INDEX

ASSIGNMENT 1

1. **Design and implement linear search and binary search algorithms. Analyze the efficiencies of algorithms. 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.**

   Program:

```
//Linear search algorithm
#include<stdio.h> int n;
int linearsearch(int a[],int k)
{
 int i=0;
 while(i<n&&a[i]!=k)
 {
            i++;
 }
 if(i<n)
return i;
else
return -1;
} int
main() {
int i,k;
 printf("enter size of array:\n");
 scanf("%d",&n);
int a[n];
 printf("enter array elements\n");
for(i=0;i<n;i++)
 {
            scanf("%d",&a[i]);
 }
 printf("enter ele to search:\n");
scanf("%d",&k);  int
res=linearsearch(a,k);  if(res!=-
1)
 printf("ele %d found at index %d\n",k,res);  else
 printf("ele %d not present\n",k);
 return 0;
}
```

   Output: enter size
   of array: 5 enter
   array elements
   1
   9
   8

4 5 enter ele to
search: -7 ele -7
not present

Program:

```
//Binary search algorithm
#include<stdio.h> int n;
void sort(int a[],int n) {
 int i,j;
 for(i=0;i<n-1;i++)
 {
		for(j=0;j<n-1-i;j++)
		{
			if(a[j+1]<a[j])
			{
				int temp=a[j];
				a[j]=a[j+1];
				a[j+1]=temp;
			}
		}
 }
}
int bin_srch(int a[],int k)
{  int l=0,r=n-
1,m;
while(l<=r)
 {
		m=l+(r-l)/2;
 if(k==a[m])	return m;
 else
		{
			if(k<a[m])
				r=m-1;
		else
 l=m+1;
		}
 }  return -1; } int main() {  int
i,k;  printf("enter size of
array:\n");
 scanf("%d",&n);  int a[n];
printf("enter array elements\n");
for(i=0;i<n;i++)
		scanf("%d",&a[i]);
```

```
printf("enter ele to search:\n");
scanf("%d",&k);
sort(a,n);  printf("sorted
array:\n");
for(i=0;i<n;i++)
 printf("%d ",a[i]);
int res=bin_srch(a,k);  if(res!=-
1)
printf("\nele %d found at index %d\n",k,res);
else
printf("\nele %d not present\n",k);
return 0; }
```

Output:

enter size of array: 3
enter array elements
1 2 3 enter ele to
search: 5
ele 5 not present

ASSIGNMENT 2

2. Sort a given set of elements using the bubble sort and selection sort method and determine the time required to sort the elements. 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.

Program:

```
//Bubble sort algorithm
#include<stdio.h> void
sort(int a[],int n)
{
 int i,j;
 for(i=0;i<n-1;i++)
 {
        for(j=0;j<n-1-i;j++)
        {
               if(a[j+1]<a[j])
               {
                      int temp=a[j];
             a[j]=a[j+1];
             a[j+1]=temp;
               }
        }
 }
```

```
} int
main() {
int i,n;
 printf("enter size of array:\n");
 scanf("%d",&n);
int a[n];
 printf("enter array elements\n");
for(i=0;i<n;i++)
 {
        scanf("%d",&a[i]);
 }
 printf("original array:\n");
for(i=0;i<n;i++)
 {
        printf("%d ",a[i]);
 }
 sort(a,n);  printf("\nsorted
array:\n");
for(i=0;i<n;i++)
 {
        printf("%d ",a[i]);
 }  return 0; }
```

Output: enter size
of array: 5 enter
array elements
1 8 4 -9
9 original
array: 1 8 4 -9
9
sorted array: -
9 1 4 8 9

Program:

```
//selection sort algorithm
#include<stdio.h> void
sort(int a[],int n) {  int
min,i,j;  for(i=0;i<n-
1;i++)
 {
        min=i;
        for(j=i+1;j<n;j++)
        {
                if(a[j]<a[min])
```

```
                min=j;
            }
            int temp=a[i];
  a[i]=a[min];    a[min]=temp;
  }
} int main() {  int i,n;
printf("enter size of array:\n");
 scanf("%d",&n);  int a[n];
printf("enter array elements\n");
for(i=0;i<n;i++)
 {
            scanf("%d",&a[i]);
 }  printf("original
array:\n");
for(i=0;i<n;i++)
 {
            printf("%d ",a[i]);
 }  sort(a,n);
printf("\nsorted array:\n");
for(i=0;i<n;i++)
 {
            printf("%d ",a[i]);
 }  return 0; }
```

