

## Experiment 6A

### Finding Max and Min in Array

#### Aim-

To write a C program to find the maximum and minimum elements of a given array.

#### Algorithm-

1. Let **maxE** and **minE** be the variable to store the minimum and maximum element of the array.
2. Initialise **minE** as INT\_MAX and **maxE** as INT\_MIN.
3. Traverse the given array arr[].
4. If the current element is smaller than **minE**, then update the **minE** as current element.
5. If the current element is greater than **maxE**, then update the **maxE** as current element.
6. Repeat the above two steps for the element in the array.

#### Code-

```
#include <limits.h>

#include <stdio.h>

void recursiveMinMax(int arr[], int N,
int* minE, int* maxE)
{
    if (N < 0) {
        return;
    }

    if (arr[N] < *minE) {
        *minE = arr[N];
    }

    if (arr[N] > *maxE) {
        *maxE = arr[N];
    }

    recursiveMinMax(arr, N - 1, minE, maxE);
}
```

```

}

void findMinimumMaximum(int arr[], int N)
{
    int i;

    int minE = INT_MAX, maxE = INT_MIN;
    recursiveMinMax(arr, N - 1, &minE, &maxE);
    printf("The minimum element is %d", minE);
    printf("\n");
    printf("The maximum element is %d", maxE);
    return;
}

int main()
{
    int arr[] = { 1, 2, 4, -1 };
    int N = sizeof(arr) / sizeof(arr[0]);
    findMinimumMaximum(arr, N);
    return 0;
}

```

**Output-**

```
1  #include <limits.h>
2  #include <stdio.h>
3  void recursiveMinMax(int arr[], int N,
4  int* minE, int* maxE)
5  {
6      if (N < 0) {
7          return;
8      }
9      if (arr[N] < *minE) {
10         *minE = arr[N];
11     }
12     if (arr[N] > *maxE) {
13         *maxE = arr[N];
14     }
15     recursiveMinMax(arr, N - 1, minE, maxE);
16 }
17 void findMinimumMaximum(int arr[], int N)
18 {
19     int i;
20
21     int minE = INT_MAX, maxE = INT_MIN;
22     recursiveMinMax(arr, N - 1, &minE, &maxE);
23     printf("The minimum element is %d", minE);
24     printf("\n");
25     printf("The maximum element is %d", maxE);

```

The minimum element is -1  
The maximum element is 4

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

**Result-** A code has been written to find the maximum and minimum elements, and the output has been verified.

## Experiment 6B

### Convex Hull Problem

#### Aim-

To write a C program to use convex hull algorithm to find the convex hull length of a set of 2D points.

#### Algorithm-

1. Choose a point roughly in the centre of your point cloud.
2. Then sort the points radially, by angle from the centre. The topmost point must be in the convex hull, so define it as having an angle of 0.0 and being first in the list.
3. Put point 2 in the "tentative" hull list.
4. Then check point 3. If the angle P1-P2-P3 is concave (relative to the centre point), remove P2 from the list, if it is convex, keep it.
5. Continue steps 3-6, backtracking and removing points if they go concave.
6. You only need two points in your "tentative" list, once you have three, they become definite.
7. You stop when you go full circle and get back to P1

#### Code-

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct point
{
    double x;
    double y;
}POINT,VECTOR;

POINT b[1000];
VECTOR normal;
int n;

int upper_lower(int i, VECTOR ab, double c) {
```

```

    double x, y,result;
    y = b[i].y;
    x = normal.x*b[i].x;
    result = -(x + c) / normal.y;
    if (y>result) return 1;
    if (y == result) return 0;
    else
        return -1;
}

int ccw(VECTOR v,VECTOR v2)
{
    double cp;

    cp = v2.x*v.y - v2.y*v.x;

    if (cp == abs(cp)) return 1;
    else
        return -1;
}

double vector_length(VECTOR v)
{
    return sqrt(pow(v.x, 2) + pow(v.y, 2));
}

int cmp_points(const void *p1, const void *p2)
{
    const POINT *pt1 = p1;
    const POINT *pt2 = p2;
    if (pt1->x > pt2->x)
        return 1;
    if (pt1->x < pt2->x)
        return -1;
    if (pt1->y > pt2->y) return 1;
    if (pt1->y < pt2->y) return -1;
    return 0;
}

int main()
{

```

```

int i, poloha, upper[1000], lower[1000], h=0, d=0;
scanf("%d", &n);
if (n <= 0 && n > 1000) return 0;
for (i = 0; i < n; i++)
{
    scanf("%lf %lf", &b[i].x, &b[i].y);
}
qsort(b, n, sizeof(POINT), cmp_points);

VECTOR ab;
double c;
ab.x = b[n - 1].x - b[0].x;
ab.y = b[n - 1].y - b[0].y;
normal.x = -ab.y;
normal.y = ab.x;
c = -normal.x*b[0].x - (normal.y*b[0].y);
for (i = 0; i < n; i++)
{
    poloha = upper_lower(i, ab, c);
    if (poloha == 1) upper[h++] = i;
    if (poloha == -1) lower[d++] = i;
    if (poloha == 0)
    {
        upper[h++] = i;
        lower[d++] = i;
    }
}

int j = 0;
double v, length = 0;
VECTOR v1, v2, v3, v4;
v3.x = 0; v3.y = 0;
for (i = 0; ; i++)
{
    int in = 0;
    if (lower[i + 2] < 0)
    {
        v1.x = b[lower[i + 1]].x - b[lower[0]].x;
        v1.y = b[lower[i + 1]].y - b[lower[0]].y;

        v2.x = b[lower[i]].x - b[lower[i + 1]].x;
        v2.y = b[lower[i]].y - b[lower[i + 1]].y;

        length += vector_length(v1);
        length += vector_length(v2);
        break;
    }
}

```

```

    }
    v1.x = b[lower[i + 1]].x - b[lower[i]].x;
    v1.y = b[lower[i + 1]].y - b[lower[i]].y;

    v2.x = b[lower[i + 2]].x - b[lower[i]].x;
    v2.y = b[lower[i + 2]].y - b[lower[i]].y;
    in = ccw(v1, v2);
    if (in == 1)
    {
        length +=
        vector_length(v1); v3 = v2;
        v4 = v1;
    }
    if (in == -1)
    {
        length -=
        vector_length(v4); if (v3.x !
        = 0 && v3.y != 0)
        {
            length += vector_length(v3);
            v3.x = 0; v3.y = 0;
        }
        else
        {
            length += vector_length(v2);

