

**ANDREAS ANDREOU**

**LEONIDAS LEONIDA**

**LOUIZA SAVVA**

Contents

[**Program 1:** 2](#_Toc3152)

[**Program 2:** 2](#_Toc3153)

[**Program 3:** 3](#_Toc3154)

*We confirm that the work is entirely our own and that no unauthorized programs were used, or we collaborated with other people, in a way that violates academic policies.*

# Program 1:

# The program reads two sorted lists of integers from the keyboard, using the character # to mark the end of each list. It then compares the lists simultaneously using a two-pointer approach to efficiently find their common elements. Each common element is stored in a vector, and their total sum is calculated.

# The algorithm makes at most **m + n − 1 comparisons**, where *m* and *n* are the sizes of the two lists, because in the worst case each step advances one of the two pointers until at least one list is fully traversed.

# **Example Execution:**

# Input: 2, 3, 4, 4, 6, 10, 18, # and 2, 4, 4, 10, 13, 17, #

# Output: Common elements: 2, 4, 4, 10

# Sum of common elements: 20

# This approach ensures efficiency compared to a brute force solution, since it leverages the fact that both lists are sorted, making the complexity **O(m + n)**.

Pseudocode

Input: Two sorted lists list1 and list2 (terminated by "#")

Output: Common elements and their sum

1. Read list1 until "#" is encountered

2. Read list2 until "#" is encountered

3. Initialize i ← 0, j ← 0

4. Initialize common ← empty list

5. Initialize sum ← 0

6. Initialize comparisons ← 0

7. While i < length(list1) AND j < length(list2) do

comparisons ← comparisons + 1

If list1[i] = list2[j] then

Add list1[i] to common

sum ← sum + list1[i]

i ← i + 1

j ← j + 1

Else if list1[i] < list2[j] then

i ← i + 1

Else

j ← j + 1

End While

8. Print "Common elements:", common

9. Print "Sum of common elements:", sum

10. Print "Maximum comparisons: m + n - 1"

11. Print "Actual comparisons made:", comparisons

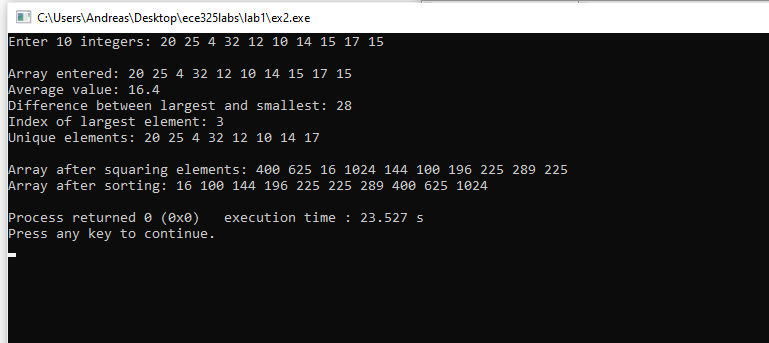
End Algorithm

**Program 2:**

The program reads 10 integers into an array and performs several operations:

* Calculates the **average value** of the elements.
* Finds the **difference between the largest and smallest values**.
* Determines the **index of the largest element**.
* Prints the **unique elements** (those that appear only once).
* Squares all array elements.
* **Sorts** the array in ascending order.

The results of each step are displayed to the user. Most operations run in linear time **O(n)**, sorting runs in **O(n log n)**, and the unique-element check is **O(n²)** due to nested loops. The program demonstrates basic array processing, searching, and sorting techniques in C++.



Pseudocode

Read 10 integers into arr[10]

Print original array

// (a) Find average

sum ← 0

For each element in arr:

sum ← sum + element

average ← sum / 10

Print average

// (b) Difference max - min

minVal ← arr[0]

maxVal ← arr[0]

For each element in arr:

If element < minVal then minVal ← element

If element > maxVal then maxVal ← element

Print maxVal - minVal

// (c) Index of maximum

maxIndex ← 0

For i from 1 to 9:

If arr[i] > arr[maxIndex] then maxIndex ← i

Print maxIndex

// (d) Find unique elements

For each element arr[i]:

count ← 0

For each element arr[j]:

If arr[i] = arr[j] then count ← count + 1

If count = 1 then print arr[i]

// (e) Square elements

For each element in arr:

arr[i] ← arr[i] \* arr[i]

// (f) Sort array

Sort arr in ascending order

Print arr