Seatwork 6.1	
Linear and Binary Search	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:9/11/25**
Section: CPE21S4	**Date Submitted: 9/11/25**
Name(s): Francis Nikko I. Andoy	**Instructor:** Jimlord Quejado
6. Output	

Answer the following questions:

1. What is a search tree in data structures?

- This is a data structure used to storing data in a sorted manner. Each node in a Binary Search Tree has at most two children, a left child and a right child, with the left child containing values less than the parent node and the right child containing values greater than the parent node. This hierarchical structure allows for efficient searching, insertion, and deletion operations on the data stored in the tree.

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

- There are two kinds of search algorithms. The first one is linear search algorithm, we iterate over all the elements of the array and check if it the current element is equal to the target element. If we find any element to be equal to the target element, then return the index of the current element. Otherwise, if no element is equal to the target element, then return -1 as the element is not found. Linear search is also known as sequential search. The second is the binary search, It is used for a sorted array. It mainly compares the array's middle element first and if the middle element is same as input, then it returns. Otherwise it searches in either left half or right half based on comparison result

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

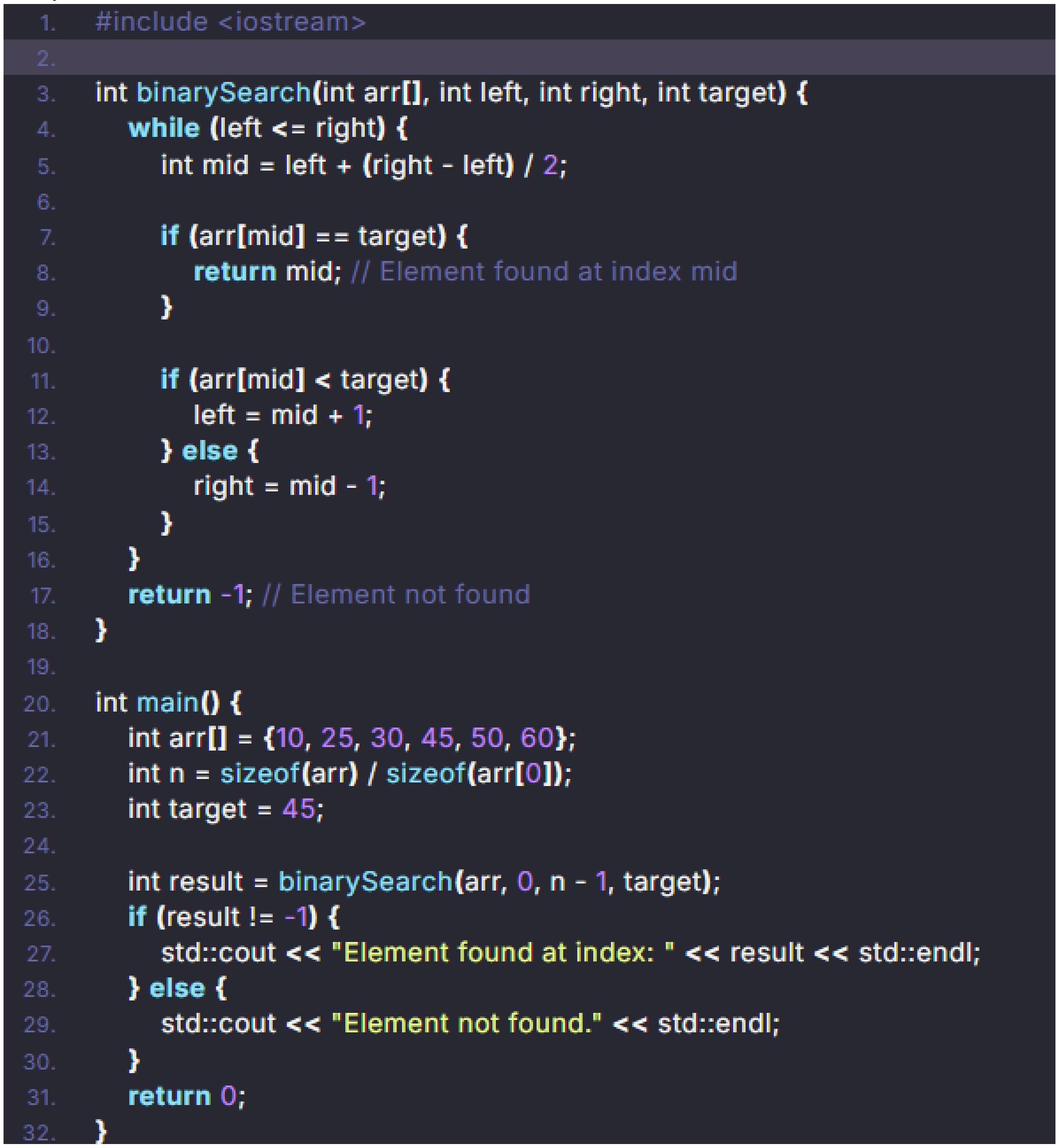
- There are two kinds of implementation that we can use, these are the array and the linked list. Since an array is just a static way of operating we can just set the size to whatever we like, but there is also a dynamic array that just resizes the array but this is inefficient to us. As for the linked list, we can add as much data as we want, since there's no fixed size when we use a linked list. We can use a search algorithm for this kind of implementation. We can also use the insertion and deletion in this kind of data structure.