        }

    }

}
}

printf("%.3lf", length);

return 0;
}

```

**Output-**

```
1
2  #include <iostream>
3  #include <stack>
4  #include <stdlib.h>
5  using namespace std;
6
7  struct Point
8  {
9      int x, y;
10 };
11
12 Point p0;
13
14 Point nextToTop(stack<Point> &S)
15 {
16     Point p = S.top();
17     S.pop();
18     Point res = S.top();
19     S.push(p);
20     return res;
21 }
22
23 void swap(Point &p1, Point &p2)
24 {
25     Point temp = p1;
26     p1 = p2;
27     p2 = temp;
28 }
```

(0, 3)  
(4, 4)  
(3, 1)  
(0, 0)

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

### Result-

A code has been written and the output has been verified.



## Experiment 7A

### Huffman Coding using Greedy

#### Aim-

To write a C program to implement Huffman coding using Greedy algorithm.

#### Algorithm-

1. Build a min heap that contains 6 nodes where each node represents root of a tree with single node.
2. Extract two minimum frequency nodes from min heap. Add a new internal node with frequency  $5 + 9 = 14$ .
3. Extract two minimum frequency nodes from heap. Add a new internal node with frequency  $12 + 13 = 25$ .
4. Extract two minimum frequency nodes. Add a new internal node with frequency  $14 + 16 = 30$ .
5. Extract two minimum frequency nodes. Add a new internal node with frequency  $25 + 30 = 55$ .
6. Extract two minimum frequency nodes. Add a new internal node with frequency  $45 + 55 = 100$ .
7. Print the resultant output.

#### Code-

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_TREE_HT

100 struct MinHeapNode {
    char data;
    unsigned freq;
    struct MinHeapNode *left, *right;
```

```

};

struct MinHeap {
    unsigned size;
    unsigned capacity;
    struct MinHeapNode** array;
};

struct MinHeapNode* newNode(char data, unsigned freq)
{
    struct MinHeapNode* temp = (struct MinHeapNode*)malloc(
        sizeof(struct MinHeapNode));
    temp->left = temp->right =
    NULL; temp->data = data;
    temp->freq =
    freq; return temp;
}

struct MinHeap* createMinHeap(unsigned capacity)
{
    struct MinHeap* minHeap
        = (struct MinHeap*)malloc(sizeof(struct
    MinHeap)); minHeap->size = 0;
    minHeap->capacity = capacity;
    minHeap->array = (struct MinHeapNode**)malloc(
        minHeap->capacity * sizeof(struct
        MinHeapNode*));
    return minHeap;
}

```

```

void swapMinHeapNode(struct MinHeapNode** a,
                    struct MinHeapNode** b)
{
    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

void minHeapify(struct MinHeap* minHeap, int idx)
{
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;
    if (left < minHeap->size
        && minHeap->array[left]->freq
            < minHeap->array[smallest]->freq)
        smallest = left;
    if (right < minHeap->size
        && minHeap->array[right]->freq
            < minHeap->array[smallest]->freq)
        smallest = right;
    if (smallest != idx) {
        swapMinHeapNode(&minHeap->array[smallest],
                        &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}

```

```

}

int isSizeOne(struct MinHeap* minHeap)
{
    return (minHeap->size == 1);
}

struct MinHeapNode* extractMin(struct MinHeap* minHeap)
{
    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size -
1];
    --minHeap->size;
    minHeapify(minHeap, 0);
    return temp;
}

void insertMinHeap(struct MinHeap* minHeap,
    struct MinHeapNode* minHeapNode)
{
    ++minHeap->size;
    int i = minHeap->size - 1;
    while (i
        && minHeapNode->freq
            < minHeap->array[(i - 1) / 2]->freq) {
        minHeap->array[i] = minHeap->array[(i - 1) / 2];
        i = (i - 1) / 2;
    }
}

```

```

    }

    minHeap->array[i] = minHeapNode;
}

void buildMinHeap(struct MinHeap* minHeap)
{
    int n = minHeap->size - 1;

    int i;

    for (i = (n - 1) / 2; i >= 0; --i)
        minHeapify(minHeap, i);
}

void printArr(int arr[], int n)
{
    int i;

    for (i = 0; i < n; ++i)
        printf("%d", arr[i]);

    printf("\n");
}

int isLeaf(struct MinHeapNode* root)
{
    return !(root->left) && !(root->right);
}

struct MinHeap* createAndBuildMinHeap(char data[],
                                       int freq[], int size)
{
    struct MinHeap* minHeap = createMinHeap(size);

```

```

    for (int i = 0; i < size; ++i)
        minHeap->array[i] = newNode(data[i],
        freq[i]); minHeap->size = size;
    buildMinHeap(minHeap);
    return minHeap;
}

struct MinHeapNode* buildHuffmanTree(char data[],
        int freq[], int size)
{
    struct MinHeapNode *left, *right, *top;
    struct MinHeap* minHeap
        = createAndBuildMinHeap(data, freq, size);
    while (!isSizeOne(minHeap)) {
        left = extractMin(minHeap);
        right =
            extractMin(minHeap);
        top = newNode('$', left->freq + right-
            >freq); top->left = left;
        top->right = right;
        insertMinHeap(minHeap, top);
    }
    return extractMin(minHeap);
}

void printCodes(struct MinHeapNode* 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)) {
        printf("%c: ", root->data);
        printArr(arr, top);
    }
}

void HuffmanCodes(char data[], int freq[], int size)
{
    struct MinHeapNode* 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', 'f' };
    int freq[] = { 5, 9, 12, 13, 16, 45 };
    int size = sizeof(arr) / sizeof(arr[0]);

```

```

    HuffmanCodes(arr, freq, size);

    return 0;
}

```

### Output-

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX_TREE_HT 100
4  struct MinHeapNode {
5      char data;
6      unsigned freq;
7      struct MinHeapNode *left, *right;
8  };
9  struct MinHeap {
10     unsigned size;
11     unsigned capacity;
12     struct MinHeapNode** array;
13 };
14 struct MinHeapNode* newNode(char data, unsigned freq)
15 {
16     struct MinHeapNode* temp = (struct MinHeapNode*)malloc(
17         sizeof(struct MinHeapNode));
18     temp->left = temp->right = NULL;
19     temp->data = data;
20     temp->freq = freq;
21     return temp;
22 }
23 struct MinHeap* createMinHeap(unsigned capacity)
24 {
25     struct MinHeap* minHeap

```

f: 0  
c: 100  
d: 101  
a: 1100  
b: 1101  
e: 111

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



**Result-**

The code has been written, and the output has been verified.

## Experiment 7B

## Knapsack using Greedy

### Aim-

To solve knapsack problems using greedy algorithm.

### Code-