Output: enter size
of array: 5 enter
array elements
9 4 -10 5
6 original
array: 9 4 -10
5 6 sorted
array:
-10 4 5 6 9

ASSIGNMENT 3

3. Sort a given set of elements using the quick sort method and determine the time required to sort the elements. 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.

Program:

```
//quick sort algorithm
#include<stdio.h> void
swap(int *a, int *b){
            int temp=*a;
```

```
        *a=*b;
        *b=temp;
}
int partition(int a[],int l,int h)
{
 int pivot=a[l],i=l+1,j=h;
while(i<j){
        while(a[i]<=pivot && i<=h-1)
            i++;
        while(a[j]>pivot && j>=l+1)
            j--;
        if(i<j)
            swap(&a[i],&a[j]);
 }
 swap(&a[l],&a[j]);
 return j; }
void quick_sort(int a[],int l,int h)
{
if(l<h)
 {
        int pi=partition(a,l,h);
  quick_sort(a,l,pi-1);
        quick_sort(a,pi+1,h);
 } } int
main() {
int i,n;
 printf("enter size of array:\n");
scanf("%d",&n);
 int a[n];  printf("enter array
elements\n");  for(i=0;i<n;i++)
 {
        scanf("%d",&a[i]);
 }  printf("original
array:\n");
for(i=0;i<n;i++)
 {
        printf("%d\t",a[i]);
 }  quick_sort(a,0,n-1);
printf("\nsorted array:\n");
for(i=0;i<n;i++)
 {
        printf("%d\t ",a[i]);
```

```
 }  return
0; }
```

Output: enter size
of array: 5 enter
array elements
9
4
-10
7   3   original
array: 9 4 -10
7   3   sorted
array: -10 3 4
7 9

ASSIGNMENT 4

4. **Implement 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, the number of elements in the list to be sorted and plot a graph of the time taken versus n.**
   Program:

```
//merge sort algorithm #include<stdio.h>
void merge(int a[],int l,int m,int h)
{  int i,j,k=l;  int p=m-
l+1;  int q=h-m;  int
b[p],c[q];
for(i=0;i<p;i++)
b[i]=a[l+i];
for(j=0;j<q;j++)
c[j]=a[m+j+1];
i=0,j=0;  while(i<p
&& j<q){
 if(b[i]<=c[j]){
               a[k]=b[i];
               i++;
        }
```

```
            else{
 a[k]=c[j];
                        j++;
            }
            k++;
 }
while(i<p){
                        a[k]=b[i];
                 i++;
     k++;
            }
 while(j<q){
                        a[k]=c[j];
                 j++;
     k++;
            }}
void mergesort(int a[],int low,int high){
if(low<high){
            int m=low+(high-low)/2;
 mergesort(a,low,m);    mergesort(a,m+1,high);
            merge(a,low,m,high);
 } } int main(){  int i,n;
printf("enter size of array:\n");
 scanf("%d",&n);  int a[n];
printf("enter array elements\n");
 for(i=0;i<n;i++)
            scanf("%d",&a[i]);
 printf("original array:\n");
for(i=0;i<n;i++)
 printf("%d\t",a[i]);
mergesort(a,0,n-1);
printf("\nsorted array:\n");
for(i=0;i<n;i++)
 printf("%d\t ",a[i]);
 return 0; }
```

Output: enter size
of array: 5 enter
array elements
7 -10 6 5
9 original
array:
7 -10 6 5 9 sorted
array:

-10 5 6 7 9

ASSIGNMENT 5

5. Implement the following algorithms to

- **a. Print all the nodes reachable from a given starting node in digraph using BFS method.**
- **b. Check whether a given graph is connected or not using DFS method.**

