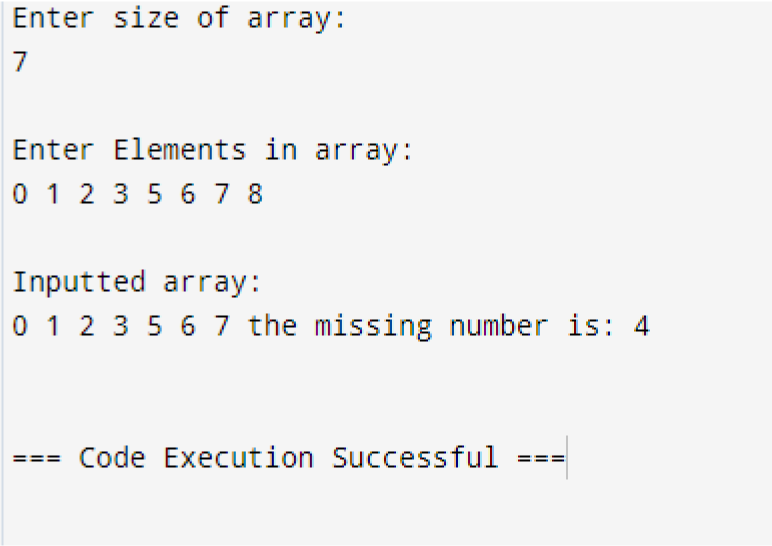
Week 3B

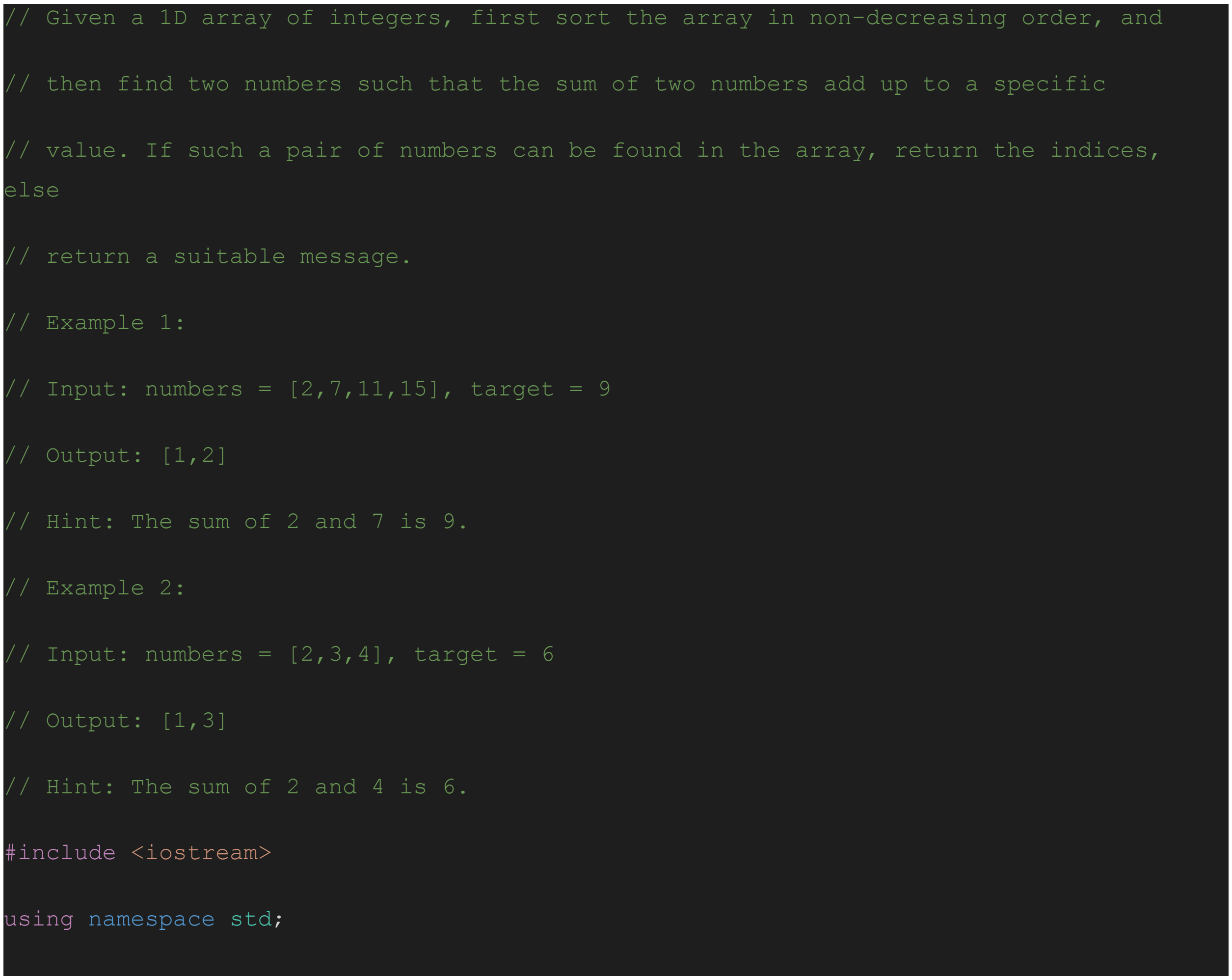
1)

```
#include <iostream>

using namespace std;

void sort(int arr[], int size) {

    for (int i = 0; i < size - 1; ++i) {

        for (int j = 0; j < size - i - 1; ++j) {

            if (arr[j] > arr[j + 1]) {

                int temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

}


int findmissingnumber(int arr[], int size) {

    sort(arr, size);


    for (int i = 0; i < size; ++i) {

        if (arr[i] != i) {

            return i;

        }
    }

    return size;
}

int main() {
  int n,a;
  cout<<"Enter size of array:\n";
  cin>>n;
  int*arr=new int[n];
  cout<<"\nEnter Elements in array:\n";
  for(int i=0;i<n;i++){
      cin>>a;
      arr[i]=a;
  }
  cout<<"\nInputted array:\n";
    for(int i=0;i<n;i++){
      cout<<arr[i]<<" ";

  }

    int missingnumber = findmissingnumber(arr, n);
    cout << "the missing number is: " << missingnumber << endl;
    return 0;
}
```



```
Enter size of array:
7

Enter Elements in array:
0 1 2 3 5 6 7 8

Inputted array:
0 1 2 3 5 6 7 the missing number is: 4


=== Code Execution Successful ===
```

