

LAB-1

ALGORITHMS LAB(18B17CI472)

Bubble Sort

ALGORITHM

This algorithm sorts the element of an array 'A' (having n elements) in the ascending (increasing) order. The pass counter is denoted by variable 'P' and the variable 'E' is used to count the number of exchanges performed on any pass. The last unsorted element is referred to by variable 'l'.

Step 1 Initialization

 set $l = n, P = 1$

Step 2 loop,

 Repeat step 3, 4 while ($P = n - 1$)

Step 3 set $E = 0$, R Initializing exchange variable.
 comparison, loop.

 Repeat for $i = 1, 1, \dots, l - 1$.

 if ($A[i] > A[i + 1]$) then

 set $A[i] = A[i + 1]$, R Exchanging values.

 set $E = E + 1$

Step 4 Finish, or reduce the size.

 if ($E = 0$) then

 exit

 else

 set $l = l - 1$.

CODE

```
/*BUBBLE SORT*/

#include <stdio.h>

#include <stdlib.h>

#include <direct.h>

#include <time.h>

int main()

{

    double start, _end, total;


    int i, n, j, temp, k;

    printf("\nEnter No of elements in the array:");

    scanf("%d", &n);

    start = clock();

    int *p = (int *)malloc(n * sizeof(int));

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

    {

        *(p + i) = rand();
```

```
}

//Bubble sort Implementation

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

{
    for (j = 0; j < n; j++)
    {
        if (*(p + j) > *(p + j + 1))
        {
            temp = *(p + j);
            *(p + j) = *(p + j + 1);
            *(p + j + 1) = temp;
        }
    }
}

_end = clock();

total = (_end - start) / CLOCKS_PER_SEC;

printf("\nThe time for the event was: %.3lf", total);

}
```

OUTPUT

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question1.c -o Question1 } ; if ($?) { .\Question1 }

Enter No of elements in the array:5000

PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question1.c -o Question1 } ; if ($?) { .\Question1 }

Enter No of elements in the array:10000

PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question1.c -o Question1 } ; if ($?) { .\Question1 }

Enter No of elements in the array:20000

The time for the event was: 1.431
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question1.c -o Question1 } ; if ($?) { .\Question1 }

Enter No of elements in the array:40000

The time for the event was: 5.977
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question1.c -o Question1 } ; if ($?) { .\Question1 }

Enter No of elements in the array:100000

The time for the event was: 39.046
PS C:\Users\hp\Desktop\Codes> █
```

GRAPH

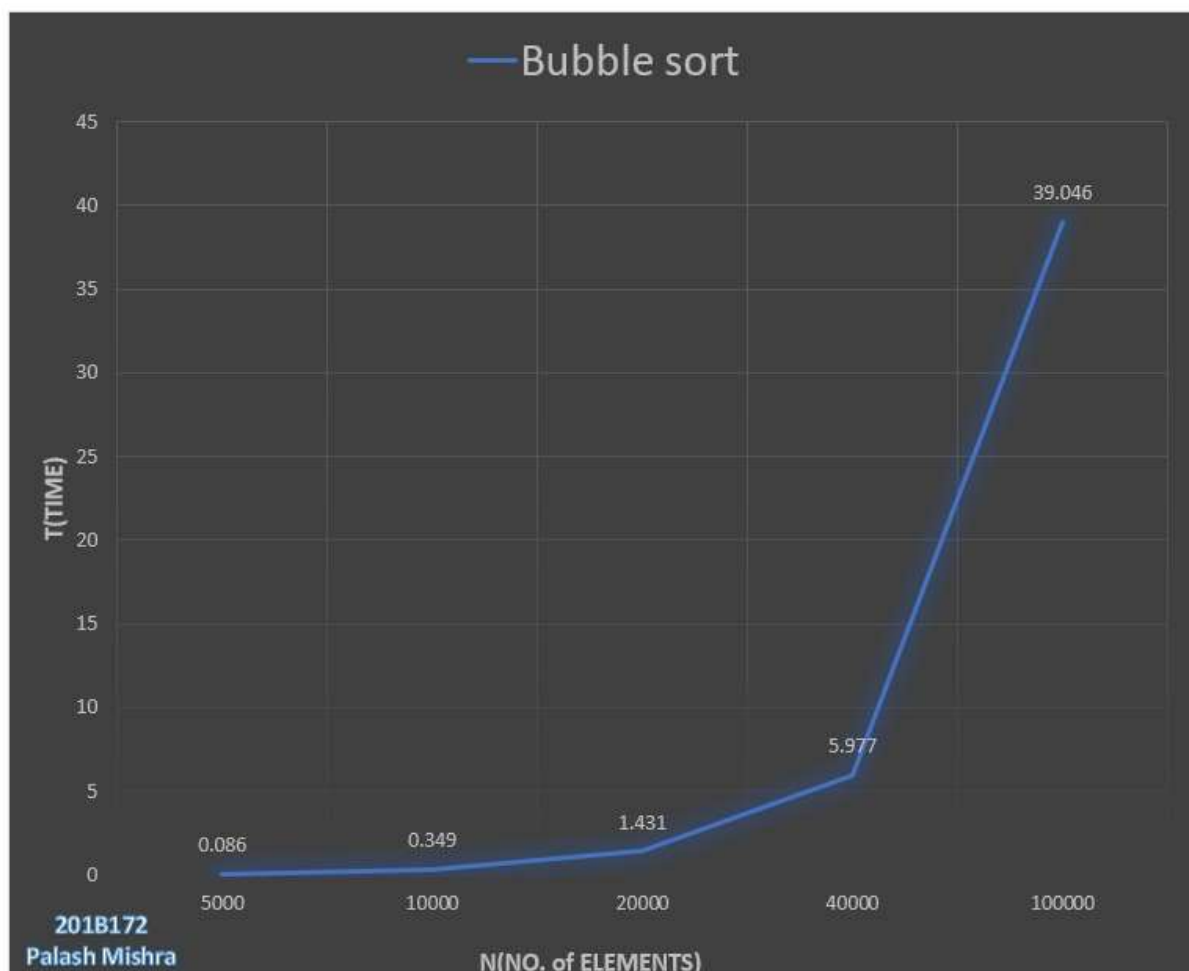


TABLE [Bubble Sort]

N	T
5000	0.086
10000	0.349
20000	1.431
40000	5.977
100000	39.046

LAB-2

APS_LAB

Merge Sort

ALGORITHM

MergeSort(arr[], l, r)

If $r > l$

1. Find the middle point to divide the array into two halves:

$$\text{middle } m = l + (r-l)/2$$

2. Call mergeSort for first half:

Call mergeSort(arr, l, m)

3. Call mergeSort for second half:

Call mergeSort(arr, m+1, r)

4. Merge the two halves sorted in step 2 and 3:

Call merge(arr, l, m, r)

CODE

