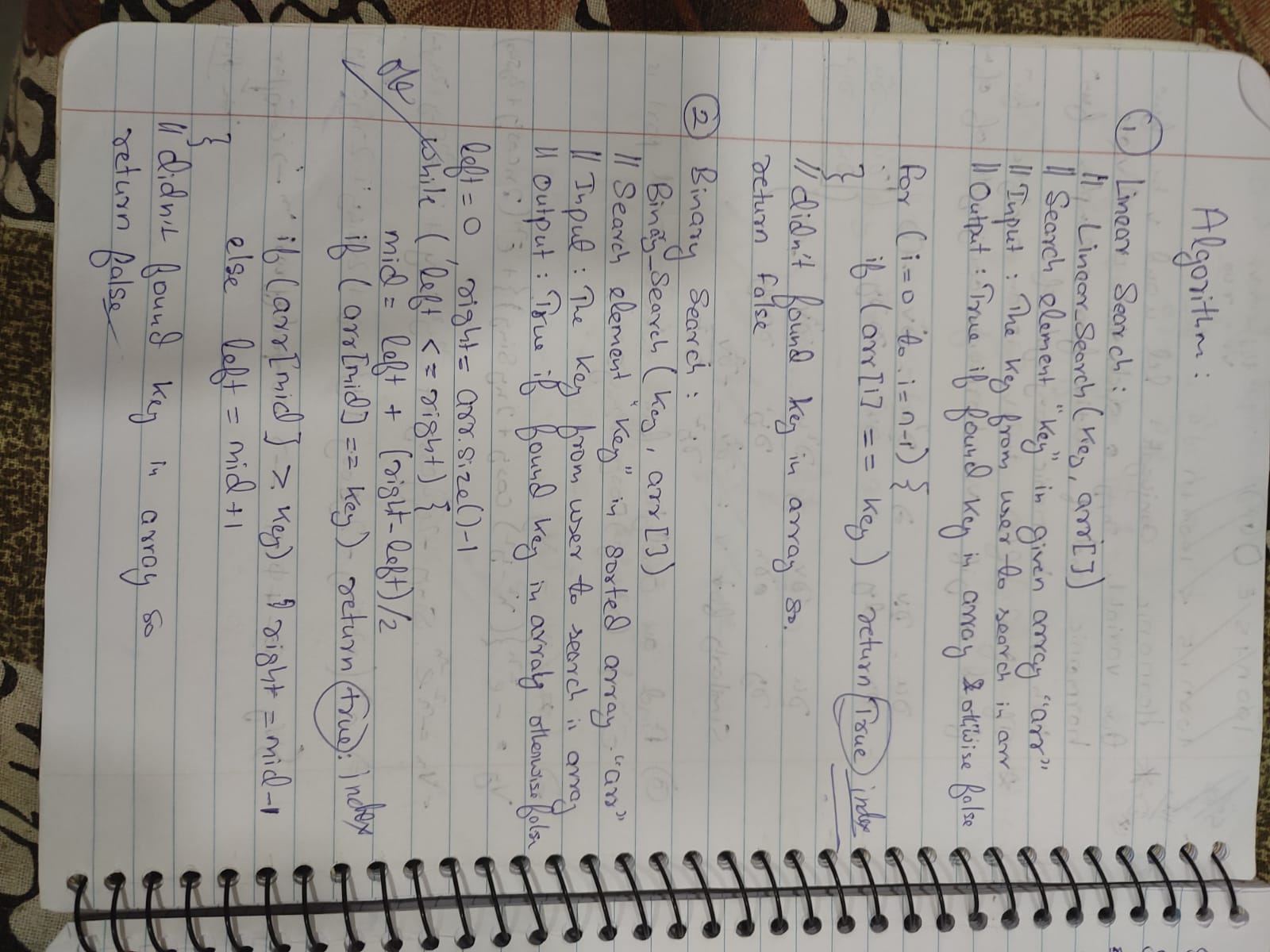
**Name: Nilanshu Ranjan**

**Roll No: 231070035**

**Branch: Computer Engineering**

**DAA Assignment 2**

**Algorithm :**

****

**Code :**

#include <bits/stdc++.h>

using *namespace* std;

#define ll *long* *long*

*int* linear\_search(vector<*int*> *arr*, *int* *key*)

{

    for (*int* i = 0; i < *arr*.size(); i++)

    {

        if (*arr*[i] == *key*)