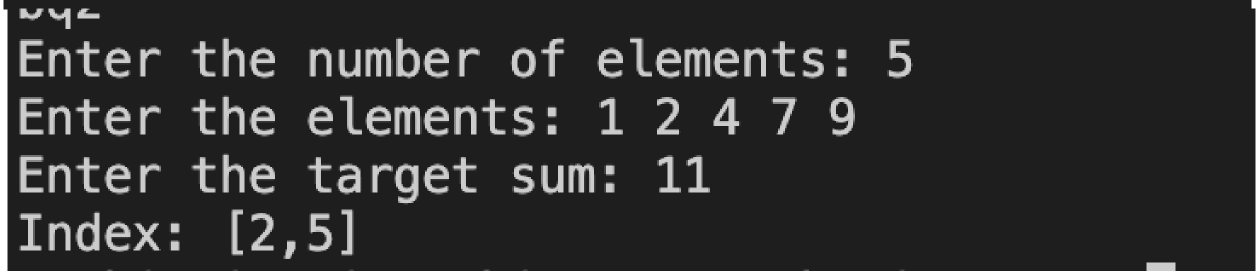
2)



```
// Given a 1D array of integers, first sort the array in non-decreasing order, and
// then find two numbers such that the sum of two numbers add up to a specific
// value. If such a pair of numbers can be found in the array, return the indices,
else
// return a suitable message.
// Example 1:
// Input: numbers = [2,7,11,15], target = 9
// Output: [1,2]
// Hint: The sum of 2 and 7 is 9.
// Example 2:
// Input: numbers = [2,3,4], target = 6
// Output: [1,3]
// Hint: The sum of 2 and 4 is 6.
#include <iostream>
using namespace std;
```

```
void bubbleSort(int arr[], int n) {
bool swapped;
for (int i = 0; i < n - 1; ++i) {

swapped = false;
for (int j = 0; j < n - i - 1; ++j) {
if (arr[j] > arr[j + 1]) {
int temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
swapped = true;
}
}
if (!swapped)
break;
}
}

void findPairWithSum(int arr[], int n, int target) {
bubbleSort(arr, n);

int left = 0;
int right = n - 1;

while (left < right) {
int sum = arr[left] + arr[right];
if (sum == target) {
cout << "Index: [" << left + 1 << "," << right + 1 << "]" << std::endl;
return;

}
else if (sum < target) {
++left;
}
else {
--right;
}
}

cout << "No such pair exists." << endl;
}

int main() {
int n;
cout << "Enter the number of elements: ";
cin >> n;

int* arr = new int[n];
cout << "Enter the elements: ";
for (int i = 0; i < n; ++i) {
cin >> arr[i];
}
```

```
int target;

cout << "Enter the target sum: ";


cin >> target;


findPairWithSum(arr, n, target);


delete[] arr;

return 0;

}
```



```
bq2
Enter the number of elements: 5
Enter the elements: 1 2 4 7 9
Enter the target sum: 11
Index: [2,5]
```