```
#include<stdio.h>

int main()
{
    float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
    int n,I,j;
    printf("Enter the number of items :");
    scanf("%d",&n);
    for (I = 0; I < n; i++)
    {
        printf("Enter Weight and Profit for item[%d] :\n",i); scanf("%f %f", &weight[i], &profit[i]);
    }
    printf("Enter the capacity of knapsack :\n"); scanf("%f",&capacity);

    for(i=0;i<n;i++)
        ratio[i]=profit[i]/weight[i];

    for (I = 0; I < n; i++)
        for (j = I + 1; j < n; j++)
```

```
if (ratio[i] < ratio[j])
```

```
{
```

```
    temp = ratio[j];
```

```
    ratio[j] = ratio[i];
```

```
    ratio[i] = temp;
```

```
    temp = weight[j];
```

```
    weight[j] = weight[i];
```

```
    weight[i] = temp;
```

```
    temp = profit[j];
```

```
    profit[j] = profit[i];
```

```
    profit[i] = temp;
```

```
}
```

```
printf("Knapsack problems using Greedy Algorithm:\n");
```

```
for (I = 0; I < n; i++)
```

```
{
```

```
    if (weight[i] > capacity)
```

```
        break;
```

```
    else
```

```
    {
```

```
        Totalvalue = Totalvalue + profit[i];
```

```
        capacity = capacity - weight[i];
```

```
    }
```

```
}  
  
    if (I < n)  
  
        Totalvalue = Totalvalue + (ratio[i]*capacity);  
  
    printf("\nThe maximum value is :%f\  
n",Totalvalue); return 0;  
  
}
```

## Output-

```
26
27     temp = weight[j];
28     weight[j] = weight[i];
29     weight[i] = temp;
30
31     temp = profit[j];
32     profit[j] = profit[i];
33     profit[i] = temp;
34 }
35
36 printf("Knapsack problems using Greedy Algorithm:\n");
37 for (i = 0; i < n; i++)
38 {
39     if (weight[i] > capacity)
40         break;
41     else
42     {
43         Totalvalue = Totalvalue + profit[i];
44         capacity = capacity - weight[i];
45     }
46 }
47 if (i < n)
48     Totalvalue = Totalvalue + (ratio[i]*capacity);
49 printf("\nThe maximum value is :%f\n",Totalvalue);
50 return 0;
51 }
52
```

Enter the number of items :1 2  
Enter Weight and Profit for item[0] :  
3 4  
Enter the capacity of knapsack :  
Knapsack problems using Greedy Algorithm:  
  
The maximum value is :-1502137548800.000000  
  
...Program finished with exit code 0  
Press ENTER to exit console.

## Result-

The code has been written, and the output has been verified.

## Experiment 8A

## Tree Traversal

**Aim-** To write a code to demonstrate tree traversal.

### Algorithm-

Inorder traversal

- 1.First, visit all the nodes in the left subtree
- 2.Then the root node
- 3.Visit all the nodes in the right subtree

Preorder traversal

- 1.Visit root node
- 2.Visit all the nodes in the left subtree
- 3.Visit all the nodes in the right

subtree Postorder traversal

- 1.Visit all the nodes in the left subtree
- 2.Visit all the nodes in the right subtree
- 3..Visit the root node

### Code-

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

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

```
struct node {
```

```
    int data;
```

```
    struct node* left;
```

```
    struct    node*
```

```
    right;
```

```
};
```

```

struct node* newNode(int data)
{
    struct node* node
        = (struct node*)malloc(sizeof(struct
node)); node->data = data;
    node->left = NULL;
    node->right =
    NULL; return (node);
}

void printPostorder(struct node* node)
{
    if (node == NULL)
        return;
    printPostorder(node->left);
    printPostorder(node->right);
    printf("%d ", node->data);
}

void printInorder(struct node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    printf("%d ", node-
>data); printInorder(node-
>right);
}

```

```

void printPreorder(struct node* node)
{
    if (node == NULL)
        return;

    printf("%d ", node->data);
    printPreorder(node->left);
    printPreorder(node->right);
}

int main()
{
    struct node* root =
    newNode(1); root->left =
    newNode(2);
    root->right = newNode(3);
    root->left->left =
    newNode(4);
    root->left->right = newNode(5);

    printf("\nPreorder traversal of binary tree is \
n"); printPreorder(root);
    printf("\nInorder traversal of binary tree is \
n"); printInorder(root);
    printf("\nPostorder traversal of binary tree is \
n"); printPostorder(root);
    getchar();
    return 0;
}

```



## Output-

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 struct node {
4     int data;
5     struct node* left;
6     struct node* right;
7 };
8 struct node* newNode(int data)
9 {
10     struct node* node
11     = (struct node*)malloc(sizeof(struct node));
12     node->data = data;
13     node->left = NULL;
14     node->right = NULL;
15     return (node);
16 }
17 void printPostorder(struct node* node)
18 {
19     if (node == NULL)
20         return;
21     printPostorder(node->left);
22     printPostorder(node->right);
23     printf("%d ", node->data);
24 }
25 void printInorder(struct node* node)
26 {
27     if (node == NULL)
28         return;
29     printInorder(node->left);
30     printf("%d ", node->data);
31     printInorder(node->right);
32 }
33 void printPreorder(struct node* node)
34 {
35     if (node == NULL)
36         return;
37     printf("%d ", node->data);
38     printPreorder(node->left);
39     printPreorder(node->right);
40 }
```

Preorder traversal of binary tree is  
1 2 4 5 3  
Inorder traversal of binary tree is  
4 2 5 1 3  
Postorder traversal of binary tree is  
4 5 2 3 1

## Result-

The code has been written and the output has been verified.

## Experiment 8B

### Krushkal's Minimum Spanning Tree

**Aim-** To write a code to demonstrate Krushkal's MST algorithm.

#### **Algorithm-**

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are  $(V-1)$  edges in the spanning tree.