Program a:

```
//bfs algorithm
#include<stdio.h> int
a[20][20],visited[20],n;
void bfs(int s){  int
queue[20];  int f=-1,r=-1;
queue[++r]=s;
visited[s]=1;
 while(f!=r){
		int curr=queue[++f];
 printf("%d",curr);	for(int
i=0;i<n;i++){
			if(!visited[i] && a[curr][i]==1){
				queue[++r]=i;
				visited[i]=1;
			}
		}
		if(f!=r)
			printf("-->");
 }
 printf("\n");
}

int main(){
int s;
 printf("Enter number of nodes: ");
scanf("%d",&n);
 printf("Enter adjacency matrix\n");
for(int i=0;i<n;i++){
		for(int j=0;j<n;j++){
 printf("a[%d][%d]: ",i,j);
			scanf("%d",&a[i][j]);
		}
 }
 printf("Enter the starting vertex:");
```

```
 scanf("%d",&s);
 printf("BFS traversal starting from vertex %d: ",s);
bfs(s);  return 0;
}
```

Output:

Enter number of nodes: 3
Enter adjacency matrix
a[0][0]: 0 a[0][1]: 1
a[0][2]: 0 a[1][0]: 0
a[1][1]: 0 a[1][2]: 1
a[2][0]: 1 a[2][1]: 0
a[2][2]: 0 Enter the
starting vertex:1
BFS traversal starting from vertex 1: 2-->3-->1

Program b:

```
//dfs algorithm
#include<stdio.h> int
a[20][20],visited[20],n;
void dfs(int s){
printf("%d",s+1);
visited[s]=1;
 for(int j=0;j<n;j++)      if(!visited[j]
&& a[s][j]==1){
                printf("-->");
  dfs(j);
        }
} int isconnected(){
for(int i=0;i<n;i++){
        if(!visited[i])return 0;
 }  return 1; } int main(){
printf("Enter number of nodes: ");
scanf("%d",&n);  printf("Enter
adjacency matrix\n");  for(int
i=0;i<n;i++){
        for(int j=0;j<n;j++){
  printf("a[%d][%d]: ",i,j);
                scanf("%d",&a[i][j]);
        }
 }  for(int
i=0;i<n;i++)
  visited[i]=0;
 printf("DFS traversal starting from node 1: ");
 dfs(0);
```

```
 if(isconnected())          printf("\nGraph is
connected\n");  else printf("\nGraph is not
connected\n");
 return 0;
}
```

Output:

```
Enter number of nodes: 3
Enter adjacency matrix
a[0][0]: 0 a[0][1]: 1
a[0][2]: 0 a[1][0]: 1
a[1][1]: 0 a[1][2]: 1
a[2][0]: 0 a[2][1]: 1
a[2][2]: 0
DFS traversal starting from node 1: 1-->2-->3
Graph is connected
```

ASSIGNMENT 6

6. Implement Horspool string matching algorithm to search for a pattern in the text.

Program:

```
//horspool algorithm
#include<stdio.h>
#include<string.h> #define
MAX 256
void shiftTable(char pattern[],int table[]){
int m=strlen(pattern);     for(int
i=0;i<MAX;i++)        table[i]=m;     for(int
i=0;i<m-1;i++)        table[(unsigned
char)pattern[i]]=m-i-1;} int
horspoolSearch(char text[],char pattern[]){
   int n=strlen(text);     int
m=strlen(pattern);  int
table[MAX];
shiftTable(pattern,table);
int i=m-1;     while(i<n){
int j=0;
      while(j<m && text[i-j]==pattern[m-1-j])
j++;        if(j==m)             return i-m+1;
      i+=table[(unsigned char)text[i]];    }
return -1;} int main(){     char
text[MAX];     char pattern[50];
   printf("Enter the string (text):");
   gets(text);     printf("Enter the
pattern:");     gets(pattern);     int
result=horspoolSearch(text,pattern);
   if (result!=-1)
      printf("Pattern found at position %d\n",result);
else
      printf("Pattern not found\n");
   return 0; }
```

Output:

```
Enter the string (text):Hello world
Enter the pattern:ello
Pattern found at position 1
```

ASSIGNMENT 7

7. Implement the following algorithms to

a. Compute the transitive closure of a given directed graph using Warshall's algorithm.

b. Compute the all pairs shortest path matrix using Floyd's algorithm.