```
#include <stdio.h>

#include <stdlib.h>

#include <direct.h>

#include <time.h>

void merge(int *array, int l, int m, int r)
{
    int i, j, k, nl, nr;

    nl = m - l + 1;

    nr = r - m;

    int larr[nl], rarr[nr];

    for (i = 0; i < nl; i++)

        larr[i] = array[l + i];

    for (j = 0; j < nr; j++)

        rarr[j] = array[m + 1 + j];

    i = 0;
```

```
j = 0;

k = l;

while (i < nl && j < nr)

{

    if (larr[i] <= rarr[j])

    {

        array[k] = larr[i];

        i++;

    }

    else

    {

        array[k] = rarr[j];

        j++;

    }

    k++;

}

while (i < nl)

{

    array[k] = larr[i];
```

```
        i++;

        k++;

    }

    while (j < nr)

    {

        array[k] = rarr[j];

        j++;

        k++;

    }

}

void mergeSort(int *array, int l, int r)

{

    int m;

    if (l < r)

    {

        int m = l + (r - l) / 2;

        mergeSort(array, l, m);

        mergeSort(array, m + 1, r);

        merge(array, l, m, r);
```

```
    }  
}  
  
int main()  
{  
    double start, _end, total;  
  
    int n;  
  
    printf("Enter the number of elements: ");  
  
    scanf("%d", &n);  
  
    start = clock();  
  
    int arr[n];  
  
    for (int i = 0; i < n; i++)  
    {  
        *(arr + i) = rand();  
    }  
  
    mergeSort(arr, 0, n - 1);  
  
    _end = clock();  
  
    total = (_end - start) / CLOCKS_PER_SEC;  
  
    printf("\nThe time for the event was: %.3lf", total);  
}
```

OUTPUT

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question2.c -o Question2 } ; if ($?) { .\Question2 }
Enter the number of elements: 5000

The time for the event was: 0.001
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question2.c -o Question2 } ; if ($?) { .\Question2 }
Enter the number of elements: 10000

The time for the event was: 0.003
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question2.c -o Question2 } ; if ($?) { .\Question2 }
Enter the number of elements: 20000

The time for the event was: 0.004
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question2.c -o Question2 } ; if ($?) { .\Question2 }
Enter the number of elements: 40000

The time for the event was: 0.009
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question2.c -o Question2 } ; if ($?) { .\Question2 }
Enter the number of elements: 100000

The time for the event was: 0.021
PS C:\Users\hp\Desktop\Codes> |
```