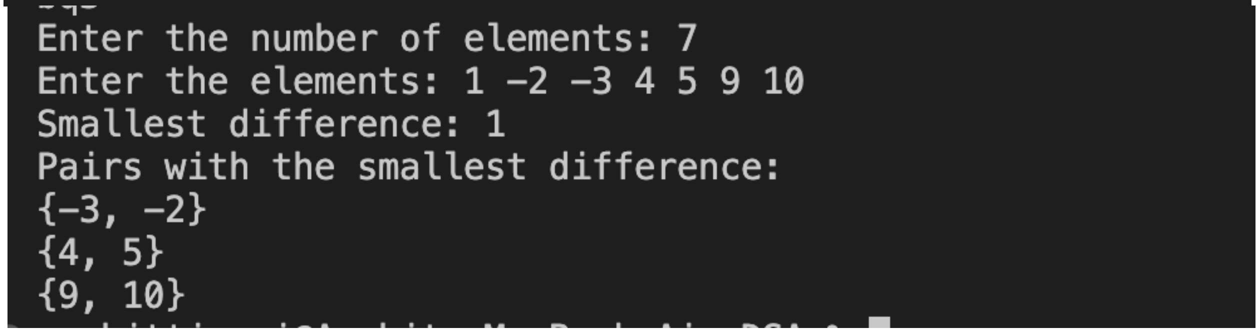
3)

```
#include <iostream>

#include <climits>

#include<math.h>

using namespace std;


void bubbleSort(int arr[], int n) {

bool swapped;

for (int i = 0; i < n - 1; ++i) {
```

```
swapped = false;
for (int j = 0; j < n - i - 1; ++j) {
if (arr[j] > arr[j + 1]) {
int temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
swapped = true;
}
}
if (!swapped)
break;
}
}

void findPairsWithSmallestDifference(int arr[], int n) {
if (n < 2) {
cout << "Not enough elements to form pairs." << std::endl;
return;
}

bubbleSort(arr, n);

int minDiff=abs(arr[0]-arr[1]);
for (int i = 1; i < n; ++i) {
int diff = arr[i] - arr[i - 1];
```

```cpp
if (abs(diff) < minDiff) {

minDiff = diff;

}

}


cout << "Smallest difference: " << minDiff << endl;

cout << "Pairs with the smallest difference: " << endl;


for (int i = 1; i < n; ++i) {

if (abs(arr[i] - arr[i - 1]) == minDiff) {

cout << "{" << arr[i - 1] << ", " << arr[i] << "}" << endl;

}

}

}


int main() {

int n;

cout << "Enter the number of elements: ";

cin >> n;


if (n <= 0) {

cout << "Number of elements must be positive." << std::endl;

return 1;

}
```

```
int* arr = new int[n];

cout << "Enter the elements: ";

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

cin >> arr[i];

}



findPairsWithSmallestDifference(arr, n);



delete[] arr;

return 0;

}
```



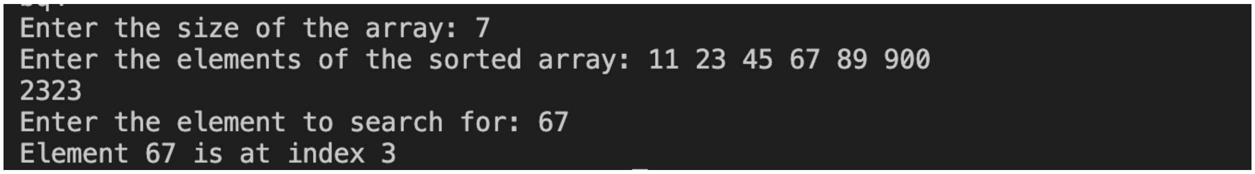
```
Enter the number of elements: 7
Enter the elements: 1 -2 -3 4 5 9 10
Smallest difference: 1
Pairs with the smallest difference:
{-3, -2}
{4, 5}
{9, 10}
```

4)

```
#include <iostream>



using namespace std;

int interpolationSearch(int arr[], int n, int k) {

int low = 0;

int high = n - 1;
```