#### **Code-**

```
#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

void main()

{

    printf("\n\tImplementation of Kruskal's Algorithm\n");

    printf("\nEnter the no. of vertices:");

    scanf("%d",&n);

    printf("\nEnter the cost adjacency matrix:\n");
```

```

for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
}

printf("The edges of Minimum Cost Spanning Tree are\
n"); while(ne < n)
{
for(i=1,min=999;i<=n;i++)
{
for(j=1;j <= n;j++)
{
if(cost[i][j] < min)
{
min=cost[i][j];
a=u=i;
b=v=j;
}
}
}

u=find(u);

```

```

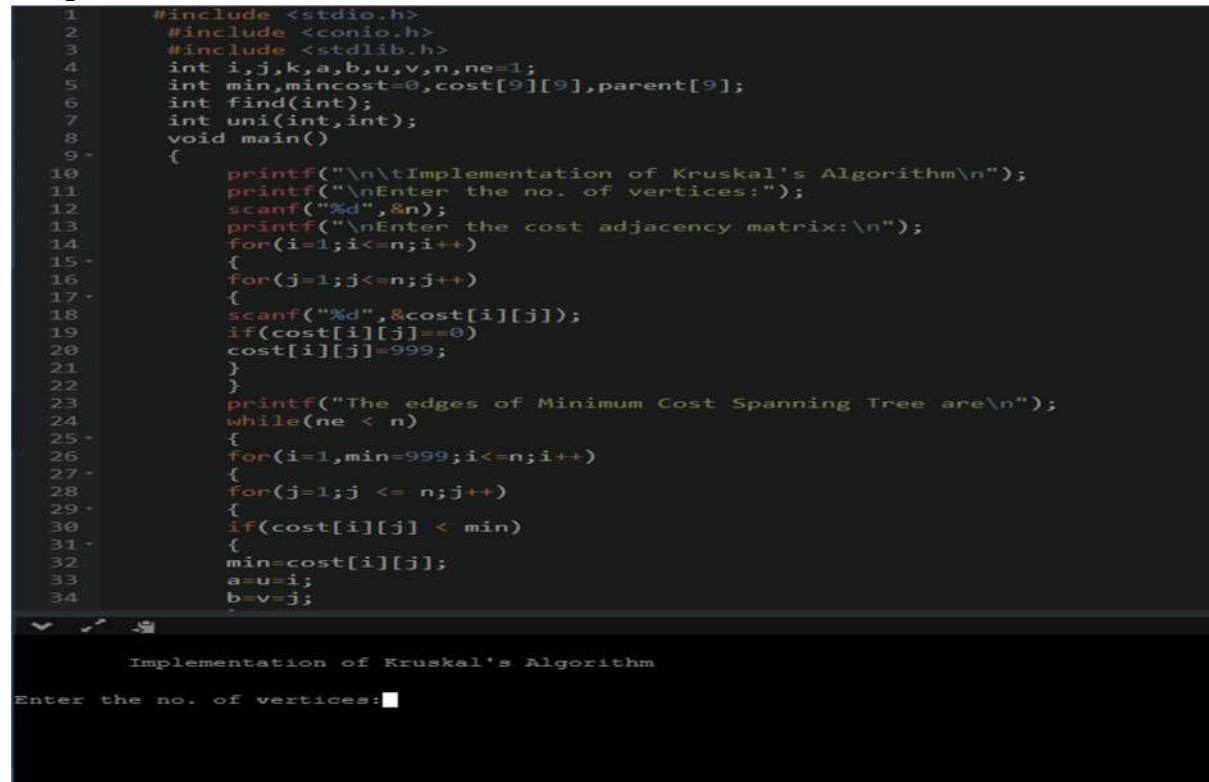
v=find(v);
if(uni(u,v))
{
printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
mincost +=min;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
getch();
}
int find(int i)
{
while(parent[i])
i=parent[i];
return i;
}
int uni(int i,int j)
{
if(i!=j)
{
parent[j]=i;
return 1;
}
return 0;
}

```

}

## Output-

```
1  #include <stdio.h>
2  #include <conio.h>
3  #include <stdlib.h>
4  int i,j,k,a,b,u,v,n,ne=1;
5  int min,mincost=0,cost[9][9],parent[9];
6  int find(int);
7  int uni(int,int);
8  void main()
9  {
10     printf("\n\tImplementation of Kruskal's Algorithm\n");
11     printf("\nEnter the no. of vertices:");
12     scanf("%d",&n);
13     printf("\nEnter the cost adjacency matrix:\n");
14     for(i=1;i<=n;i++)
15     {
16         for(j=1;j<=n;j++)
17         {
18             scanf("%d",&cost[i][j]);
19             if(cost[i][j]==0)
20                 cost[i][j]=999;
21         }
22     }
23     printf("The edges of Minimum Cost Spanning Tree are\n");
24     while(ne < n)
25     {
26         for(i=1,min=999;i<=n;i++)
27         {
28             for(j=1;j <= n;j++)
29             {
30                 if(cost[i][j] < min)
31                 {
32                     min=cost[i][j];
33                     a=u=i;
34                     b=v=j;
```



Implementation of Kruskal's Algorithm

Enter the no. of vertices:█

## Result-

The code has been written and the output has been verified.

## Experiment 8C

### Prim's Minimum Spanning Tree

#### Aim-

To write a code to demonstrate Prim's MST.

#### Algorithm-

1. Create edge list of given graph, with their weights.
2. Draw all nodes to create skeleton for spanning tree.
3. Select an edge with lowest weight and add it to skeleton and delete edge from edge list.
4. Add other edges. While adding an edge take care that the one end of the edge should always be in the skeleton tree and its cost should be minimum.
5. Repeat step 5 until n-1 edges are added.
6. Return.