GRAPH

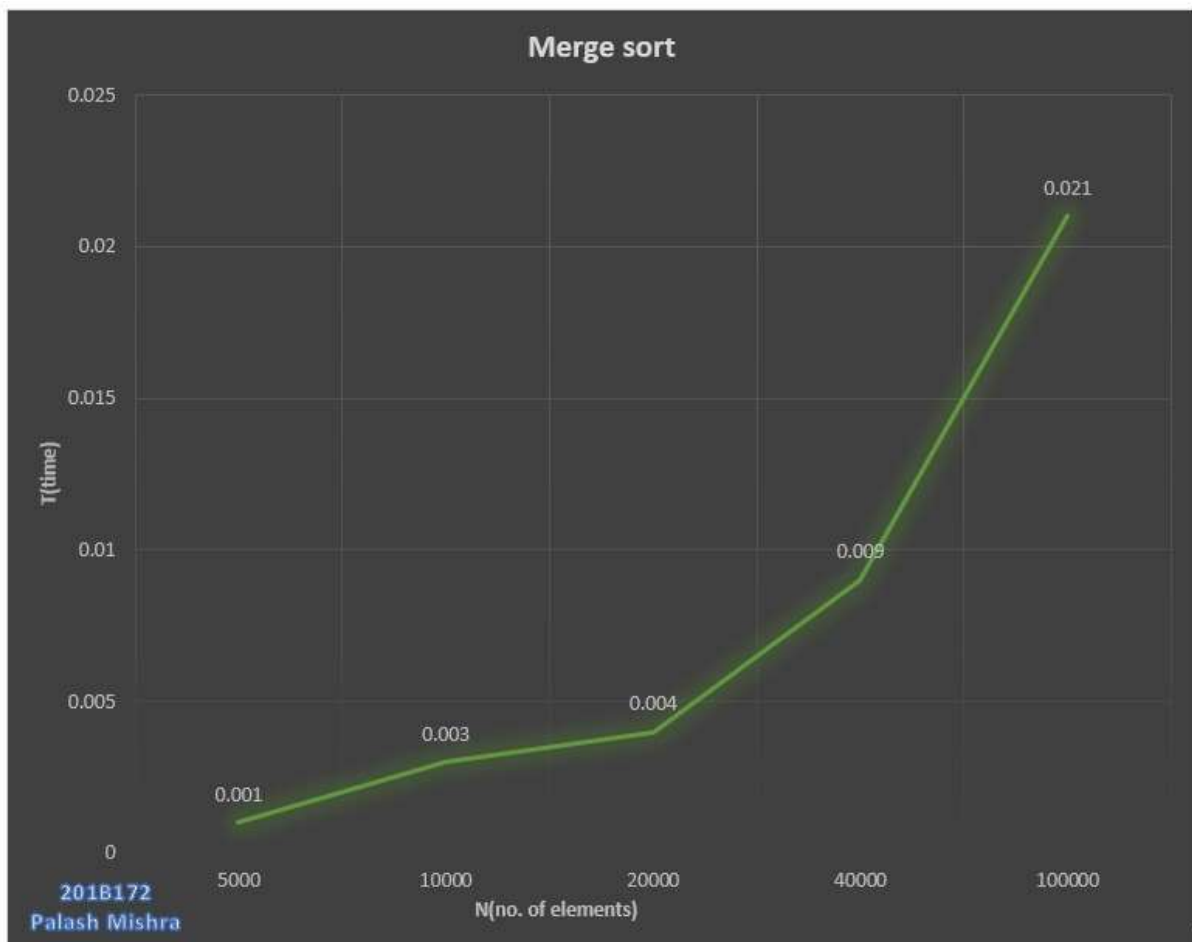


TABLE [Merge Sort]

N	T
5000	0.001
10000	0.003
20000	0.004
40000	0.009
100000	0.021

Insertion Sort

ALGORITHM

Step 1 – If it is the first element, it is already sorted. return 1;

Step 2 – Pick next element

Step 3 – Compare with all elements in the sorted sub-list

Step 4 – Shift all the elements in the sorted sub-list that is greater than the
value to be sorted

Step 5 – Insert the value

Step 6 – Repeat until list is sorted

CODE

```
#include <stdio.h>

#include <stdlib.h>

#include <direct.h>

#include <time.h>

int main()

{

    int i, j, N, temp;

    double start, _end, total;

    printf("\nEnter No of elements in the array:");

    scanf("%d", &N);

    start = clock();

    int arr[N];

    for (int i = 0; i < N; i++)

    {

        *(arr + i) = rand();

    }

}
```

```
// Implementation of insertion sort algorithm

for (i = 1; i < N; i++)
{
    temp = arr[i];

    j = i - 1;

    while ((temp < arr[j]) && (j >= 0))
    {
        arr[j + 1] = arr[j];

        j = j - 1;
    }

    arr[j + 1] = temp;
}

_end = clock();

total = (_end - start) / CLOCKS_PER_SEC;

printf("\nThe time for the event was: %.3lf", total);

return 0;
}
```

OUTPUT

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question3.c -o Question3 } ; if ($?) { .\Question3 }

Enter No of elements in the array:5000

The time for the event was: 0.015
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question3.c -o Question3 } ; if ($?) { .\Question3 }

Enter No of elements in the array:10000

The time for the event was: 0.063
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question3.c -o Question3 } ; if ($?) { .\Question3 }

Enter No of elements in the array:20000

The time for the event was: 0.256
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question3.c -o Question3 } ; if ($?) { .\Question3 }

Enter No of elements in the array:40000

The time for the event was: 1.035
PS C:\Users\hp\Desktop\Codes> cd "c:\Users\hp\Desktop\Codes\" ; if ($?) { gcc Question3.c -o Question3 } ; if ($?) { .\Question3 }