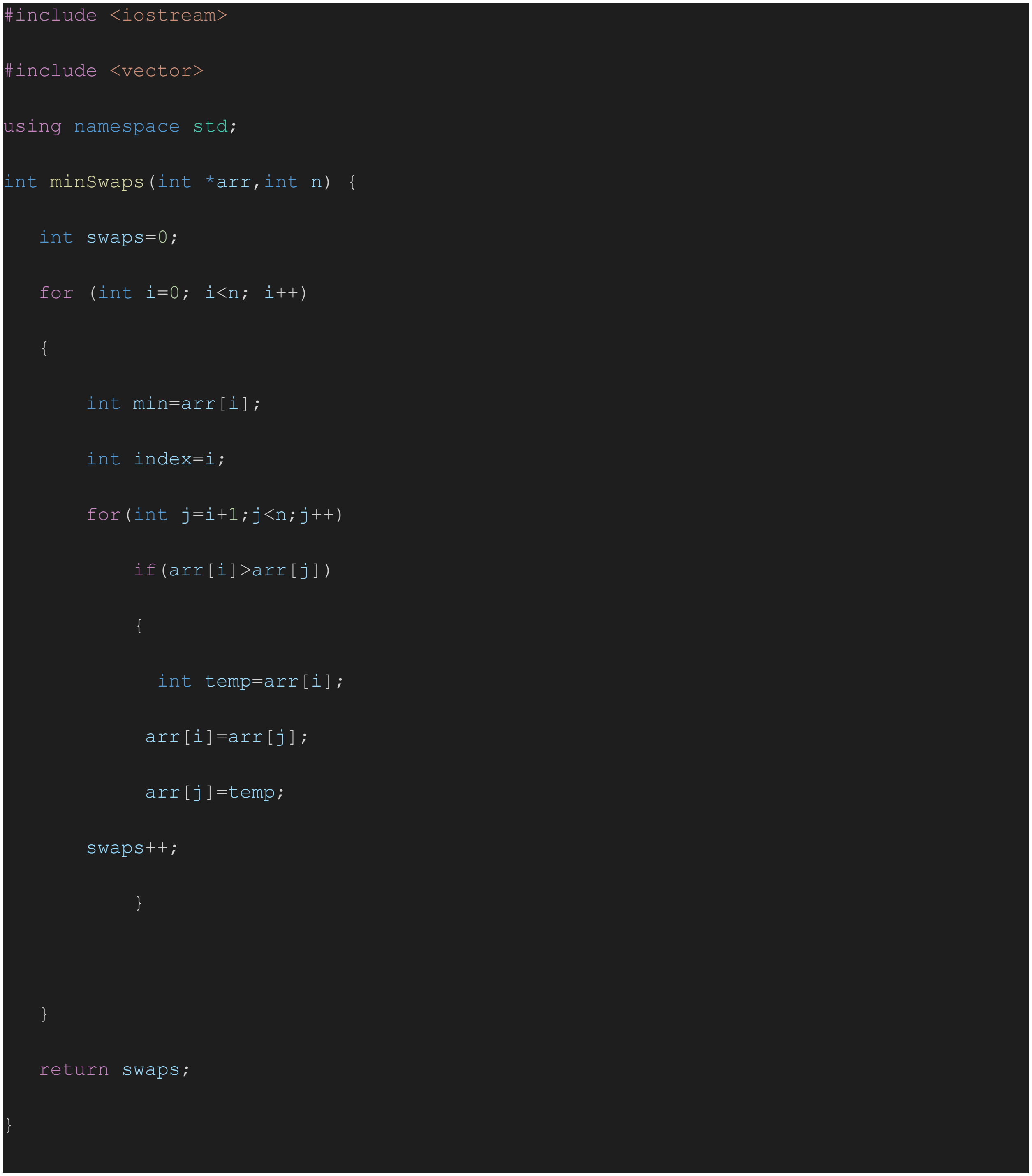
```
while (low <= high && k >= arr[low] && k <= arr[high]) {
if (low == high) {
if (arr[low] == k) {
return low;
}
return -1;
}

int pos = low + ((k - arr[low]) * (high - low) / (arr[high] - arr[low]));

if (arr[pos] == k) {
return pos;
}
if (arr[pos] < k) {
low = pos + 1;
} else {
high = pos - 1;
}
}

return -1;
}

int main() {
int n;
cout << "Enter the size of the array: ";
```

```
cin >> n;

int* arr = new int[n];
cout << "Enter the elements of the sorted array: ";
for (int i = 0; i < n; ++i) {
cin >> arr[i];
}

int k;
cout << "Enter the element to search for: ";
cin >> k;

int index = interpolationSearch(arr, n, k);

if (index != -1) {
cout << "Element " << k << " is at index " << index <<endl;
} else {
cout << "Element " << k << " is not present in the array." <<endl;
}

delete[] arr;
return 0;
}
```

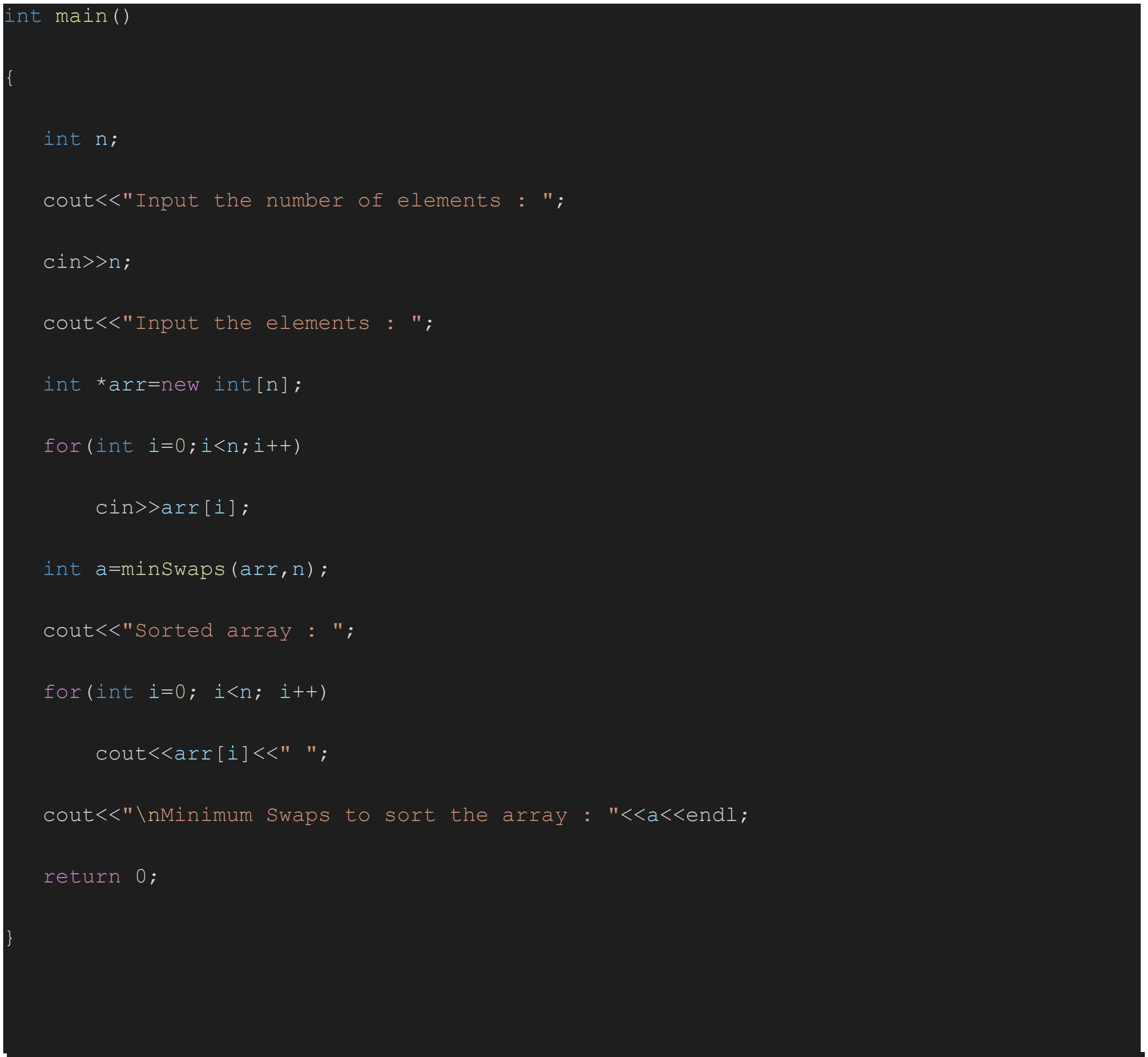


```
Enter the size of the array: 7
Enter the elements of the sorted array: 11 23 45 67 89 900
2323
Enter the element to search for: 67
Element 67 is at index 3
```