#### Code-

```
#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("\n
spanning 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);
}

```

**Output-**

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  #define infinity 9999
5  #define MAX 20
6
7  int G[MAX][MAX],spanning[MAX][MAX],n;
8
9  int prims();
10
11 int main()
12 {
13     int i,j,total_cost;
14     printf("Enter no. of vertices:");
15     scanf("%d",&n);
16
17     printf("\nEnter the adjacency matrix:\n");
18
19     for(i=0;i<n;i++)
20         for(j=0;j<n;j++)
21             scanf("%d",&G[i][j]);
22
23     total_cost=prims();
24     printf("\nspanning tree matrix:\n");
25
26     for(i=0;i<n;i++)
27     {
28         printf("\n");
29         for(j=0;j<n;j++)
30             printf("%d\t",spanning[i][j]);
31     }
32
33     printf("\n\nTotal cost of spanning tree=%d",total_cost);
34     return 0;

```

input

Enter no. of vertices:█

## Result-

The code has been written and the output has been verified.

## Experiment 9

### Longest Common Subsequence

#### Aim-

To write a C program to print the longest common subsequence.

#### Algorithm-

- 1) Construct  $L[m+1][n+1]$
- 2) The value  $L[m][n]$  contains length of LCS. Create a character array `lcs[]` of length equal to the length of `lcs` plus 1 (one extra to store `\0`).
- 2) Traverse the 2D array starting from  $L[m][n]$ . Do following for every cell  $L[i][j]$ 
  - .....a) If characters (in X and Y) corresponding to  $L[i][j]$  are same (Or  $X[i-1] == Y[j-1]$ ), then include this character as part of LCS.
  - .....b) Else compare values of  $L[i-1][j]$  and  $L[i][j-1]$  and go in direction of greater value.

#### Code-

```
#include<stdio.h>

#include<string.h>

int i,j,m,n,c[20][20];

char x[20],y[20],b[20][20];

void print(int i,int j)

{

    if(i==0 || j==0)

        return;

    if(b[i][j]=='c')

    {

        print(i-1,j-1);

        printf("%c",x[i-1]);

    }

}
```

```

        else if(b[i][j]=='u')
            print(i-1,j);
        else
            print(i,j-1);
    }
void lcs()
{
    m=strlen(x);
    n=strlen(y);
    for(i=0;i<=m;i++)
        c[i][0]=0;
    for(i=0;i<=n;i++)
        c[0][i]=0;
    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++)
        {
            if(x[i-1]==y[j-1])
            {
                c[i][j]=c[i-1][j-1]+1;
                b[i][j]='c';
            }
            else if(c[i-1][j]>=c[i][j-1])
            {
                c[i][j]=c[i-1][j];
                b[i][j]='u';
            }
        }
    }
}

```

```
        }
        else
        {
            c[i][j]=c[i][j-1];
            b[i][j]='l';
        }
    }
}

int main()
{
    printf("Enter 1st sequence:");
    scanf("%s",x);
    printf("Enter 2nd sequence:");
    scanf("%s",y);
    printf("\nThe Longest Common Subsequence is ");
    lcs();
    print(m,n);
    return 0;
}
```

Output-

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int i, j, m, n, c[20][20];
5
6  char x[20], y[20], b[20][20];
7
8
9  void
10 print (int i, int j)
11 {
12
13     if (i == 0 || j == 0)
14         return;
15
16     if (b[i][j] == 'c')
17     {
18
19         {
20
21             print (i - 1, j - 1);
22
23             printf ("%c", x[i - 1]);
24
25         }
26
27         else if (b[i][j] == 'u')
28
29             print (i - 1, j);
30
31         else
32
33             print (i, j - 1);
34 }
```

Enter 1st sequence:

Result-

The code has been written and the output has been verified.

## Experiment 10

### N Queen problem

#### Aim-

To write a C program to display and explain the N Queens problem.

#### Algorithm-

- 1) Start in the leftmost column
- 2) If all queens are placed  
    return true
- 3) Try all rows in the current  
    column. Do following for every  
    tried row.
  - a) If the queen can be placed safely in this  
    row then mark this [row, column] as part of  
    the solution and recursively check if placing  
    queen here leads to a solution.
  - b) If placing the queen in [row, column] leads  
    to a solution then return true.
  - c) If placing queen doesn't lead to a solution then  
    unmark this [row, column] (Backtrack) and go  
    to step (a) to try other rows.
- 3) If all rows have been tried and nothing  
    worked, return false to trigger backtracking.

#### Code-

```
#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) //for nxn board
{
    if(board[i]==j)
        printf("\tQ"); //queen at i,j position
    else
        printf("\t-"); //empty slot
}
}
}

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; //no conflicts
}

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);
    }
}
}
```

**Output-**

```
1 #include<stdio.h>
2 #include<math.h>
3
4 int board[20],count;
5
6 int main()
7 {
8     int n,i,j;
9     void queen(int row,int n);
10
11     printf(" - N Queens Problem Using Backtracking -");
12     printf("\n\nEnter number of Queens:");
13     scanf("%d",&n);
14     queen(1,n);
15     return 0;
16 }
17 void print(int n)
18 {
19     int i,j;
20     printf("\n\nSolution %d:\n\n",++count);
21
22     for(i=1;i<=n;++i)
23         printf("\t%d",i);
24
25     for(i=1;i<=n;++i)
26     {
27         printf("\n\n%d",i);
28         for(j=1;j<=n;++j) //for nxn board
29         {
30             if(board[i]==j)
31                 printf("\tQ"); //queen at i,j position
32             else
33                 printf("\t-"); //empty slot
34         }
35     }
36 }
```

main.c:45:7: warning: implicit declaration of function 'abs' [-Wimplicit-function-declaration]

- N Queens Problem Using Backtracking -

Enter number of Queens:

## Result-

The code has been written and the output has been verified.

## EX.NO.11

### TRAVELLING SALESMAN PROBLEM

#### **Aim:**

To write a C program to solve the travelling sales man problem using the dynamic programming approach.

#### **Algorithm:**

- Step1: Start the process
- Step2: Enter the number of cities
- Step3: Enter the cost matrix of all the cities
- Step4: Find all possible feasible solutions by taking the permutation of the cities which is to be covered.
- Step5: Find the cost of each path using the cost matrix.
- Step6: Find out the path with minimum cost.
- Step7: If more than one path having the same cost considers the first occurring path.
- Step8: That is selected as the optimum solution.
- Step9: Stop the process.