Enter No of elements in the array:100000

The time for the event was: 6.403
PS C:\Users\hp\Desktop\Codes> █
```

GRAPH

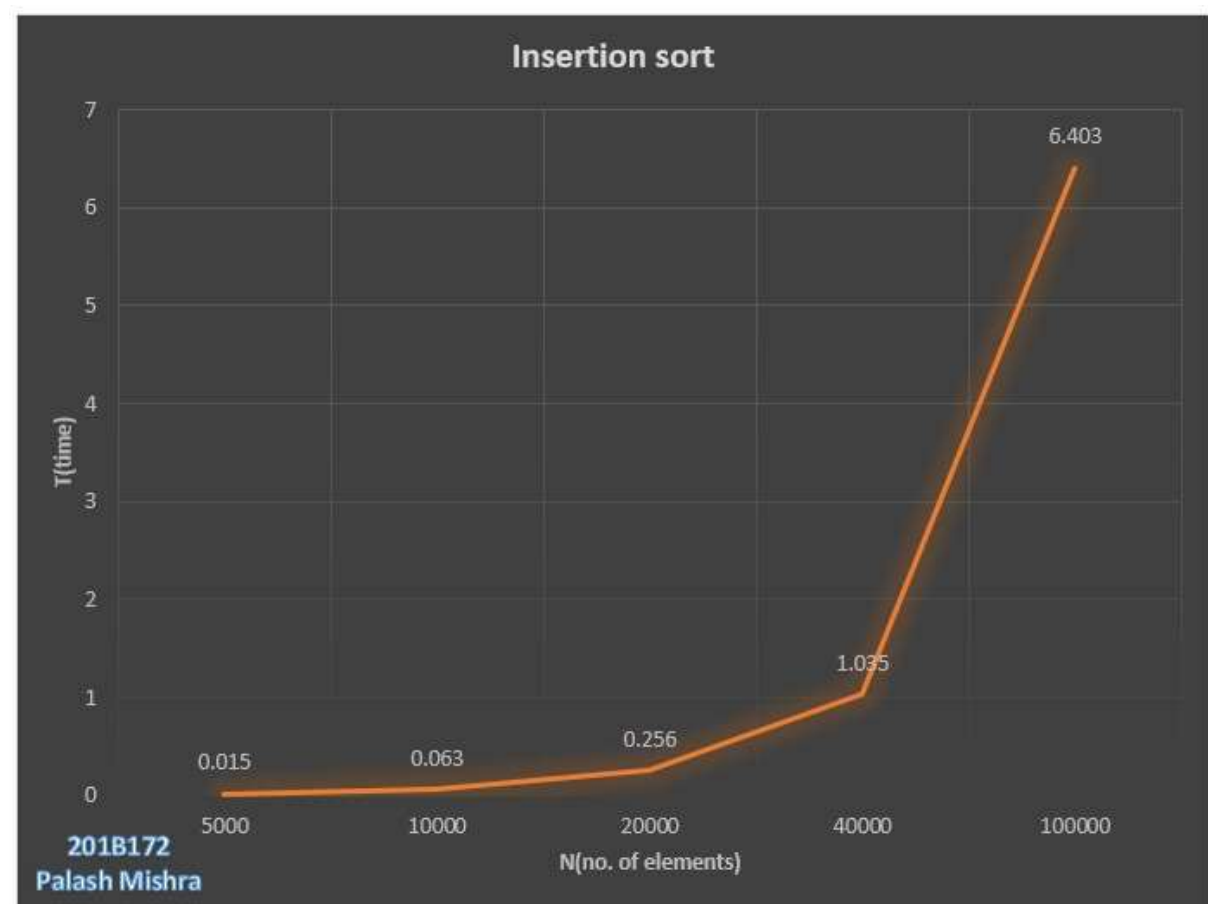
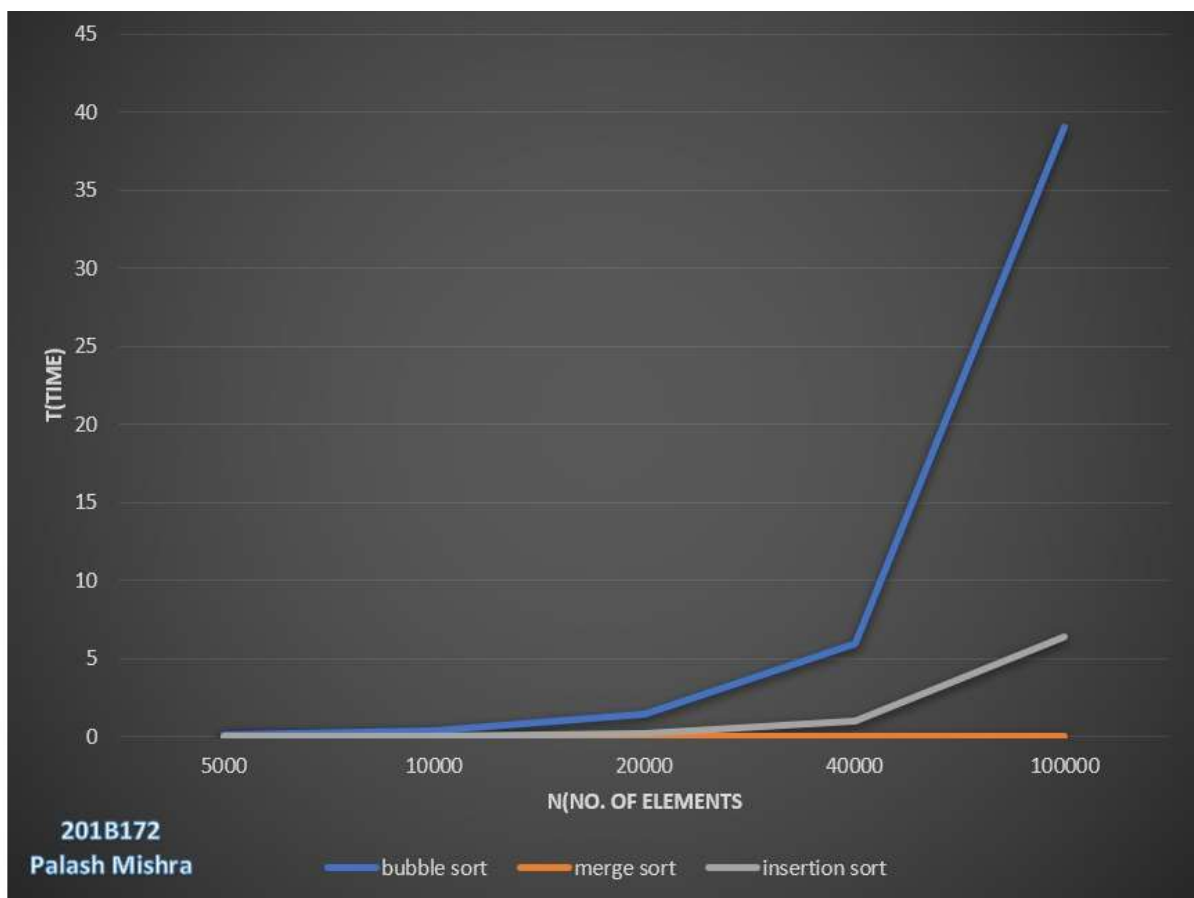


TABLE [Insertion Sort]

N	T
5000	0.015
10000	0.063
20000	0.256
40000	1.035
100000	6.403

Comparison Graph



LAB-4

APS_LAB

Problem 4.1:

Objective:-Given an array S of unsorted elements. Design an algorithm and implement that to find a pair x, y such that $x \neq y$ from S that minimizes $|x-y|$. The worst case running time of the algorithm should be $O(n \lg n)$.

Instruction: - Solve the above problem with the help of following example

Given Array is

**S = {4, 15, 8, 1, 19, 0, 12}, the output should be
0 and 1.**

Solution:

```
#include<bits/stdc++.h>
#include <stdlib.h>
#include <direct.h>
#include <time.h>
using namespace std;
int main()
{   vector<int>::iterator ip;
    double start, _end, total;
    start=clock();
    vector<int> arr={4,15,62,15,15,15,8,1,19,0,12,62};
    int n = sizeof(arr) / sizeof(arr[0]);
    sort(arr.begin(),arr.end()); //Sorting
    ip = unique(arr.begin(), arr.begin() + n); //making unique elements
    arr.resize(distance(arr.begin(), ip));
    cout<<arr[0]<<" "<<arr[1] <<endl;
    _end = clock();
    total = (_end - start) / CLOCKS_PER_SEC;
```

```
cout<<"Total Time : "<<total;
return 0;

}
```

Output:

Problem 4.2:

Objective:-Given two arrays A_1 , A_2 of size n and a number x , design an algorithm to find whether there exists a pair of elements one from A_1 and other from A_2 whose sum is equal to x . Also find the indices of those elements

Instruction: - Solve the above problem with the help of following example

Given Arrays are

$A_1=\{4,5,8,1,3,9,0,2\}$ and $A_2=\{2,3,35,32,12,9,2\}$ and $x = 41$

The output should be yes with $i_1=5$ and $i_2=3$.

For $x=25$, the output should be no.

Solution:

```
#include<bits/stdc++.h>
#include <stdlib.h>
#include <direct.h>
#include <time.h>
using namespace std;
int main()
{
    double start, _end, total;
```

```

    start=clock();
int A1[]={4,5,8,1,3,9,0,2},A2[]={2,3,35,32,12,9,2,5},x=41,i,j,flag=0,I,J;
int n=sizeof(A1)/sizeof(A1[0]);
cout<<"Size : "<<n<<endl;
    for( i=0;i<n;i++)
    { for( j=0;j<n;j++)
        {
            if((A1[i]+A2[j])==x)
            {
                I=i;
                J=j;
                flag=1;
                break;
            } } }
    if(flag==1)
        cout<<I<<" "<<J;
    else
        cout<<"No";

    _end = clock();
total = (_end - start) / CLOCKS_PER_SEC;
cout<<endl<<"Total Time : "<<total;
return 0;
}

```

Output:

```

C:\Users\201B172\Documents\LAB4_Que2.exe
Size : 8
5 3
Total Time : 0.001
Process returned 0 (0x0) execution time : 0.141 s
Press any key to continue.

```

Heap Sort

```
#include <iostream>
#include <stdlib.h>
#include <direct.h>
#include <time.h>
using namespace std;
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2 * i + 1; // left = 2*i + 1
    int r = 2 * i + 2; // right = 2*i + 2
    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;
    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}
```

```
}  
}  
void printArray(int arr[], int n)  
{  
    for (int i = 0; i < n; ++i)  
        cout << arr[i] << " ";  
    cout << "\n";  
}  
int main()  
{  
    double start, _end, total;  
    start=clock();  
    int arr[] = { 12, 11, 13, 5, 6, 7 };  
    int n = sizeof(arr) / sizeof(arr[0]);  
  
    heapSort(arr, n);  
    cout << "Sorted array is \n";  
    printArray(arr, n);  
    _end = clock();  
    total = (_end - start) / CLOCKS_PER_SEC;  
    cout<<endl<<"Total Time : "<<total;  
  
}
```

```
C:\Users\201B172\Documents\HEapsort.exe  
Sorted array is  
5 6 7 11 12 13  
  
Total Time : 0.002  
Process returned 0 (0x0)   execution time : 0.041 s  
Press any key to continue.
```

BST

//C++ Program for BST -

Traverse, Insertion, Searching.

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int key;
    struct node *left, *right;
};
struct node *newNode(int item){
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}
void traversetree(struct node *root){
    if (root != NULL){
        traversetree(root->left);
        printf("%d \t", root->key);
        traversetree(root->right);
    }
}
struct node* search(struct node* root, int key){
    if (root == NULL || root->key == key)
        return root;
    if (root->key < key)
        return search(root->right, key);
    return search(root->left, key);
}
struct node* insert(struct node* node, int key){
    if (node == NULL) return newNode(key);
    if (key < node->key)
```

```
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    return node;
}

int main(){
    struct node *root = NULL;
    root = insert(root, 23);
    insert(root, 15);
    insert(root, 12);
    insert(root, 17);
    insert(root, 32);
    insert(root, 29);
    insert(root, 45);
    printf("The tree is :\n");
    traversetree(root);
    printf("\nSearching for 12 in this tree ");
    if(search(root , 12))
        printf("\nelement found");
    else
        printf("\nelement not found");
    return 0;
}
```

```
The tree is :
7 10 16 22 23 29 36 45 62 95
Searching for 12 in this tree
element found
Total Time : 0.005
PS D:\Desktop_Items\DSA\BST>
```

//C++ Program for BST - Deletion.

```
#include <bits/stdc++.h>
using namespace std;

struct node
{
    int key;
    struct node *left, *right;
};

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

void traversetree(struct node *root)
{
    if (root != NULL)
    {
        traversetree(root->left);
        cout << root->key << " ";
        traversetree(root->right);
    }
}
```

```
}  
// inorder traversal of BST  
void inorder(struct node *root)  
{  
    if (root != NULL)  
    {  
        inorder(root->left);  
        cout << root->key << " ";  
        inorder(root->right);  
    }  
}  
struct node *insert(struct node *node, int key)  
{  
  
    if (node == NULL)  
        return newNode(key);  
    if (key < node->key)  
        node->left = insert(node->left, key);  
    else  
        node->right = insert(node->right, key);  
    return node;  
}  
struct node *minValueNode(struct node *node)  
{  
    struct node *current = node;  
    while (current && current->left != NULL)  
        current = current->left;
```

```
    return current;
}
struct node *deleteNode(struct node *root, int key)
{
    if (root == NULL)
        return root;
    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    else
    {
        if (root->left == NULL and root->right == NULL)
            return NULL;
        else if (root->left == NULL)
        {
            struct node *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            struct node *temp = root->left;
            free(root);
            return temp;
        }
    }
}
```

```
    }
    struct node *temp = minValueNode(root->right);
    root->key = temp->key;
    root->right = deleteNode(root->right, temp->key);
}
return root;
}
int main()
{
    double start, _end, total;
    start = clock();

    struct node *root = NULL;
    root = insert(root, 23);
    insert(root, 45);
    insert(root, 62);
    insert(root, 7);
    insert(root, 22);
    insert(root, 29);
    insert(root, 16);
    insert(root, 36);
    insert(root, 10);
    insert(root, 95);

    cout << "Your BST : ";
    traversetree(root);
    cout << "\nInorder traversal of the given tree \n";
```

```
    inorder(root);
    cout << "\nDelete 29\n";
    root = deleteNode(root, 29);
    cout << "Inorder traversal of the modified tree \n";
    inorder(root);

    _end = clock();
    total = (_end - start) / CLOCKS_PER_SEC;
    cout << endl
          << "Total Time : " << total;
    return 0;
}
```

```
Your BST : 7 10 16 22 23 29 36 45 62 95
Inorder traversal of the given tree
7 10 16 22 23 29 36 45 62 95
Delete 29
Inorder traversal of the modified tree
7 10 16 22 23 36 45 62 95
Total Time : 0.012
```