        {

            return i;

        }

    }

    return -1;

}

*int* binary\_search(vector<*int*> *arr*, *int* *key*)

{

*int* left = 0, right = *arr*.size() - 1;

    sort(*arr*.begin(), *arr*.end());

    while (left <= right)

    {

*int* mid = left + (right - left) / 2;

        if (*arr*[mid] == *key*)

        {

            return mid;

        }

        if (*arr*[mid] < *key*)

            left = mid + 1;

        else

            right = mid - 1;

    }

    return -1;

}

*int* main()

{

*int* n;

    cout << "Number of element in arr: ";

    cin >> n;

    vector<*int*> arr(n);

    cout << "Enter element: ";

    for (*int* i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

*int* key;

    cout << "Enter the key to search: ";

    cin >> key;

    if (linear\_search(arr, key) != -1) cout << "Number found at index (Linear Search): "<< linear\_search(arr, key) << endl;

    else cout << "Number not found" << endl;

    if (binary\_search(arr, key) != -1)

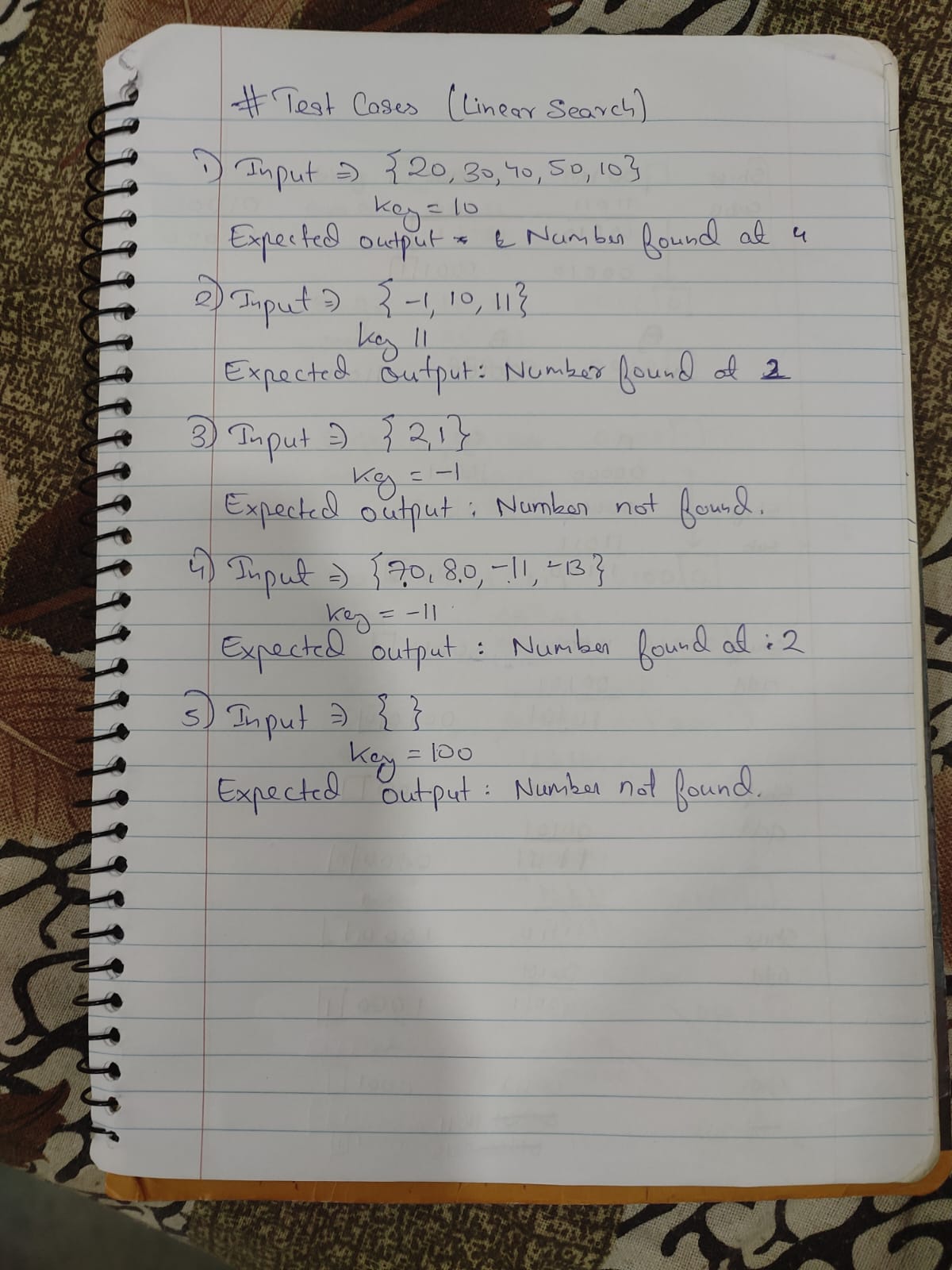
        cout << "Number found at index (Binary Search): " << binary\_search(arr, key) << endl;