#### **Program:**

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int a[10][10],visited[10],n,cost=0;
void get()
{
    int i,j;
    printf("\n\nEnter Number of Cities: ");
    scanf("%d",&n);
    printf("\nEnter Cost Matrix: \n");
    for( i=0;i<n;i++)
    {
        printf("\n Enter Elements of Row # : %d\n",i+1);
        for( j=0;j<n;j++)
            scanf("%d",&a[i][j]);
        visited[i]=0;
    }
    printf("\n\nThe Cost Matrix is:\n");
    for( i=0;i<n;i++)
    {
        printf("\n\n");
        for(j=0;j<n;j++)
            printf("\t%d",a[i][j]);
    }
}
```

```

void mincost(int city)
{
    int i,ncity,least(int city);
    visited[city]=1;
    printf("%d ==> ",city+1);
    ncity=least(city);
    if(ncity==999)
    {
        ncity=0;
        printf("%d",ncity+1);
        cost+=a[city][ncity];
        return;
    }
    mincost(ncity);
}
int least(int c)
{
    int i,nc=999;
    int min=999,kmin;
    for(i=0;i<n;i++)
    {
        if((a[c][i]!=0)&&(visited[i]==0))
        if(a[c][i]<min)
        {
            min=a[i][0]+a[c][i];
            kmin=a[c][i];
            nc=i;
        }
    }
    if(min!=999)
    cost+=kmin;
    return nc;
}
void put()
{
    printf("\n\nMinimum cost:");
    printf("%d",cost);
}
void main()
{
    clrscr();
    get();
    printf("\n\nThe Path is:\n\n");
    mincost(0);
    put();
    getch();
}

```

}

## OUTPUT

The Cost Matrix is:

14	15	18	34	98	12
45	67	81	34	12	8
23	86	15	3	57	34
47	92	31	7	68	39
95	85	10	55	72	43
63	86	93	56	13	49

The Path is:

1 ==> 6 ==> 5 ==> 3 ==> 4 ==> 2 ==> 1

Minimum cost:175\_

## RESULT

Thus the travelling Salesman problem using the dynamic programming approach was executed successfully.

**EX.NO.12a****BFS Implementation with Array****Aim:**

To write a C program to solve the BFS problem with array.

**Algorithm:**

Step1: Start by putting any one of the graph's vertices at the back of a queue.

Step2: Take the front item of the queue and add it to the visited list.

Step3: Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.

Step4: Keep repeating steps 2 and 3 until the queue is empty.

Step5: The graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the BFS algorithm on every node

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 100
#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);
int queue[MAX], front = -1, rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
{
    create_graph();
    BF_Traversal();
    return 0;
}

void BF_Traversal()
{

```

```

    int v;

    for(v=0; v<n; v++)
        state[v] = initial;

    printf("Enter Start Vertex for BFS: \n");
    scanf("%d", &v);
    BFS(v);
}

void BFS(int v)
{
    int i;
    insert_queue(v);
    state[v] = waiting;
    while(!isEmpty_queue())
    {
        v = delete_queue( );
        printf("%d ",v);
        state[v] = visited;

        for(i=0; i<n; i++)
        {
            if(adj[v][i] == 1 && state[i] == initial)
            {
                insert_queue(i);
                state[i] = waiting;
            }
        }
    }
    printf("\n");
}

void insert_queue(int vertex)
{
    if(rear == MAX-1)
        printf("Queue Overflow\n");
    else
    {
        if(front == -1)
            front = 0;
        rear = rear+1;
        queue[rear] = vertex ;
    }
}

```



```

int isEmpty_queue()
{
    if(front == -1 || front >
        rear) return 1;
    else
        return 0;
}

int delete_queue()
{
    int delete_item;
    if(front == -1 || front > rear)
    {
        printf("Queue Underflow\n");
        exit(1);
    }

    delete_item = queue[front];
    front = front+1;
    return delete_item;
}

void create_graph()
{
    int count,max_edge,origin,destin;

    printf("Enter number of vertices : ");
    scanf("%d",&n);
    max_edge = n*(n-1);

    for(count=1; count<=max_edge; count++)
    {
        printf("Enter edge %d( -1 -1 to quit ) :
            ",count); scanf("%d %d",&origin,&destin);

        if((origin == -1) && (destin == -1))
            break;

        if(origin>=n || destin>=n || origin<0 || destin<0)
        {
            printf("Invalid edge!\n");
            count--;
        }
        else
        {
            adj[origin][destin] = 1;

```

## OUTPUT

```
tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$ ./a.out
Enter number of vertices : 9
Enter edge 1( -1 -1 to quit ) : 0
1
Enter edge 2( -1 -1 to quit ) : 0
3
Enter edge 3( -1 -1 to quit ) : 0
4
Enter edge 4( -1 -1 to quit ) : 1
2
Enter edge 5( -1 -1 to quit ) : 3
6
Enter edge 6( -1 -1 to quit ) : 4
7
Enter edge 7( -1 -1 to quit ) : 6
4
Enter edge 8( -1 -1 to quit ) : 6
7
Enter edge 9( -1 -1 to quit ) : 2
5
Enter edge 10( -1 -1 to quit ) : 4
5
Enter edge 11( -1 -1 to quit ) : 7
5
Enter edge 12( -1 -1 to quit ) : 7
8
Enter edge 13( -1 -1 to quit ) : -1
-1
Enter Start Vertex for BFS:
0
0 1 3 4 2 6 5 7 8
tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$
```

## RESULT

Thus the BFS problem using array was executed successfully.

**EX.NO.12b****DFS Implementation with Array****Aim:**

To write a C program to solve the DFS problem with array implementation.

**Algorithm:**

Step1: Start by putting any one of the graph's vertices on top of a stack.

Step2: Take the top item of the stack and add it to the visited list.

Step3: Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.

Step4: Keep repeating steps 2 and 3 until the stack is empty.

**Program:**

```
#include <stdio.h>
#include <stdlib.h>
/*      ADJACENCY MATRIX      */
int source,V,E,time,visited[20],G[20][20];
void DFS(int i)
{
    int j;
    visited[i]=1;
    printf(" %d->",i+1);
    for(j=0;j<V;j++)
    {
        if(G[i][j]==1&&visited[j]==0)
            DFS(j);
    }
}
int main()
{
    int i,j,v1,v2; printf("\t\t\t\t\t\n"); printf("Enter the\n"); printf("no of edges:");
    scanf("%d",&E);
    printf("Enter the no of vertices:");
    scanf("%d",&V); for(i=0;i<V;i++)
    {
        for(j=0;j<V;j++)
            G[i][j]=0;
    }
    /*      creating edges :P      */
    for(i=0;i<E;i++)
```

```

{
    printf("Enter the edges (format: V1 V2) : ");
    scanf("%d%d",&v1,&v2);
    G[v1-1][v2-1]=1;
}
for(i=0;i<V;i++)
{
    for(j=0;j<V;j++)
        printf(" %d ",G[i][j]);
    printf("\n");
}
printf("Enter the source: ");
scanf("%d",&source);
DFS(source-1);
return 0;
}

