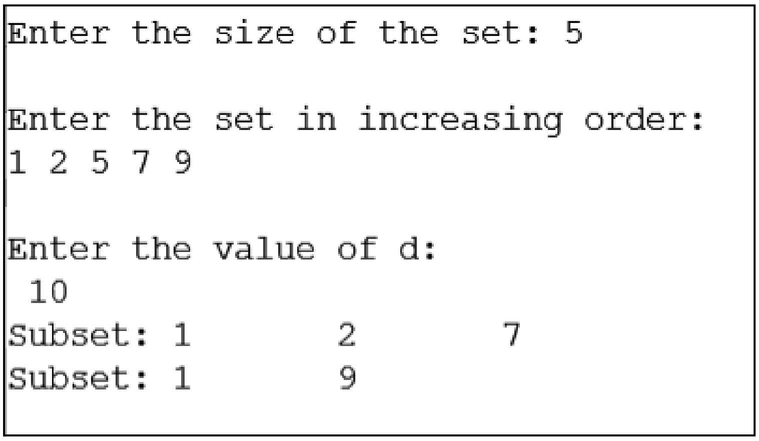


```
 Enter the number of vertices: 4
 Enter the number of edges: 5

 Enter the end vertices of edge 1 with its weight: 1 2 1

 Enter the end vertices of edge 2 with its weight: 2 3 3

 Enter the end vertices of edge 3 with its weight: 3 4 2

 Enter the end vertices of edge 4 with its weight: 4 1 4

 Enter the end vertices of edge 5 with its weight: 2 4 5

 Matrix of input data:
999     1       999     999
999     999     3       5
999     999     999     2
4       999     999     999

 Transitive closure:
0       1       4       6
9       0       3       5
6       7       0       2
4       5       8       0

 The shortest paths are:

 <1,2>=1
 <1,3>=4
 <1,4>=6
 <2,1>=9
 <2,3>=3
 <2,4>=5
 <3,1>=6
 <3,2>=7
 <3,4>=2
 <4,1>=4
 <4,2>=5
 <4,3>=8
```

Q10. 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.

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

int s[10], x[10], d;

void sumofsub(int, int, int);

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

    printf("\nEnter the size of the set: ");
    scanf("%d", &n);

    printf("\nEnter the set in increasing order:\n");
    for (i = 1; i <= n; i++)
        scanf("%d", &s[i]);

    printf("\nEnter the value of d: \n ");
    scanf("%d", &d);

    for (i = 1; i <= n; i++)
        sum = sum + s[i];

    if (sum < d || s[1] > d)
        printf("\nNo subset possible: ");
    else
        sumofsub(0, 1, sum);

    getch();
}

void sumofsub(int m, int k, int r) {
    int i = 1;
    x[k] = 1;

    if ((m + s[k]) == d) {
        printf("Subset:");
        for (i = 1; i <= k; i++)
            if (x[i] == 1)
                printf("\t%d", s[i]);

        printf("\n");
    } else if (m + s[k] + s[k + 1] <= d)
        sumofsub(m + s[k], k + 1, r - s[k]);

    if ((m + r - s[k] >= d) && (m + s[k + 1] <= d)) {
        x[k] = 0;
        sumofsub(m, k + 1, r - s[k]);
    }
}
```

<u>Output</u>



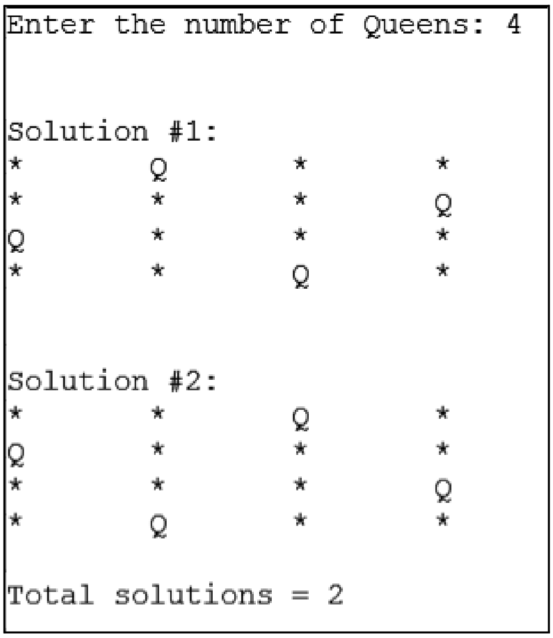
```
Enter the size of the set: 5

Enter the set in increasing order:
1 2 5 7 9

Enter the value of d:
 10
Subset: 1       2       7
Subset: 1       9
```

Q11. Implement N Queen's problem using Back Tracking.

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

int a[30], count = 0;

int place(int pos) {
    int i;
    for (i = 1; i < pos; i++) {
        if ((a[i] == a[pos]) || ((abs(a[i] - a[pos]) == abs(i - pos))))
            return 0;
    }
    return 1;
}

void print_sol(int n) {
    int i, j;
    count++;
    printf("\n\nSolution #%d:\n", count);
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            if (a[i] == j)
                printf("Q\t");
            else
                printf("*\t");
        }
        printf("\n");
    }
}

void queen(int n) {
    int k = 1;
    a[k] = 0;
    while (k != 0) {
        a[k] = a[k] + 1;
        while ((a[k] <= n) && !place(k))
            a[k]++;
        if (a[k] <= n) {
            if (k == n)
                print_sol(n);
            else {
                k++;
                a[k] = 0;
            }
        } else
```

```
            k--;
    }
}

void main() {
    int i, n;
    printf("Enter the number of Queens: ");
    scanf("%d", &n);
    queen(n);
    printf("\nTotal solutions = %d", count);
    getch();
}
```

Output



```
Enter the number of Queens: 4


Solution #1:
*       Q       *       *
*       *       *       Q
Q       *       *       *
*       *       Q       *


Solution #2:
*       *       Q       *
Q       *       *       *
*       *       *       Q
*       Q       *       *

Total solutions = 2
```