    else

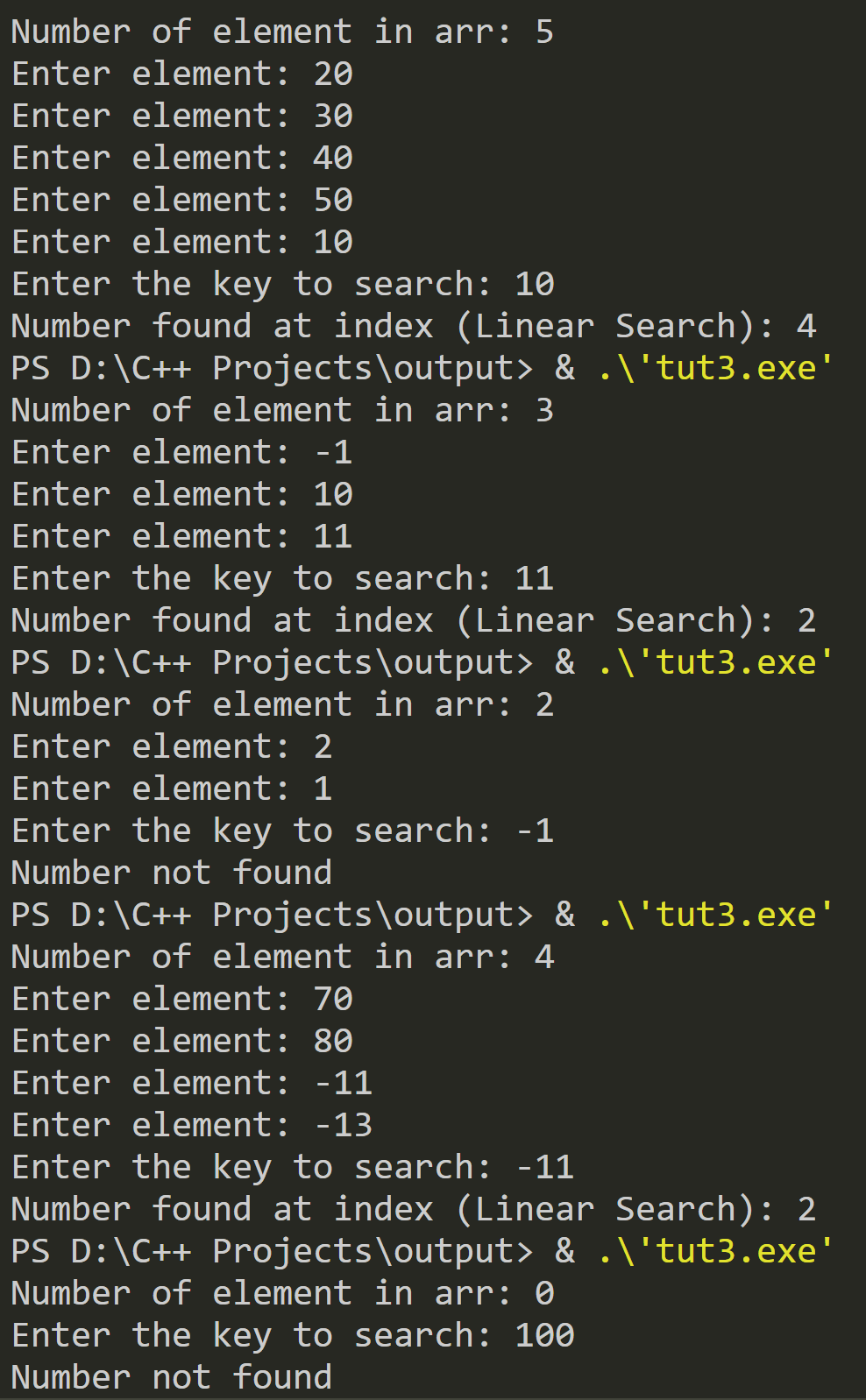
        cout << "Number not found" << endl;