```

**Output:**

```

E:\CodeBlocks\share\CodeBlocks\graphs\bin\Debug\graphs.exe
Graphs
Enter the no of edges:11
Enter the no of vertices:10
Enter the edges <format: V1 U2> : 1 2
Enter the edges <format: V1 U2> : 1 3
Enter the edges <format: V1 U2> : 2 4
Enter the edges <format: V1 U2> : 2 5
Enter the edges <format: V1 U2> : 3 6
Enter the edges <format: V1 U2> : 3 7
Enter the edges <format: V1 U2> : 4 8
Enter the edges <format: V1 U2> : 5 9
Enter the edges <format: V1 U2> : 6 10
Enter the edges <format: V1 U2> : 8 9
Enter the edges <format: V1 U2> : 9 10
0 1 1 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
Enter the source: 1
1-> 2-> 4-> 8-> 9-> 10-> 5-> 3-> 6-> 7->
Process returned 0 (0x0) execution time : 42.232 s
Press any key to continue.

```

**RESULT**

Thus the BFS problem using array was executed successfully.

## EX.NO.13

### Randomized Quick Sort

#### **Aim:**

To write a C program to solve the Randomized quick sort.

#### **Algorithm:**

QUICKSORT(A,p,r)

    if  $p < r$

        then  $q \leftarrow \text{PARTITION}(A,p,r)$

        QUICKSORT(A,p,q)

        QUICKSORT(A,q + 1,r)

To sort an entire array A, the initial call is QUICKSORT(A, 1, length[A]).

#### **Partitioning the array**

The key to the algorithm is the PARTITION procedure, which rearranges the subarray A[p . . r] in place.

PARTITION(A,p,r)

$x \leftarrow A[p]$

$i \leftarrow p - 1$

$j \leftarrow r + 1$

    while TRUE

do repeat  $j \leftarrow j - 1$

until  $A[j] \leq x$

    repeat  $i \leftarrow i + 1$

until  $A[i] \geq x$

    if  $i < j$

        then exchange  $A[i] \leftrightarrow A[j]$

    else return j

RANDOMIZED-QUICKSORT(A,p,r)

    if  $p < r$

        then  $q \leftarrow \text{RANDOMIZED-PARTITION}(A,p,r)$

        RANDOMIZED-QUICKSORT(A,p,q)

        RANDOMIZED-QUICKSORT(A,q + 1,r)

#### **Program:**

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

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

```
#define MAX 100
```

```
void random_shuffle(int arr[])
```

```
{
```

```
    srand(time(NULL));
```

```
    int i, j, temp;
```

```

    for (i = MAX - 1; i > 0; i--)
    {
        j = rand()%(i + 1);
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

void swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

int partion(int arr[], int p, int r)
{
    int pivotIndex = p + rand()%(r - p + 1); //generates a random number as a pivot
    int pivot;
    int i = p - 1;
    int j;
    pivot = arr[pivotIndex];
    swap(&arr[pivotIndex], &arr[r]);
    for (j = p; j < r; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i+1], &arr[r]);
    return i + 1;
}

void quick_sort(int arr[], int p, int q)
{
    int j;
    if (p < q)
    {
        j = partion(arr, p, q);
        quick_sort(arr, p, j-1);
        quick_sort(arr, j+1, q);
    }
}

```

```

}
int main()
{
    int i;
    int arr[MAX];
    for (i = 0; i < MAX; i++)
        arr[i] = i;
    random_shuffle(arr); //To randomize the array
    quick_sort(arr, 0, MAX-1); //function to sort the elements of
    array for (i = 0; i < MAX; i++)
        printf("%d \n", arr[i]);
    return 0;
}

```

### **Output:**

```

$ gcc randomizedquicksort.c -o randomizedquicksort
$ ./randomizedquicksort

```

Sorted array is : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26  
 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57  
 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88  
 89 90 91 92 93 94 95 96 97 98 99

### **RESULT**

Thus the Randomized quick sort problem using array was executed successfully.



## EX.NO.14

### String Matching Algorithm

#### **Aim:**

To write a C program to solve the String Matching algorithm.

#### **Algorithm:**

Start

```
pat_len := pattern Size
str_len := string size
for i := 0 to (str_len - pat_len), do
    for j := 0 to pat_len, do
        if text[i+j] ≠ pattern[j], then
            break
    if j == patLen, then
        display the position i, as there pattern found
```

End

#### **Program:**

```
#include <stdio.h>
#include <string.h>
int match(char [], char []);

int main() {
    char a[100], b[100];
    int position;

    printf("Enter some text\n");
    gets(a);

    printf("Enter a string to find\n");
    gets(b);

    position = match(a, b);

    if (position != -1) {
        printf("Found at location: %d\n", position + 1);
    }
    else {
        printf("Not found.\n");
    }

    return 0;
}
```

```

int match(char text[], char pattern[]) {
    int c, d, e, text_length, pattern_length, position = -1;

    text_length  = strlen(text);
    pattern_length =
    strlen(pattern);

    if (pattern_length > text_length) {
        return -1;
    }

    for (c = 0; c <= text_length - pattern_length; c++)
        { position = e = c;

        for (d = 0; d < pattern_length; d++) {
            if (pattern[d] == text[e]) {
                e++;
            }
            else {
                break;
            }
        }
        if (d == pattern_length) {
            return position;
        }
    }

    return -1;
}

```

## Output



A screenshot of a Windows command prompt window. The title bar shows the file path "E:\programmingsimplified.com\c\pattern-matching.exe". The command prompt contains the following text: "Enter some text", "computer programming is fun", "Enter a string to find", "programming is fun", and "Found at location 10". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons.

```
E:\programmingsimplified.com\c\pattern-matching.exe
Enter some text
computer programming is fun
Enter a string to find
programming is fun
Found at location 10
```

## RESULT

Thus the String matching algorithm was executed successfully.

## EX.NO.15

### Analysing - Real Time Problem

#### **Aim:**

To write a C program to sort an array using Merge sort and manipulate the time complexity of the program.

