```
//Linear search
#include <iostream>
using namespace std;

int main() {
    int arr[10], i, num, index;

    cout << "Enter 10 numbers: ";
    for(i = 0; i < 10; i++) {
        cin >> arr[i];
    }

    cout << "\nEnter a number to search: ";
    cin >> num;

    for(i = 0; i < 10; i++) {
        if(arr[i] == num) {
            index = i;
            break;
        }
    }

    if(index != -1) {
        cout << "\nFound at index no: " << index+1;
    } else {
        cout << "\nNumber not found.";
    }

    return 0;
}
```

```
//Quick sort
#include <iostream>

// Function to partition the array into two sub-arrays
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Choose the pivot as the last element
    int i = (low - 1); // Index of the smaller element

    for (int j = low; j <= high - 1; j++) {
        // If the current element is smaller or equal to the pivot
        if (arr[j] <= pivot) {
            i++; // Increment the index of the smaller element
            std::swap(arr[i], arr[j]);
        }
    }

    std::swap(arr[i + 1], arr[high]);
    return (i + 1);
}

// Function to perform Quick Sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // Partition the array into two sub-arrays and get the pivot
        int pivot = partition(arr, low, high);

        // Recursively sort the sub-arrays
        quickSort(arr, low, pivot - 1);
        quickSort(arr, pivot + 1, high);
    }
}

int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Original array: ";
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    quickSort(arr, 0, n - 1);
    std::cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

```
//Insertion sort
#include <iostream>

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move elements of arr[0..i-1] that are greater than key
        // to one position ahead of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = key;
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6};
    int n = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Original array: ";
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    insertionSort(arr, n);

    std::cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

```
//Minimum spanning tree
#include<stdio.h>
int main()
{
    int cost[10][10],visited[10]={0},i,j,n,no_e=1,min,a,b,min_cost=0;
    printf("Enter number of nodes ");
    scanf("%d",&n);
    printf("Enter cost in form of adjacency matrix\n");
    //input graph
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            // cost is 0 then initialize it by maximum value
            if(cost[i][j]==0)
              cost[i][j]=1000;
        }
    }

    // logic for finding minimum cost spanning tree
    visited[1]=1; // visited first node
    while(no_e<n)
    {
        min=1000;
        // in each cycle find minimum cost
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                if(cost[i][j]<min)
                {
                    if(visited[i]!=0)
                    {
                        min=cost[i][j];
                        a=i;
                        b=j;
                    }
                }
            }
        }
        //if node is not visited
        if(visited[b]==0)
        {
            printf("\n%d to %d  cost=%d",a,b,min);
            min_cost=min_cost+min;
            no_e++;
        }
        visited[b]=1;
        // initialize with maximum value you can also use any other value
        cost[a][b]=cost[b][a]=1000;
    }
    printf("\nminimum weight is %d",min_cost);
    return 0;
}
```

```
//Bubble sort
#include <iostream>

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        // Flag to optimize the algorithm by avoiding unnecessary passes
        bool swapped = false;

        // Last i elements are already in place, so no need to compare them
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;

                swapped = true;
            }
        }
        // If no two elements were swapped in the inner loop, the array is already sorted
        if (!swapped) {
            break;
        }
    }
}
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    std::cout << "Original array: ";
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    bubbleSort(arr, n);

    std::cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

```
//Binary search tree
#include <stdio.h>
int main()
{
int i, low, high, mid, n, key, array[100];
printf("Enter number of elementsn");
scanf("%d",&n);
printf("Enter %d integersn", n);
for(i = 0; i < n; i++)
scanf("%d",&array[i]);
printf("Enter value to findn");
scanf("%d", &key);
low = 0;
high = n - 1;
mid = (low+high)/2;
while (low <= high) {
if(array[mid] < key)
low = mid + 1;
else if (array[mid] == key) {
printf("%d found at location %d.n", key, mid+1);
break;
}
else
high = mid - 1;
mid = (low + high)/2;
}
if(low > high)
printf("Not found! %d isn't present in the list.n", key);
return 0;
}
```