}

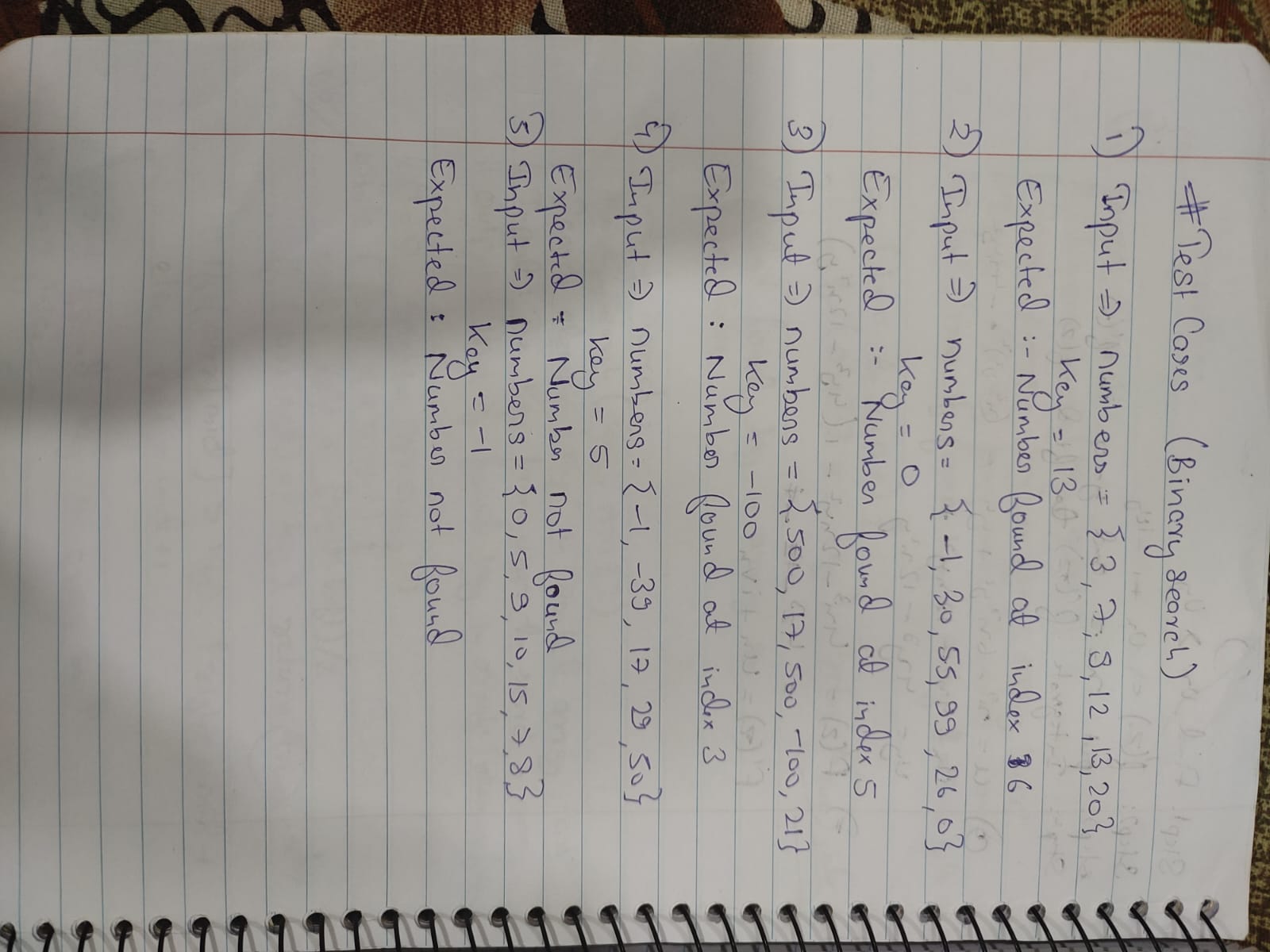
**Test Cases: ( Linear Search)**

****