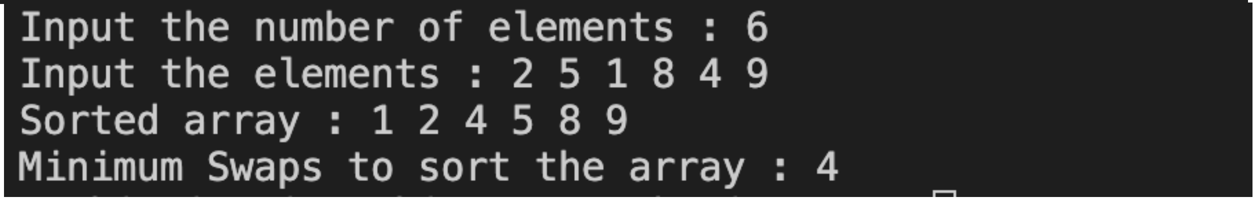
5)



```
#include <iostream>
#include <vector>
using namespace std;
int minSwaps(int *arr,int n) {
   int swaps=0;
   for (int i=0; i<n; i++)
   {
       int min=arr[i];
       int index=i;
       for(int j=i+1;j<n;j++)
           if(arr[i]>arr[j])
           {
             int temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
       swaps++;
           }

   }
   return swaps;
}
```

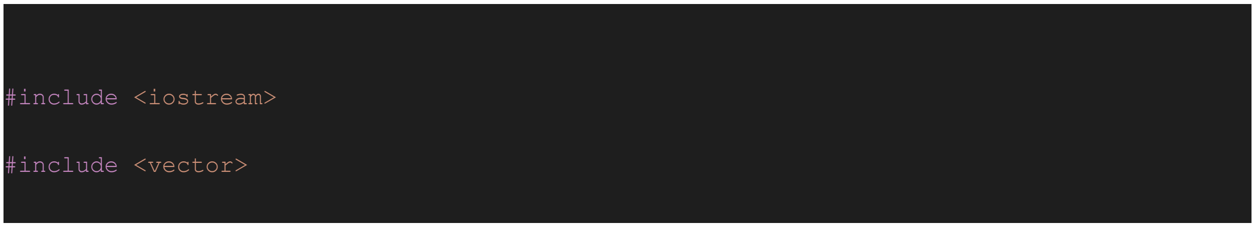


```
int main()
{
    int n;
    cout<<"Input the number of elements : ";
    cin>>n;
    cout<<"Input the elements : ";
    int *arr=new int[n];
    for(int i=0;i<n;i++)
        cin>>arr[i];
    int a=minSwaps(arr,n);
    cout<<"Sorted array : ";
    for(int i=0; i<n; i++)
        cout<<arr[i]<<" ";
    cout<<"\nMinimum Swaps to sort the array : "<<a<<endl;
    return 0;
}
```