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

- The advantages of a binary search tree first is efficient searching search operations this can be performed in O(log n) time complexity on average. This makes BSTs an excellent choice for applications that require frequent searching. Second is efficient insertion and deletion when adding or removing a node from a BST, the tree is automatically rebalanced to maintain the binary search property. This ensures that the tree remains efficient even after many insertions and deletions. Third is memory efficiency, this can be implemented using pointers or references, which allows for efficient use of memory. Fourth is sorted order, the sorted order of a BST makes it easy to perform operations that require the data to be sorted, such as finding the maximum or minimum value, or performing in-order traversal. Lastly is the flexibility, BSTs are highly flexible and can be adapted to suit a wide range of applications. For example, a BST can be used to implement a priority queue, a symbol table, or a balanced binary tree.

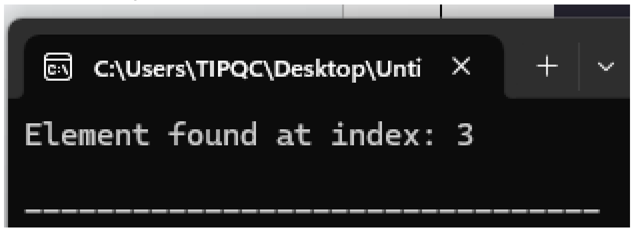
5. Give an example program using binary search and Linear search.