RB-Tree

1. Left Rotation:

Algorithm:

1. $y \leftarrow \text{right}[x]$
2. $\text{right}[x] \leftarrow \text{left}[y]$
3. if $\text{left}[y] \neq \text{NIL}$
4. then $p[\text{left}[y]] \leftarrow x$
5. $p[y] \leftarrow p[x]$
6. if $p[x] = \text{NIL}$
7. then $\text{root}[T] \leftarrow y$
8. else if $x = \text{left}[p[x]]$
9. then $\text{left}[p[x]] \leftarrow y$
10. else $\text{right}[p[x]] \leftarrow y$
11. $\text{left}[y] \leftarrow x$
12. $p[x] \leftarrow y$

2. Right Rotation:

Algorithm:

1. $y \leftarrow \text{left}[x]$
2. $\text{left}[x] \leftarrow \text{right}[y]$
3. if $\text{left}[y] \neq \text{NIL}$

-
4. then $p[\text{left}[y]] \leftarrow x$
 5. $p[y] \leftarrow p[x]$
 6. if $p[x] = \text{NIL}$
 7. then $\text{root}[T] \leftarrow y$
 8. else if $x = \text{left}[p[x]]$
 9. then $\text{left}[p[x]] \leftarrow y$
 10. else $\text{right}[p[x]] \leftarrow y$
 11. $\text{right}[y] \leftarrow x$
 12. $p[x] \leftarrow y$

3. Insert Fixup:

Algorithm:

1. while $\text{color}[p[z]] = \text{RED}$
2. do if $p[z] = \text{left}[p[p[z]]]$
3. then $y \leftarrow \text{right}[p[p[z]]]$
4. if $\text{color}[y] = \text{RED}$
5. then Case1
6. else if $z = \text{right}[p[z]]$
7. then Case2
8. Case3
9. else (same as then clause with "right" and "left" exchanged)
10. $\text{color}[\text{root}[T]] \leftarrow \text{BLACK}$

4. Insert:

Algorithm:

1. $y \leftarrow \text{NIL}$
2. $x \leftarrow \text{root}[T]$
3. while $x \neq \text{NIL}$
4. do $y \leftarrow x$
5. if $\text{key}[z] < \text{key}[x]$
6. then $x \leftarrow \text{left}[x]$
7. else $x \leftarrow \text{right}[x]$
8. $p[z] \leftarrow y$
9. if $y = \text{NIL}$
10. then $\text{root}[T] \leftarrow z$
11. else if $\text{key}[z] < \text{key}[y]$
12. then $\text{left}[y] \leftarrow z$
13. else $\text{right}[y] \leftarrow z$
14. $\text{left}[z] \leftarrow \text{NIL}$
15. $\text{right}[z] \leftarrow \text{NIL}$
16. $\text{color}[z] \leftarrow \text{RED}$
17. $\text{RB-INSERT-FIXUP}(T, z)$

CODE:

```
#include <bits/stdc++.h>

using namespace std;
```

```
enum Color {RED, BLACK};

struct Node{

    int data;

    bool color;

    Node *left, *right, *parent;

    Node(int data){

        this->data = data;

        left = NULL, right = NULL, parent = NULL;

        color = RED;

    }

};

void left_rotation(Node *&root, Node *&x){

    Node *y = x->right;

    x->right = y->left; //y's left subtree becomes x's right subtree

    if (x->right != NULL){

        x->right->parent = x; //Set the parent relation from left[y] to x

    }

    y->parent = x->parent; //The parent of x becomes the parent of y

    if (x->parent == NULL){

        root = y;

    }

    else if (x == x->parent->left){
```

```
x->parent->left = y;
}
else x->parent->right = y;
y->left = x; //Put x on y's left
x->parent = y; //y becomes x's parent
}

void right_rotation(Node *&root, Node *&x){
    Node *y = x->left;
    x->left = y->right; //y's right subtree becomes x's left subtree
    if (x->left != NULL){
        x->left->parent = x; //Set the parent relation from right[y] to x
    }
    y->parent = x->parent; //The parent of x becomes the parent of y
    if (x->parent == NULL){
        root = y;
    }
    else if (x == x->parent->left){
        x->parent->left = y;
    }
    else x->parent->right = y;
    y->right = x; //Put x on y's right
    x->parent = y; //y becomes x's parent
```

```

}

void case_1(Node *&parent, Node *&grandparent, Node *&y, Node *&z){
    //Case:1 The uncle of x is also red Only Recoloring required
    grandparent->color = RED;
    parent->color = BLACK;
    y->color = BLACK;
    z = grandparent;
}

void insert_fixup(Node *&root, Node *&z){
    Node *parent = NULL , *y = NULL , *grandparent = NULL;
    while ((z != root) && (z->color != BLACK) && (z->parent->color == RED)){
        parent = z->parent;
        grandparent = z->parent->parent;
        //Case:A Parent of x is left child of Grandparent of x
        if (parent == grandparent->left){
            y = grandparent->right;
            if (y != NULL && y->color == RED){
                case_1(parent, grandparent, y, z);
            }
        }
        else{
            //Case:2 x is right child of its parent Left rotation required
            if (z == parent->right){