Program a:

```
//warshell algorithm
#include<stdio.h> void
```

```
warshell(int n, int a[][n]){
for(int i=0;i<n;i++){
          for(int j=0;j<n;j++){
                    for(int k=0;k<n;k++){
                              a[i][j] = a[i][j] || (a[i][k] && a[k][j]);
                    }
          }
} } int
main(){
int n;
 printf("Enter number of nodes: ");
 scanf("%d",&n);
int a[n][n];
 printf("Enter adjacency matrix\n");
for(int i=0;i<n;i++){
          for(int j=0;j<n;j++){
          printf("a[%d][%d]: ",i,j);
                    scanf("%d",&a[i][j]);
          }
 }
 printf("Adjacency matrix:\n");
for(int i=0;i<n;i++){
          for(int j=0;j<n;j++){
                    printf("%d\t",a[i][j]);
          }
          printf("\n");
 }
warshell(n,a);
 printf("Adjacency matrix:\n");
for(int i=0;i<n;i++){
          for(int j=0;j<n;j++){
                    printf("%d\t",a[i][j]);
          }
          printf("\n");
 }
return 0;
}
```

Output:

```
Enter number of nodes:
3 Enter adjacency matrix
a[0][0]: 0 a[0][1]: 1
a[0][2]: 0 a[1][0]: 0
a[1][1]: 0 a[1][2]: 1
a[2][0]: 1 a[2][1]: 0
```

a[2][2]: 0 Adjacency
matrix:

```
0	1	0
0	0	1
1	0	0
```

Updated Adjacency matrix:

```
0	1	1
1	1	1
1	1	1
```

Program b:

```
//floyd algorithm
#include<stdio.h>
int n,dist[20][20];
void floyd(){
 for(int i=0;i<n;i++){
		for(int j=0;j<n;j++){
			for(int k=0;k<n;k++){
		if(dist[i][k]+dist[k][j]<dist[i][j])
					dist[i][j]=dist[i][k]+dist[k][j];
			}
		}
 }
}

void printmat(){
 for (int i=0;i<n;i++){
		for(int j=0;j<n;j++){
			printf("%d\t",dist[i][j]);
		}
		printf("\n");
 } } int
main(){
printf("Ent
er number
of
vertices:");
scanf("%d
",&n);
for(int
i=0;i<n;i+
+){
		for(int j=0;j<n;j++){
		printf("dist[%d][%d]:",i,j);
```

```
                    scanf("%d",&dist[i][j]);
            }
 }  printf("\n\nCost
matrix:\n");  printmat();
floyd();
 printf("\n\nShortest distance
matrix:\n");  printmat();  return 0; }
```

Output:

```
Enter number of
vertices:4 dist[0][0]:0
dist[0][1]:3 dist[0][2]:999
dist[0][3]:7 dist[1][0]:999
dist[1][1]:0 dist[1][2]:1
dist[1][3]:5 dist[2][0]:999
dist[2][1]:999 dist[2][2]:0
dist[2][3]:2 dist[3][0]:999
dist[3][1]:999
dist[3][2]:999 dist[3][3]:0
Cost matrix:
0      3      999    7
999    0      1      5
999    999    0      2
999    999    999    0 Shortest
distance matrix:
0      3      4      6
999    0      1      3
999    999    0      2
999    999    999    0
```