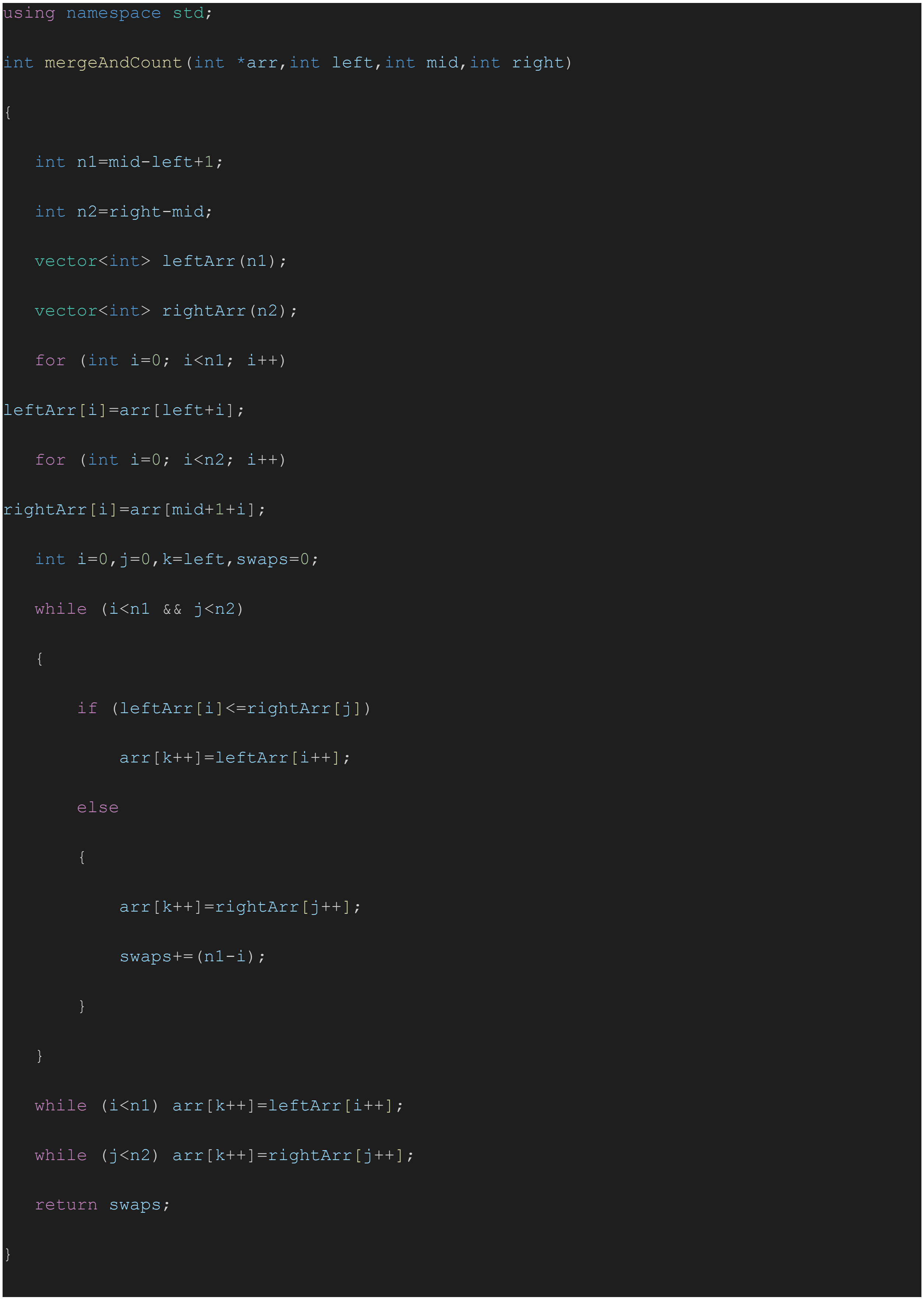


```
Input the number of elements : 6
Input the elements : 2 5 1 8 4 9
Sorted array : 1 2 4 5 8 9
Minimum Swaps to sort the array : 4
```

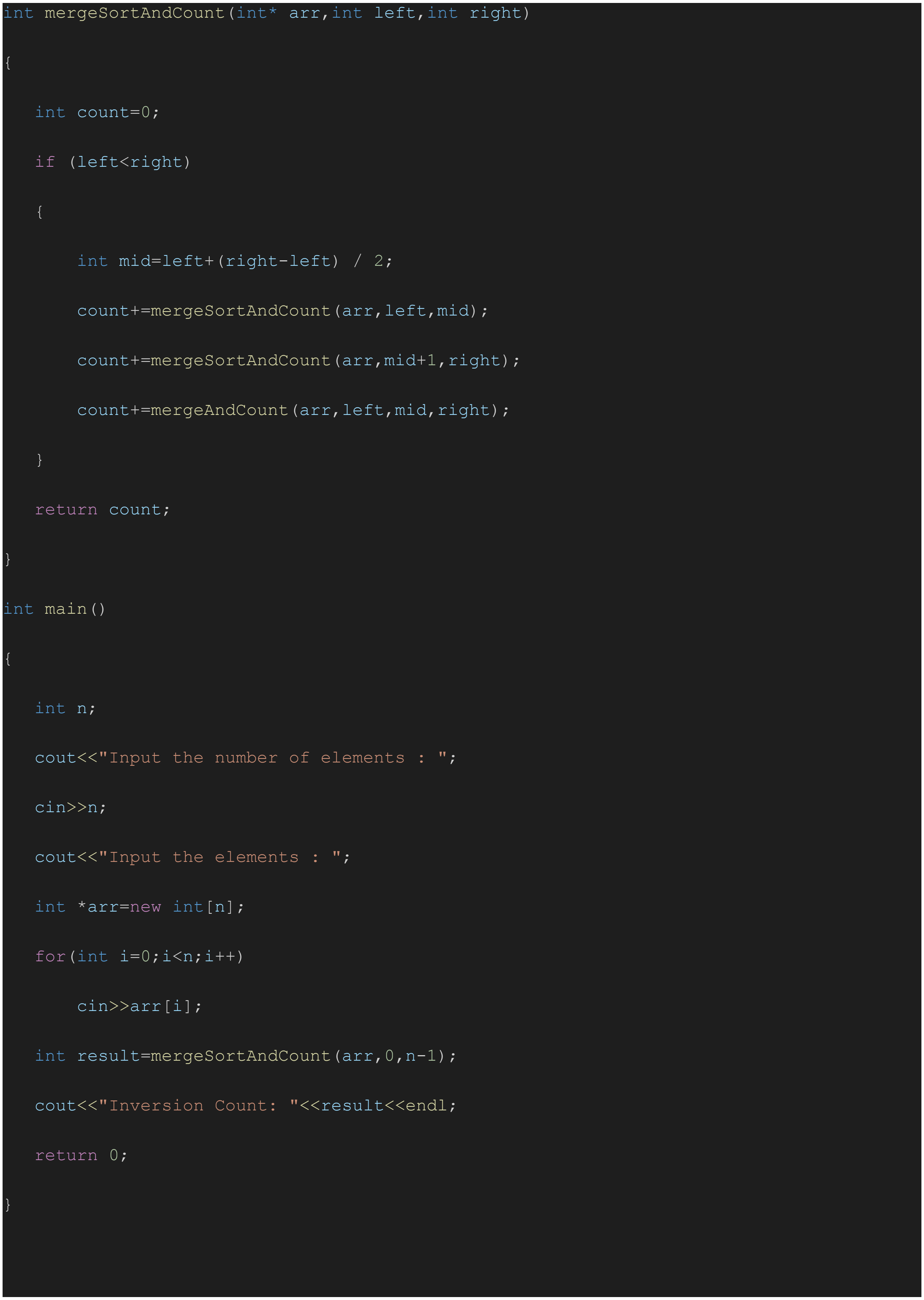
6)