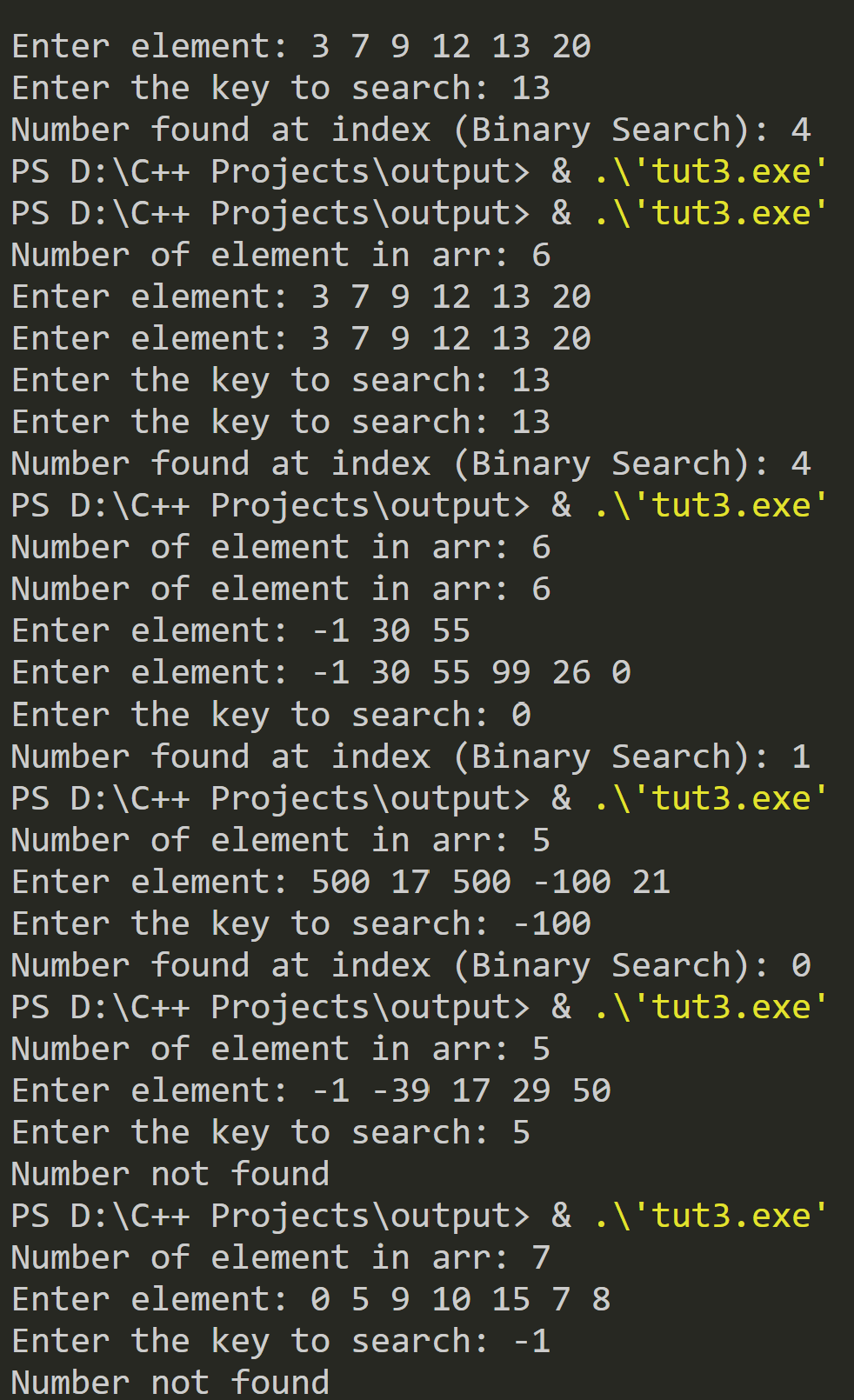
**Output for testcases (Linear search):**

