Activity No. <n>	
<Seatwork 6.1 Linear and Binary Search >	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 09/11/25
Section: CPE 010-CPE21S4	**Date Submitted:** 09/11/25
Name(s): Kerwin Jan B. Catungal	**Instructor:** Jimlord Quejado

6. Output

Answer the following questions:

1. What is a search tree in data structures?

A Binary Search Tree (BST) is a type of binary tree data structure in which each node contains a unique key and satisfies a specific ordering property and also organizing data like a tree structure so we can quickly find, insert, or delete elements.

2. What are the Different types of search algorithm in data structures? Differentiate each type of search.

The two main types of search are linear search, which checks items one by one, and binary search, which repeatedly divides the data in half to find the target faster

3. What operations / implementations can be performed using binary and linear search operations?

Linear and binary search can be used to look for specific elements, check if something exists, or get the position of an item in a list or array.

4. What are the advantages in using binary search tree as data structure?

A binary search tree is good because it keeps data sorted, allows faster searching compared to linear search, and makes insertions or deletions more efficient.

5. Give an example program using binary search and Linear search.

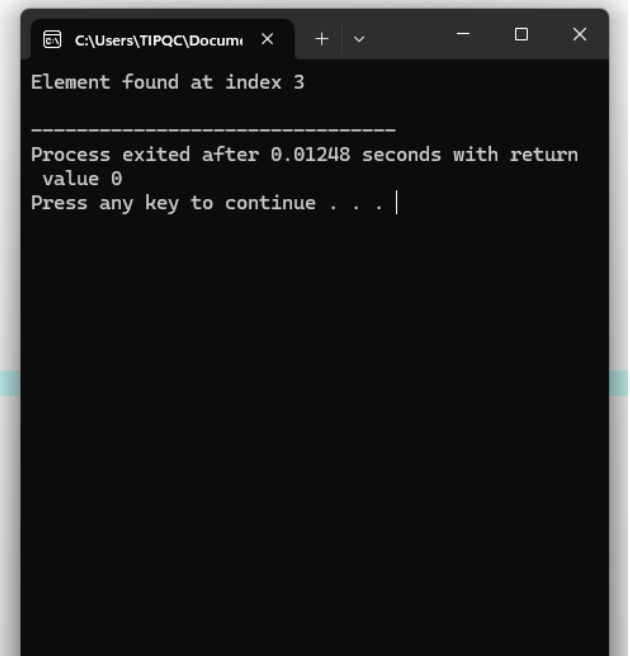
LINEAR SEARCH:

```cpp
#include <iostream>
using namespace std;

// Function to perform Linear Search
int linearSearch(int arr[], int size, int target) {
    for(int i = 0; i < size; i++) {
        if(arr[i] == target)
            return i; // Target found at index i
    }
    return -1; // Target not found
}

int main() {
    int data[] = {34, 78, 12, 56, 89, 23};
    int size = sizeof(data)/sizeof(data[0]);
    int target = 56;
    int result = linearSearch(data, size, target);
    if(result != -1)
        cout << "Element found at index " << result << endl;
    else
        cout << "Element not found in the array." << endl;
    return 0;
}

// Output:
// Element found at index 3
```



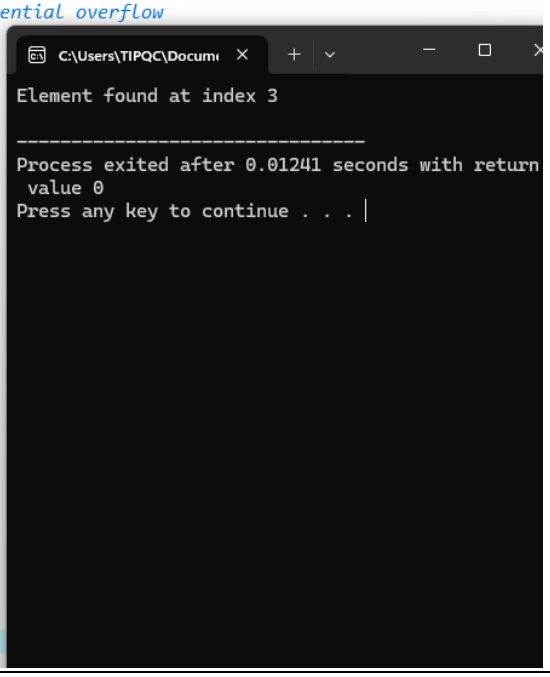
BINARY:

ex.cpp

```cpp
using namespace std;

// Function to perform Binary Search
int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size -1;
    while(left <= right) {
        int mid = left + (right - left) / 2; // Prevents potential overflow
        // Check if target is present at mid
        if(arr[mid] == target)
            return mid;
        // If target greater, ignore left half
        if(arr[mid] < target)
            left = mid + 1;
        // If target is smaller, ignore right half
        else
            right = mid - 1;
    }
    return -1; // Target not found
}
int main() {
    int data[] = {12, 23, 34, 56, 78, 89};
    int size = sizeof(data)/sizeof(data[0]);
    int target = 56;
    int result = binarySearch(data, size, target);
    if(result != -1)
        cout << "Element found at index " << result << endl;
    else
        cout << "Element not found in the array." << endl;
    return 0;
}

// Output:
// Element found at index 3
```



Paul E. Black, "search tree", in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed. 14 December 2005.

GeeksforGeeks. (n.d.). *Searching algorithms*. GeeksforGeeks. Retrieved September 11, 2025,

Singh, R. (2025, January 3). *Understanding searching algorithms in C++: Guide to linear and binary search*. Medium. https://medium.com/@RobuRishabh/understanding-searching-algorithms-in-c-guide-to-linear-and-binary-search-1c0db52dba9e

8. Conclusion

I understood that linear and binary search are useful ways to find data, with binary search being faster but needing sorted data. And also using a binary search tree helps keep data organized and makes searching and updating more efficient.

9. Assessment Rubric