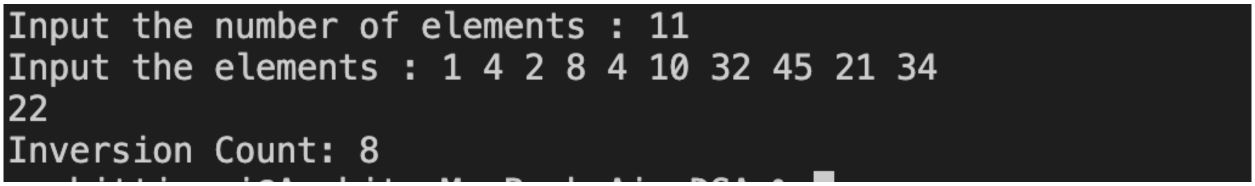
```
#include <iostream>
#include <vector>
```



```cpp
using namespace std;
int mergeAndCount(int *arr,int left,int mid,int right)
{
   int n1=mid-left+1;
   int n2=right-mid;
   vector<int> leftArr(n1);
   vector<int> rightArr(n2);
   for (int i=0; i<n1; i++)
leftArr[i]=arr[left+i];
   for (int i=0; i<n2; i++)
rightArr[i]=arr[mid+1+i];
   int i=0,j=0,k=left,swaps=0;
   while (i<n1 && j<n2)
   {
       if (leftArr[i]<=rightArr[j])
           arr[k++]=leftArr[i++];
       else
       {
           arr[k++]=rightArr[j++];
           swaps+=(n1-i);
       }
   }
   while (i<n1) arr[k++]=leftArr[i++];
   while (j<n2) arr[k++]=rightArr[j++];
   return swaps;
}
```