ASSIGNMENT 8

8. Implement Knapsack problem using Dynamic Programming approach.

Program:

```
//knapsack algorithm
#include<stdio.h> int
max(int a,int b){
return a>b?a:b;
}
void knapSack(int w, int wt[], int val[], int n){
 int k[n+1][w+1];
 for(int i=0;i<=n;i++){
```

```
 for(int j=0;j<=w;j++){    if(i==0 ||
j==0)
                                k[i][j]=0;
                      else if(wt[i-1]<=j)
                                k[i][j]=max(val[i-1]+k[i-1][j-wt[i-1]], k[i-1][j]);
                      else
                                k[i][j]=k[i-1][j];
           }
 }  printf("\nKnapsack
matrix:\n");  for(int
i=0;i<=n;i++){
           for(int j=0;j<=w;j++){
                      printf("%d\t",k[i][j]);
           }
           printf("\n");
 }
 printf("\nMaximum profit:%d",k[n][w]);
} int
main(){
int w;
 printf("Enter the max size of sack:");
 scanf("%d",&w);
 int n;  printf("Enter the number of
items:");
 scanf("%d",&n);  int
val[n],wt[n];  printf("Enter the
items:\n");  for(int
i=0;i<n;i++){    printf("Item
%d: ",i+1);
           scanf("%d",&wt[i]);
 }
 printf("\nEnter the values:\n");
 for(int i=0;i<n;i++){
           printf("Value %d: ",i+1);
           scanf("%d",&val[i]);
 }
knapSack(w,wt,val,n);
return 0; }
```

Output:

Enter the max size of sack:5
Enter the number of items:4 Enter
the items:
Item 1: 2
Item 2: 3

Item 3: 4
Item 4: 5 Enter
the values:
Value 1: 3
Value 2: 4
Value 3: 5
Value 4: 6 Knapsack
matrix:

0	0	0	0	0	0
0	0	3	3	3	3
0	0	3	4	4	7
0	0	3	4	5	7
0	0	3	4	5	7

ASSIGNMENT 9

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

Program:

```
#include <stdio.h>
#define MAX_VERTICES 10

int a, b, u, v, n, i, j, ne = 1,
visited[MAX_VERTICES]={0},min,mincost=0,cost[MAX_VERTICES][MAX_VERTIC
```

```
ES];
void prim(int cost[MAX_VERTICES][MAX_VERTICES]) {
while (ne < n) {
        for (i = 0, min = 999; i < n; i++) {
            for (j = 0; 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\n Minimum cost = %d", mincost);
}

int main() {     printf("Enter no. of
vertices: ");     scanf("%d", &n);
    printf("Enter adjacency matrix\n");
    for (i = 0; i < n; i++) {         for (j
= 0; j < n; j++) {
printf("Mat[%d][%d]: ", i, j);
            scanf("%d", &cost[i][j]);
        }
}
    for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
if (cost[i][j] == 0)
cost[i][j] = 999;
        }     }
visited[0] = 1;
prim(cost);
return 0; }
```

Output:

Enter no. of vertices: 4

Enter adjacency matrix

Mat[1][1]: 0

```
Mat[1][2]: 1
Mat[1][3]: 5
Mat[1][4]: 2
Mat[2][1]: 1
Mat[2][2]: 0
Mat[2][3]: 999
Mat[2][4]: 999
Mat[3][1]: 5
Mat[3][2]: 999
Mat[3][3]: 0
Mat[3][4]: 3
Mat[4][1]: 2
Mat[4][2]: 999
Mat[4][3]: 3
Mat[4][4]: 0

Edge 1:(1 - 2) cost:1
Edge 2:(1 - 4) cost:2
Edge 3:(4 - 3) cost:3

Minimun cost=6
```

ASSIGNMENT 10

10. **Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm**.