#### **Algorithm:**

Step1: Mergesort(A[0 .. n - 1])

Step2: Sorts array A[0 .. n - 1] by recursive mergesort

Step3: Input: An array A[0 .. n - 1] of orderable elements

Step4: Output: Array A[0 .. n - 1] sorted in nondecreasing

order Step5: Merge(B[0 .. p- 1], C[0 .. q -1], A[0.. p + q -1])

Step6: Merges two sorted arrays into one sorted array

#### **Program:**

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

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

```
#include <sys/time.h>
```

```
#include <omp.h>
```

```
void simplemerge(int a[], int low, int mid, int high)
```

```
{
```

```
    int i,j,k,c[20000];
```

```
    i=low;
```

```
    j=mid+1;
```

```
    k=low;
```

```
    int tid;
```

```
    omp_set_num_threads(10);
```

```
{
    tid=omp_get_thread_num();
    while(i<=mid&& j<=high)
    {
        if(a[i] < a[j])
        {
            c[k]=a[i];
            //printf("%d%d",tid,c[k]);

            i++;
            k++;
        }
        else
        {
            c[k]=a[j];
            //printf("%d%d", tid, c[k]);

            j++;
            k++;
        }
    }
}
while(i<=mid)
{
    c[k]=a[i];
    i++;
    k++;
}
```

```

    }
    while(j<=high)
    {
        c[k]=a[j];
        j++;
        k++;
    }
    for(k=low;k<=high;k++)
        a[k]=c[k];
}

void merge(int a[],int low,int high)
{
    int mid;
    if(low <
    high)
    {
        mid=(low+high)/2;
        merge(a,low,mid);
        merge(a,mid+1,high);
        simplemerge(a,low,mid,high);
    }
}

void getnumber(int a[], int n)
{
    int i;
    for(i=0;i < n;i++)

```

```

        a[i]=rand()%100;
    }

int main()
{
    FILE *fp;

    int a[2000],i;

    struct timeval tv;

    double start, end, elapse;

    fp=fopen("mergesort.txt","w");

    for(i=10;i<=1000;i+=10)
    {
        getnumber(a,i);

        gettimeofday(&tv,NULL);

        start=tv.tv_sec+(tv.tv_usec/1000000.0);

        merge(a,0,i-1);

        gettimeofday(&tv,NULL);

        end=tv.tv_sec+(tv.tv_usec/1000000.0);

        elapse=end-start; fprintf(fp,"%d\t%lf\n",i,elapse);
    }

    fclose(fp);

    system("gnuplot");

    return 0;
}

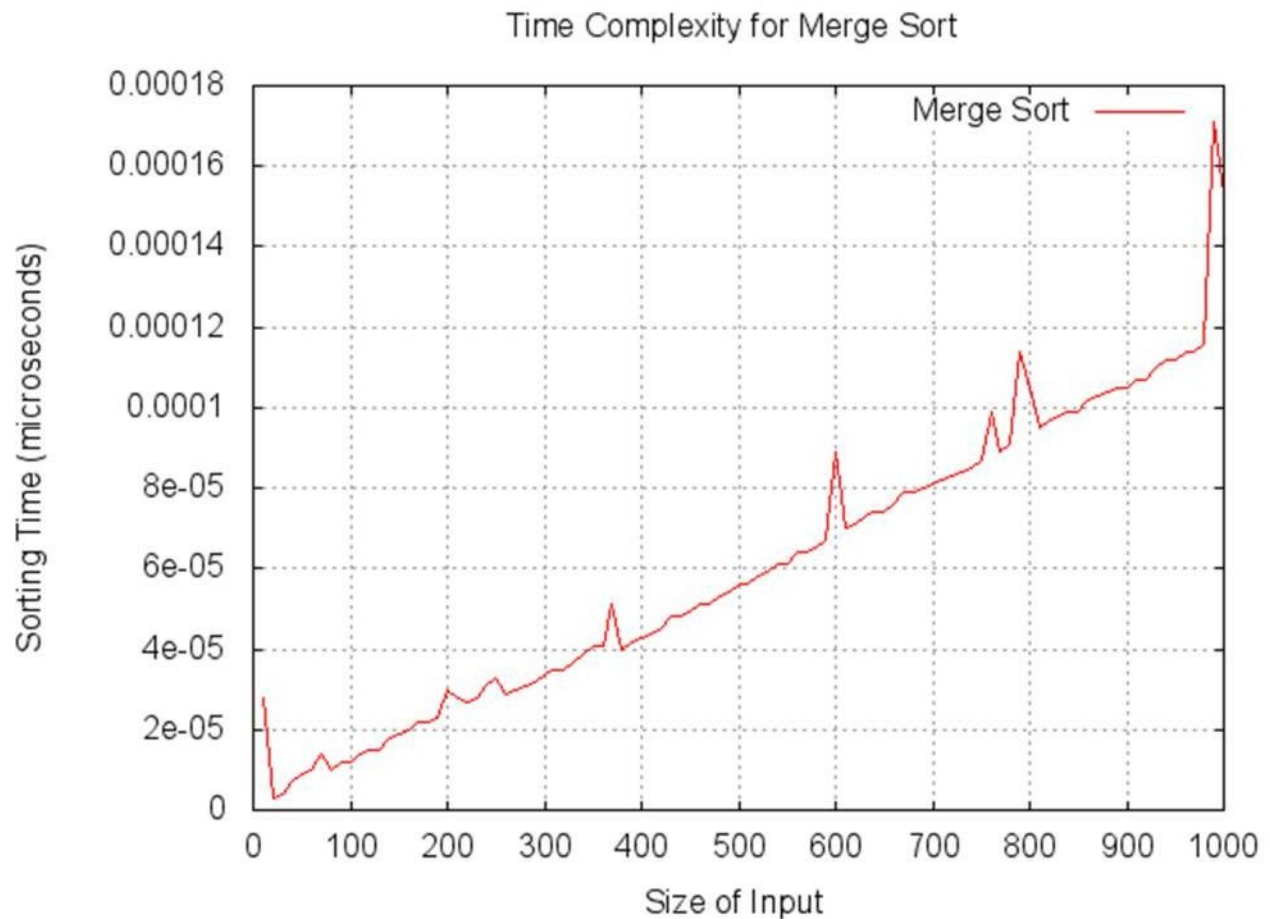
```

**mergesort.gpl**

**Gnuplot script file for plotting data in file "mergesort.txt" This file is called mergesort.gpl**

```
set terminal png font arial
set title "Time Complexity for Merge Sort"
set autoscale
set xlabel "Size of Input"
set ylabel "Sorting Time
(microseconds)" set grid
set output "mergesort.png"
plot "mergesort.txt" t "Merge Sort" with lines
```

## OUTPUT





## **RESULT**

Thus the merge sort program was executed successfully with the time complexity.