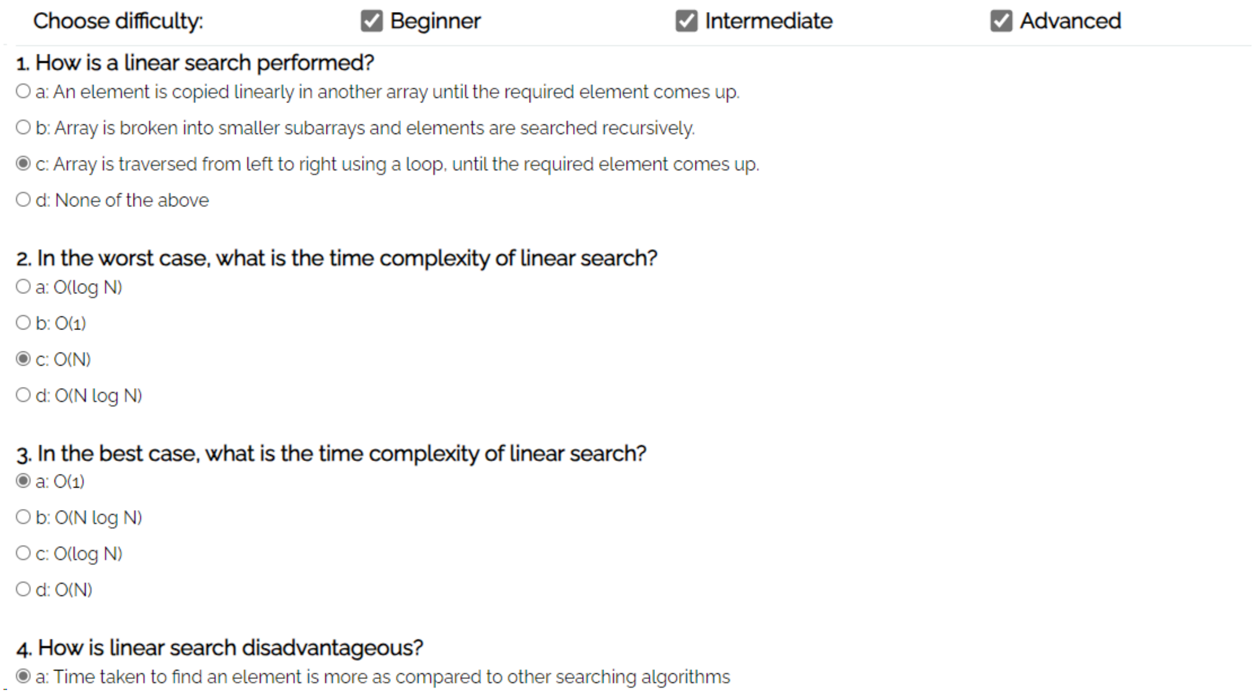


```
int mergeSortAndCount(int* arr,int left,int right)
{
   int count=0;
   if (left<right)
   {
       int mid=left+(right-left) / 2;
       count+=mergeSortAndCount(arr,left,mid);
       count+=mergeSortAndCount(arr,mid+1,right);
       count+=mergeAndCount(arr,left,mid,right);
   }
   return count;
}
int main()
{
   int n;
   cout<<"Input the number of elements : ";
   cin>>n;
   cout<<"Input the elements : ";
   int *arr=new int[n];
   for(int i=0;i<n;i++)
       cin>>arr[i];
   int result=mergeSortAndCount(arr,0,n-1);
   cout<<"Inversion Count: "<<result<<endl;
   return 0;
}
```



```
Input the number of elements : 11
Input the elements : 1 4 2 8 4 10 32 45 21 34
22
Inversion Count: 8
```

Virtual Labs :



Choose difficulty: ☑ Beginner ☑ Intermediate ☑ Advanced

1. How is a linear search performed?

○ a: An element is copied linearly in another array until the required element comes up.

○ b: Array is broken into smaller subarrays and elements are searched recursively.

● c: Array is traversed from left to right using a loop, until the required element comes up.

○ d: None of the above

2. In the worst case, what is the time complexity of linear search?

○ a: O(log N)

○ b: O(1)

● c: O(N)

○ d: O(N log N)

3. In the best case, what is the time complexity of linear search?

● a: O(1)

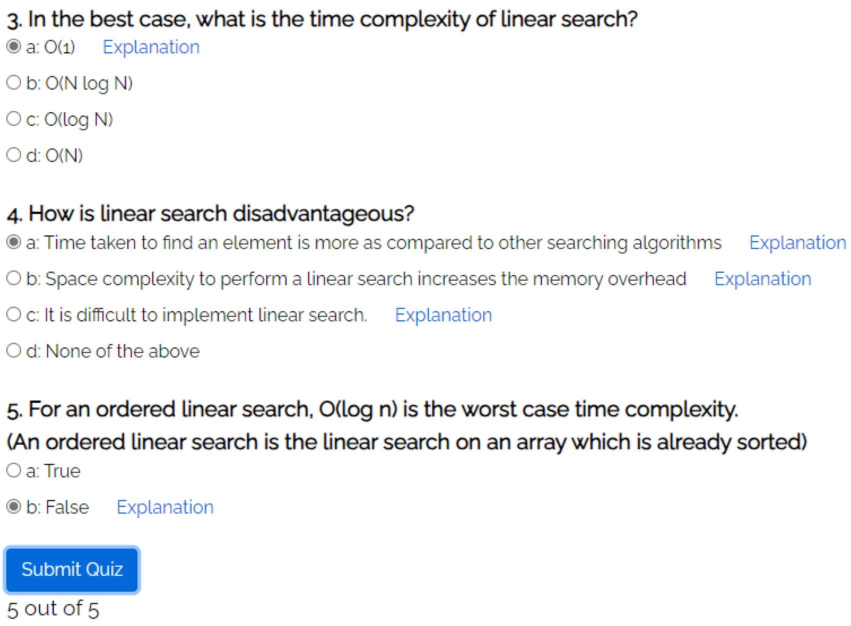
○ b: O(N log N)

○ c: O(log N)

○ d: O(N)

4. How is linear search disadvantageous?

● a: Time taken to find an element is more as compared to other searching algorithms



3. In the best case, what is the time complexity of linear search?

● a: O(1) Explanation

○ b: O(N log N)

○ c: O(log N)

○ d: O(N)

4. How is linear search disadvantageous?

● a: Time taken to find an element is more as compared to other searching algorithms Explanation

○ b: Space complexity to perform a linear search increases the memory overhead Explanation

○ c: It is difficult to implement linear search. Explanation

○ d: None of the above

5. For an ordered linear search, O(log n) is the worst case time complexity.
(An ordered linear search is the linear search on an array which is already sorted)

○ a: True

● b: False Explanation

5 out of 5