****

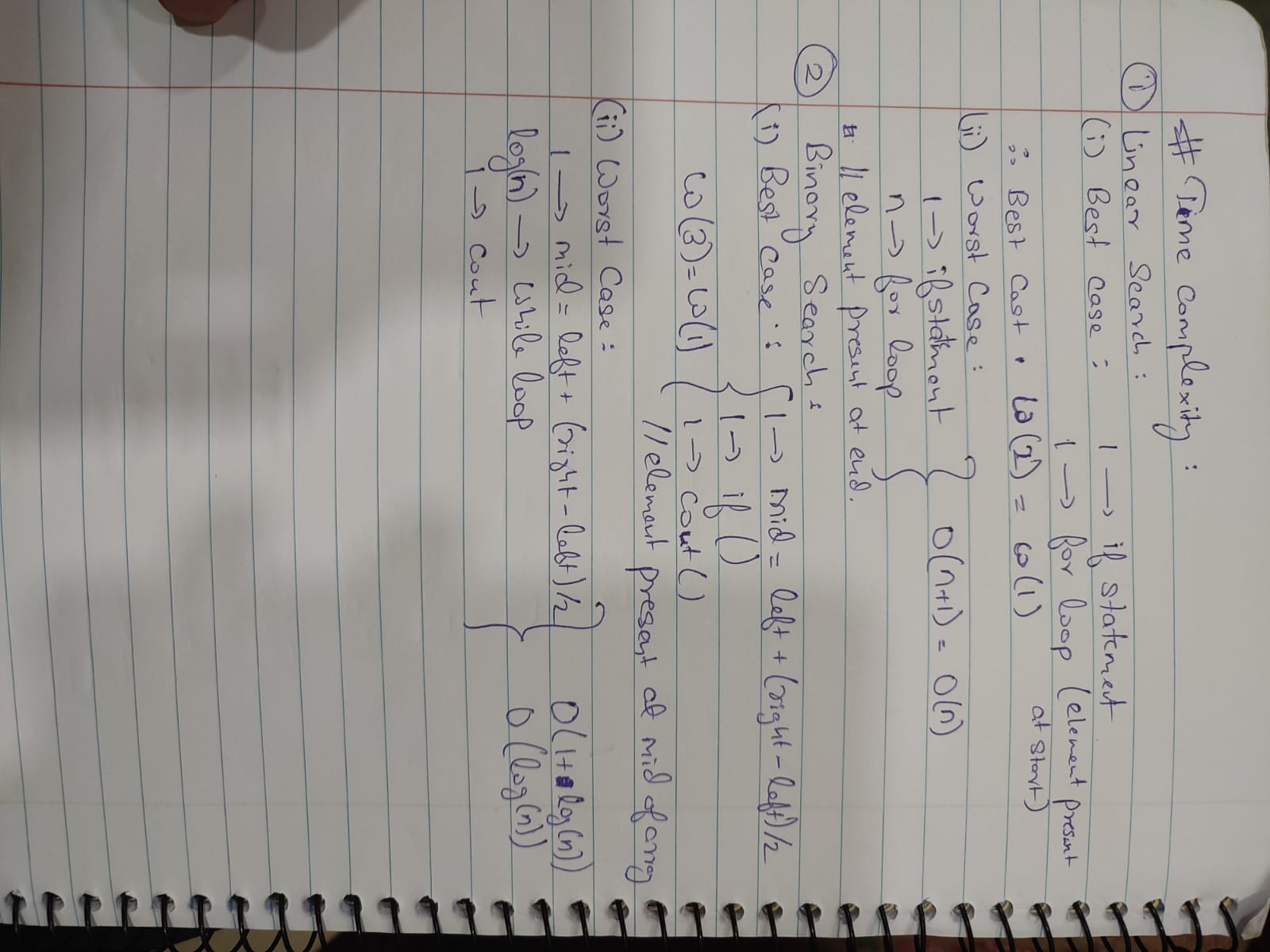
**Testcases for Binary search :**

****

**Output for testcases:**

****

**Time Complexity:**

****

**Conclusion:**

The provided code demonstrates two fundamental search algorithms: linear search and binary search.

1. Linear Search:
   * This algorithm sequentially checks each element in the list until it finds the target key or reaches the end.
   * It has a time complexity of O(n), where n is the number of elements in the list. This makes it straightforward but potentially slow for large datasets, especially if the target is near the end or not present at all.
2. Binary Search:
   * Binary search is more efficient, but it requires the list to be sorted. If the list is not sorted, the code sorts it first, which adds to the complexity.
   * The search process repeatedly divides the list in half, quickly narrowing down the possible locations of the target key.
   * The sorting step has a time complexity of O(n log n), and the search itself is O(log n). This makes binary search significantly faster than linear search for large, sorted datasets.

* Linear Search is simple and works on unsorted data but is less efficient for large lists.
* Binary Search is much faster but requires the data to be sorted. It is the preferred method when dealing with large datasets where the overhead of sorting is acceptable.

The code effectively illustrates the trade-offs between the two methods, with linear search being easier to implement but slower, and binary search being more complex but faster for sorted data.