National University of Computer and Emerging Sciences



Lab Manual # 10 Programming Fundamentals

|  |  |
| --- | --- |
| Course Instructor | Ms. Anoosha Khan |
| Lab Instructor(s) | Samia Akhter  Wajahat Ali |
| Section | BSE-1A |
| Semester | Fall 2022 |

# Instructions:

* Do your tasks individually
* Anyone indulged in the act of plagiarism would be awarded zero
* Pay attention to details, do paper work before starting your code
* Understanding of the question is also part of the lab work

# Objectives:

In this lab, students will study:

* Sorting
* Arrays

**Task 1:** Write C++ program that initializes an array of ten integers (sorted array). It inputs an integer from the user and searches the value in the array doing the following tasks:

* First locate the middle element of array and compare with the search number
* If they are equal, search is successful and the index of middle element is returned
* If they are not equal, reduces the search to the half of the array
* If the search number is less than the middle element, the first half of the array search, otherwise the second half of the array is search
* The process continues until the required number is found or loop completes without successful search

**Output:**

Enter values: 10 20 30 40 50 60 70 80 90 100 Enter any number to find: 30

Output: 30 found at index 2

**Task 2: (1D-Array) :**Write a function that search an integer in given array using binary search and returns the bool value as, 1 or 0 depending integer is present in given array or not.

**Sample Input**: { 4,6,7,1,9,5,2,8,3} , 10

**Output:** No, given integer is not present in the given array

**Sample Input:** { 4,6,7,1,9,5,2,8,3} , 5

**Sample Output:** Yes, given integer is present in the given array.

**Prerequisite for Binary Search: Array should be sorted.**

**For sorting the array write a function that sort the given array in ascending order using bubble sort.**

**Task 3:** Write C++ program that initializes an array of ten integers. It should output the largest value of an array after sorting the array by doing the following tasks:

* Initially take first two element of an array and compare them which one is smaller. Place the smaller one on 1st index of an array.
* Now take element of an array on 3rd index and compare it with element present on 2nd index. Store the smaller one on 2nd index by swapping and then compare 3rd index element with 1st index element and swap them until the 3rd index element placed on right order in ascending order.
* Keep on comparing and swapping the elements until you get the sorted array and then print the element present on last index.

**For Example: {12 11 14 5 6 4 15 10 3 2}**

* First compare12 and 11 as 11 is smaller than 12 so swap them. **{11 12 14 5 6 4 15 10 3 2}**
* Now compare 14 and 12 as 12 is smaller than 14 so no need to swap. Similarly compare 14 with 11 as 11 is also smaller than 14 so no need to swap **{11 12 14 5 6 4 15 10 3 2}**
* Now compare 5 with all previous elements one by one and place it on right position in ascending order. **{5 11 12 14 6 4 15 10 3 2}**
* Keep on comparing and swapping the elements until you get the sorted array. **{2 3 4 5 6 10 11 12 14 15}**
* Then print the (largest) element present on last index. **“The largest element is 15”**

**Task 4:** Write C++ program that initializes an array of ten integers. It should output the smallest value of an array after sorting the array by using selection sort.

**Implement selection sort Algorithm**

The **selection sort algorithm** sorts an array by repeatedly finding the minimum element (considering ascending order) from the unsorted part and putting it at the beginning.

The algorithm maintains two subarrays in a given array.

* The subarray which already sorted.
* The remaining subarray was unsorted.

In every iteration of the selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

**How does selection sort work?**

Let’s consider the following array as an example:

**arr[] = {64, 25, 12, 22, 11}**

**First pass:**

For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where **64**is stored presently, after traversing whole array it is clear that **11**is the lowest value.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **64** | 25 | 12 | 22 | 11 |

Thus, replace 64 with 11. After one iteration **11**, which happens to be the least value in the array, tends to appear in the first position of the sorted list.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **11** | 25 | 12 | 22 | 64 |

**Second Pass:**

For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | **25** | 12 | 22 | 64 |

After traversing, we found that **12** is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | **12** | 25 | 22 | 64 |

**Third Pass:**

Now, for third place, where **25** is present again traverse the rest of the array and find the third least value present in the array.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 12 | **25** | 22 | 64 |

While traversing, **22**came out to be the third least value and it should appear at the third place in the array, thus swap **22**with element present at third position.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 12 | **22** | 25 | 64 |

**Fourth pass:**

Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array

As **25**is the 4th lowest value hence, it will place at the fourth position.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | 12 | 22 | **25** | 64 |

**Fifth Pass:**

At last the largest value present in the array automatically get placed at the last position in the array

The resulting array is the sorted array.

| 11 | 12 | 22 | **25** | 64 |
| --- | --- | --- | --- | --- |

## 