Program:

```
//Kruskal's program
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 100
typedef struct Edge {
```

```
   int src, dest, weight;
} Edge;
typedef struct Graph {
   int V, E;
Edge* edge;
} Graph;
Graph* createGraph(int V, int E) {
   Graph* graph = (Graph*)malloc(sizeof(Graph));
graph->V = V;     graph->E = E;
   graph->edge = (Edge*)malloc(E * sizeof(Edge));
return graph;
} int find(int parent[], int i)
{     if (parent[i] == -1)
        return i;
   return find(parent, parent[i]);
}
void Union(int parent[], int x, int y) {
int xset = find(parent, x);     int yset =
find(parent, y);     parent[xset] =
yset;
} int compare(const void* a, const void* b)
{
   Edge* a_edge = (Edge*)a;
Edge* b_edge = (Edge*)b;
   return a_edge->weight - b_edge->weight;
}

void kruskalMST(Graph* graph) {
   int V = graph->V;
Edge result[V];
int e = 0;      int i =
0;
   qsort(graph->edge, graph->E, sizeof(graph->edge[0]), compare);
int* parent = (int*)malloc(V * sizeof(int));
```

```
    for (int v = 0; v < V; ++v)
        parent[v] = -1;
    while (e < V - 1 && i < graph->E) {
Edge next_edge = graph->edge[i++];
int x = find(parent, next_edge.src);
int y = find(parent, next_edge.dest);
if (x != y) {             result[e++] =
next_edge;
            Union(parent, x, y);
        }
    }
    printf("Edges in the minimum spanning tree:\n");
int minimumCost = 0;     for (i = 0; i < e; ++i) {
        printf("(%d, %d) -> %d\n", result[i].src, result[i].dest, result[i].weight);
minimumCost += result[i].weight;
    }
    printf("Minimum Cost Spanning Tree: %d\n", minimumCost);
} int main()
{     int V,
E;
    printf("Enter the number of vertices and edges: ");
    scanf("%d %d", &V, &E);
    Graph* graph = createGraph(V, E);
    printf("Enter the source, destination, and weight for each edge:\n");
for (int i = 0; i < E; ++i) {
        scanf("%d        %d       %d",    &graph->edge[i].src,   &graph->edge[i].dest,
>edge[i].weight);
    }
    kruskalMST(graph);
return 0;
}
```

Output:

```
Enter the number of vertices and edges: 4 5
Enter the source, destination, and weight for each edge:
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in the minimum spanning tree:
(2, 3) -> 4
(0, 3) -> 5
(0, 1) -> 10                                                        &graph-
```

Minimum Cost Spanning Tree: 19

ASSIGNMENT 11

11. Find Single source shortest path of a given undirected graph using Dijkstra's algorithm.

Program:

```
//dijkstra's algorithm #include<limits.h>
#include<stdbool.h> #include<stdio.h>
int N;
int minDistance(int dist[], bool visited[]) {
   int min = INT_MAX, min_index;
for (int i = 0; i < N; i++) {
      if (visited[i] == false && dist[i] <= min) {
min = dist[i];
         min_index = i;
      }
   }
   return min_index;
}
void printSolution(int dist[]) {
   printf("Vertex \t\t Distance from Source\n");
   for (int i = 0; i < N; i++) {
      printf("%d \t\t\t\t %d\n", i, dist[i]);
} }
void dijkstra(int graph[N][N], int src)
{    int dist[N];    bool visited[N];
for (int i = 0; i < N; i++) {        dist[i]
= INT_MAX;        visited[i] = false;
   }
   dist[src] = 0;
   for (int i = 0; i < N - 1; i++) {
int u = minDistance(dist, visited);
      visited[u] = true;
      for (int j = 0; j < N; j++) {
         if (!visited[j] && graph[u][j] && dist[u] != INT_MAX
&&              dist[u] + graph[u][j] < dist[j]) {              dist[j] =
dist[u] + graph[u][j];
         }
      }
   }
   printSolution(dist);
}
```

```
int main() {
 printf("Enter the number of vertices: ");
 scanf("%d",&N);
    int graph[N][N], src;
    printf("Enter the adjacency matrix for the graph (%d x %d):\n", N, N);
for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
                        printf("Mat[%d][%d]: ",i,j);
            scanf("%d", &graph[i][j]);
        }
    }
    printf("Enter the source node (0 to %d): ", N - 1);
scanf("%d", &src);     dijkstra(graph, src);     return
0; }
```

Output:

```
Enter the number of vertices: 4
Enter the adjacency matrix for the graph (4 x 4):
Mat[0][0]: 0
Mat[0][1]: 1
Mat[0][2]: 0
Mat[0][3]: 0
Mat[1][0]: 1
Mat[1][1]: 0
Mat[1][2]: 1
Mat[1][3]: 0
Mat[2][0]: 0
Mat[2][1]: 1
Mat[2][2]: 0
Mat[2][3]: 1
Mat[3][0]: 0
Mat[3][1]: 0
Mat[3][2]: 1
Mat[3][3]: 0
Enter the source node (0 to 3): 2
Vertex          Distance from Source
0                       2
1                       1
2                       0
3                       1
```