```

```
left_rotation(root, parent);

z = parent;

parent = z->parent;

}

// Case:3 x is left child of its parent Right rotation required

right_roation(root, grandparent);

swap(parent->color,grandparent->color);

z = parent;

}

}

//Case:B Parent of x is right child of Grandparent of x

else{

Node *y = grandparent->left;

if ((y != NULL) && (y->color ==RED)){

case_1(parent,grandparent,y,z);

}

else{

//Case:2 x is left child of its parent Right rotation required

if (z == parent->left){

right_roation(root, parent);

z = parent;

parent = z->parent;
```

```
}
```

```
//Case:3 x is right child of its parent Left rotation required
```

```
left_rotation(root, grandparent);
```

```
swap(parent->color, grandparent->color);
```

```
z = parent;
```

```
}
```

```
}
```

```
}
```

```
root->color = BLACK;
```

```
}
```

```
void insert(Node *&root,int x){ //insertion of node in RB tree
```

```
Node *newNode = new Node(x);
```

```
Node *temp = root;
```

```
Node *parent = root;
```

```
if (root == NULL){
```

```
root = newNode;
```

```
return;
```

```
}
```

```
while (temp != NULL){
```

```
parent = temp;
```

```
if (temp->data > x)
```

```
temp = temp->left;
```

```
else
temp = temp->right;
}
if (parent->data > x)
parent->left = newNode;
if (parent->data < x)
parent->right = newNode;
insert_fixup(root, newNode); //Fixup any violation of property of RB
tree
}
void inorder(Node *root){
if (root == NULL) return;
inorder(root->left);
cout << root->data << " ";
inorder(root->right);
}
int main(){
int n,x;
cin>>n;
Node *root =NULL;
for(int i=0;i<n;i++){
cin>>x;
```

```
insert(root,x);  
}  
cout<<"RB Tree is : ";  
inorder(root);  
return 0;  
}
```

Prims Algo:

```
//Code for Prims Algorithm  
#include <bits/stdc++.h>  
using namespace std;  
#define V 5  
  
//function- find min vertex which is not in MST  
int Minimum_key(int key[], bool mstSet[])  
{  
    int min = INT_MAX, min_index;//infinity  
  
    for (int v = 0; v < V; v++)  
        if (mstSet[v] == false && key[v] < min)  
            min = key[v], min_index = v;  
  
    return min_index;
```

```
}

//Print MST function
void printMST(int parent[], int graph[V][V])
{
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]] << " \n";
}

void primMST(int graph[V][V])
{
    // Array to store constructed MST
    int parent[V];

    // Key values used to pick minimum weight edge in cut
    int key[V];

    // To represent set of vertices included in MST
    bool mstSet[V];

    // set all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    //picking up the first vertex-0
    key[0] = 0;
```

```
parent[0] = -1; // First node is always root of MST
for (int count = 0; count < V - 1; count++)
{
    //find minkey
    int u = Minimum_key(key, mstSet);
    //add it to MST set
    mstSet[u] = true;

    // Update key value and parent - adjacent vertices
    for (int v = 0; v < V; v++)
        if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
            parent[v] = u, key[v] = graph[u][v];
}

// print MST
printMST(parent, graph);
}

// Driver code
int main()
{
    int graph[V][V] = {{0, 3, 0, 7, 0},
```

```
{3, 0, 4, 9, 6},  
{0, 4, 0, 0, 8},  
{7, 9, 0, 0, 10},  
{0, 6, 9, 10, 0}};
```

```
// Print the solution
```

```
primMST(graph);
```

```
return 0;
```

```
}
```

```
PS D:\SEM4\APS_LAB> cd "d:\SEM4\APS_LAB\" ;  
Edge      Weight  
0 - 1      3  
1 - 2      4  
0 - 3      7  
1 - 4      6  
PS D:\SEM4\APS_LAB>
```

Kruskals Algo:

```
//Code for Kruskals Algo:
```

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

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

```
#include <string.h>

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

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

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

struct subset
{
    int parent;
    int rank;
```

```
};  
  
int find(struct subset subsets[], int i)  
{  
    if (subsets[i].parent != i)  
        subsets[i].parent = find(subsets, subsets[i].parent);  
    return subsets[i].parent;  
}  
  
void Union(struct subset subsets[], int x, int y)  
{  
    int xroot = find(subsets, x);  
    int yroot = find(subsets, y);  
    if (subsets[xroot].rank < subsets[yroot].rank)  
        subsets[xroot].parent = yroot;  
    else if (subsets[xroot].rank > subsets[yroot].rank)  
        subsets[yroot].parent = xroot;  
    else  
    {  
        subsets[yroot].parent = xroot;  
        subsets[xroot].rank++;  
    }  
}  
  
int Comparator(const void *a, const void *b)
```

```
{
    struct Edge *a1 = (struct Edge *)a;
    struct Edge *b1 = (struct Edge *)b;
    return a1->weight > b1->weight;
}

void KruskalMST(struct Graph *graph)
{
    int V = graph->V;
    struct Edge
        result[V];
    int e = 0;
    int i = 0;
    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), Comparator);
    struct subset *subsets = (struct subset *)malloc(V * sizeof(struct
subset));
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    while (e < V - 1 && i < graph->E)
    {
```

```
    struct Edge next_edge = graph->edge[i++];
    int x = find(subsets, next_edge.src);
    int y = find(subsets, next_edge.dest);
    if (x != y)
    {
        result[e++] = next_edge;
        Union(subsets, x, y);
    }
}
printf("Following are the edges in the constructed MST\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", minimumCost);
return;
}

int main()
{
```

```
int V = 24;
int E = 25;
struct Graph *graph = createGraph(V, E);
graph->edge[0].src = 20;
graph->edge[0].dest = 21;
graph->edge[0].weight = 30;
graph->edge[1].src = 20;
graph->edge[1].dest = 22;
graph->edge[1].weight = 26;
graph->edge[2].src = 20;
graph->edge[2].dest = 23;
graph->edge[2].weight = 25;
graph->edge[3].src = 21;
graph->edge[3].dest = 23;
graph->edge[3].weight = 35;
graph->edge[4].src = 22;
graph->edge[4].dest = 23;
graph->edge[4].weight = 24;
KruskalMST(graph);
return 0;
}
```

```
Following are the edges in the constructed MST
22 -- 23 == 24
20 -- 23 == 25
20 -- 21 == 30
Minimum Cost Spanning tree : 79
```

Bellman Ford Algo:

// Code for BellmanFord Algo:

```
struct Edge
```

```
{
```

```
    int src, dest, weight;
```

```
};
```

```
struct Graph
```

```
{
```

```
    int V, E;
```

```
    struct Edge *edge;
```

```
};
```

```
struct Graph *createGraph(int V, int E)
```

```
{
```

```
    struct Graph *graph = new Graph;
```

```
    graph->V = V;
```

```
graph->E = E;
graph->edge = new Edge[E];
return graph;
}

// function -> print
void printArr(int dist[], int n)
{
    printf("Vertex Distance from Source\n");
    for (int i = 0; i < n; ++i)
        printf("%d \t\t %d\n", i, dist[i]);
}

void BellmanFord(struct Graph *graph, int src)
{
    int V = graph->V;
    int E = graph->E;
    int dist[V];

    // Step 1: set distances from src to all other vertices as INFINITE
    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX;
    dist[src] = 0;
```

```
// Step 2: Relax all edges |V| - 1 times.
for (int i = 1; i <= V - 1; i++)
{
    for (int j = 0; j < E; j++)
    {
        int u = graph->edge[j].src;
        int v = graph->edge[j].dest;
        int weight = graph->edge[j].weight;
        if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
            dist[v] = dist[u] + weight;
    }
}

// Step 3: check for negative-weight cycles.
for (int i = 0; i < E; i++)
{
    int u = graph->edge[i].src;
    int v = graph->edge[i].dest;
    int weight = graph->edge[i].weight;
    if (dist[u] != INT_MAX && dist[u] + weight < dist[v])
    {
        printf("Graph contains negative weight cycle");
    }
}
```

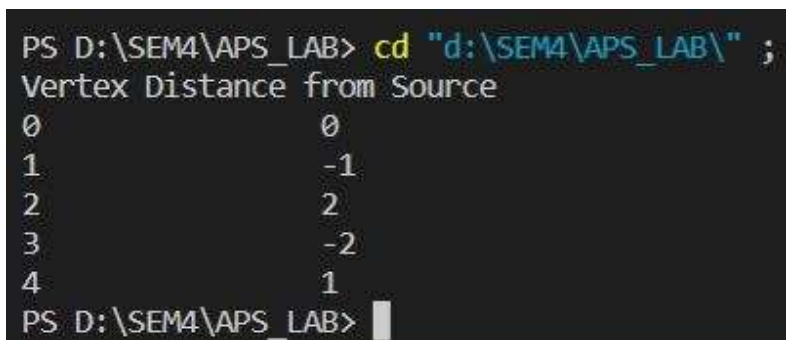
```
        return; // If negative cycle is detected -> return
    }
}

printArr(dist, V);

return;
}
```

```
int main()
{
    int V = 5; // vertices
    int E = 8; // edges
    struct Graph *graph = createGraph(V, E);
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = -1;
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 4;
    graph->edge[2].src = 1;
    graph->edge[2].dest = 2;
    graph->edge[2].weight = 3;
    graph->edge[3].src = 1;
```

```
graph->edge[3].dest = 3;
graph->edge[3].weight = 2;
graph->edge[4].src = 1;
graph->edge[4].dest = 4;
graph->edge[4].weight = 2;
graph->edge[5].src = 3;
graph->edge[5].dest = 2;
graph->edge[5].weight = 5;
graph->edge[6].src = 3;
graph->edge[6].dest = 1;
graph->edge[6].weight = 1;
graph->edge[7].src = 4;
graph->edge[7].dest = 3;
graph->edge[7].weight = -3;
BellmanFord(graph, 0);
return 0;
}
```



```
PS D:\SEM4\APS_LAB> cd "d:\SEM4\APS_LAB\" ;
Vertex Distance from Source
0          0
1         -1
2          2
3         -2
4          1
PS D:\SEM4\APS_LAB> 
```

Dijkstra Algo:

//Code for Dijkstra Algo:

#include <limits.h>

#include <stdio.h>

#define V 9

//function- find min vertex which is not in in shortest path tree

int Minimun_Distance(int dist[], bool sptSet[])

{

int min = INT_MAX, min_index;//Infinity

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min_index = v;

return min_index;

}

//function to print

int printSol(int dist[], int n)

```
{  
    printf("Vertex Distance from Source\n");  
    for (int i = 0; i < V; i++)  
        printf("%d \t\t %d\n", i, dist[i]);  
}  
  
void dijkstra(int graph[V][V], int src)  
{  
    int dist[V]; // The output array  
    bool sSet[V]; // sSet[i] will be true if vertex i is included in shortest  
  
    // Set all distances as INFINITE and sSet[] as false  
    for (int i = 0; i < V; i++)  
        dist[i] = INT_MAX, sSet[i] = false;  
  
    // Distance of source vertex from itself is always 0  
    dist[src] = 0;  
  
    // Find shortest  
    for (int count = 0; count < V - 1; count++)  
    {  
        int u = Minimun_Distance(dist, sSet);  
        sSet[u] = true;
```

```
    for (int v = 0; v < V; v++)
        if (!sSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] +
graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
    }
    printSol(dist, V);
}
int main()
{
    int graph[V][V] = {{0, 5, 0, 0, 0, 0, 0, 8, 0},
        {4, 0, 9, 0, 0, 0, 0, 11, 0},
        {0, 8, 0, 7, 0, 4, 0, 0, 2},
        {0, 0, 7, 0, 9, 14, 0, 0, 0},
        {0, 0, 0, 9, 0, 10, 0, 0, 0},
        {0, 0, 4, 14, 10, 0, 2, 0, 0},
        {0, 0, 0, 0, 0, 2, 0, 1, 6},
        {8, 11, 0, 0, 0, 0, 1, 0, 7},
        {0, 0, 2, 0, 0, 0, 6, 7, 0}};

    dijkstra(graph, 0);

    return 0;
```

}

```
PS D:\SEM4\APS_LAB> cd "d:\SEM4\APS_LAB\" ;  
Vertex Distance from Source  
0          0  
1          5  
2         14  
3         21  
4         21  
5         11  
6          9  
7          8  
8         15  
PS D:\SEM4\APS_LAB> |
```