- binary search



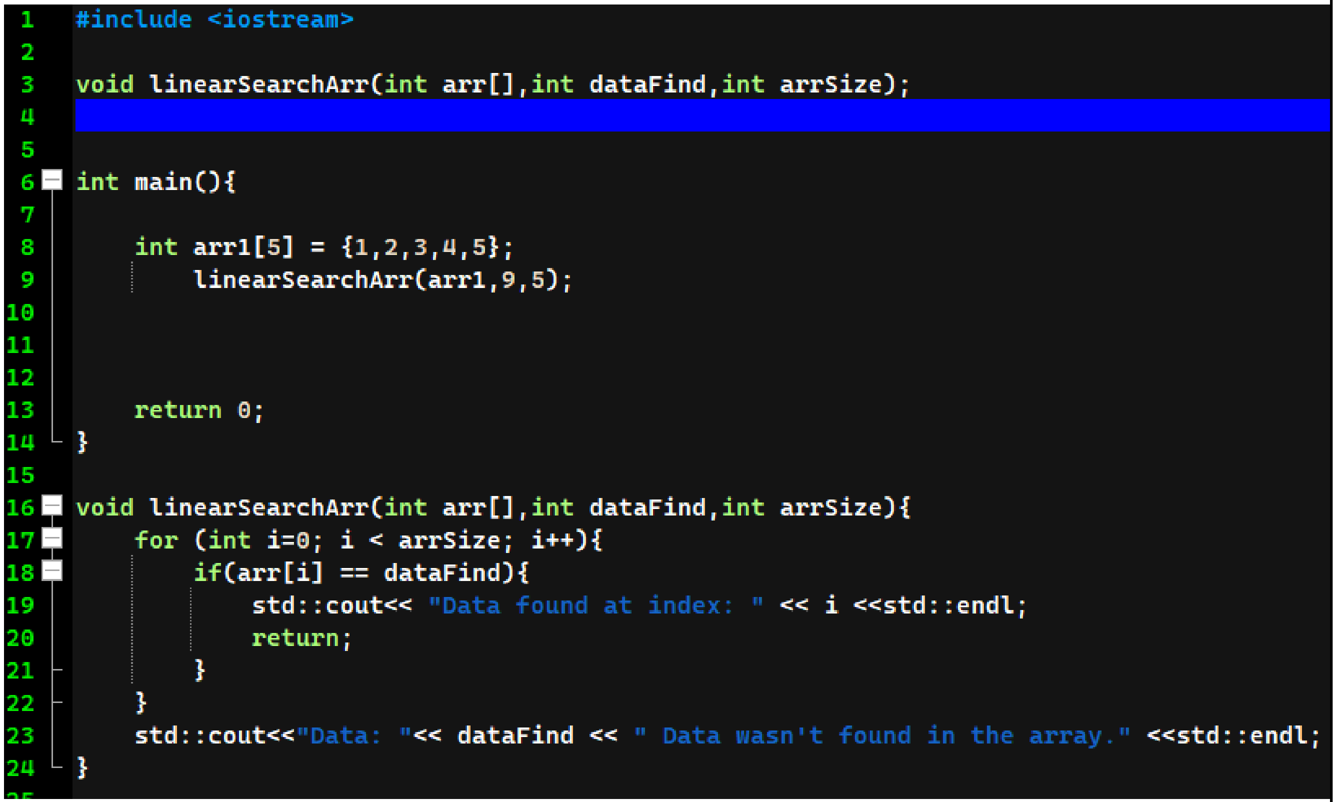
```cpp
#include <iostream>

int binarySearch(int arr[], int left, int right, int target) {
    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
            return mid; // Element found at index mid
        }

        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1; // Element not found
}

int main() {
    int arr[] = {10, 25, 30, 45, 50, 60};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 45;

    int result = binarySearch(arr, 0, n - 1, target);
    if (result != -1) {
        std::cout << "Element found at index: " << result << std::endl;
    } else {
        std::cout << "Element not found." << std::endl;
    }
    return 0;
}
```

- output



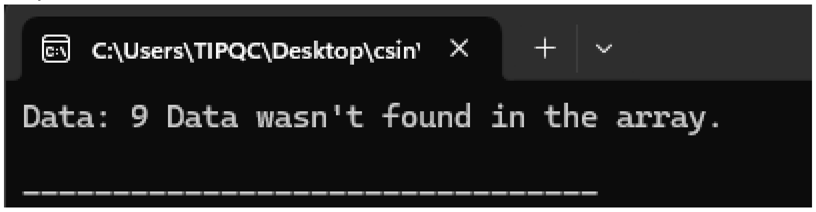
```
Element found at index: 3

--------------------------------
```

- linear search



```
#include <iostream>

void linearSearchArr(int arr[],int dataFind,int arrSize);


int main(){

    int arr1[5] = {1,2,3,4,5};
        linearSearchArr(arr1,9,5);



    return 0;
}

void linearSearchArr(int arr[],int dataFind,int arrSize){
    for (int i=0; i < arrSize; i++){
        if(arr[i] == dataFind){
            std::cout<< "Data found at index: " << i <<std::endl;
            return;
        }
    }
    std::cout<<"Data: "<< dataFind << " Data wasn't found in the array." <<std::endl;
}
```

- output:



```
Data: 9 Data wasn't found in the array.

--------------------------------
```

References:

1. **GeeksforGeeks. (2025, July 23). *Applications, Advantages and Disadvantages of Binary Search Tree*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/applications-advantages-and-disadvantages-of-binary-search-tree/**

2. **GeeksforGeeks. (2025b, August 6). *Linear Search Algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/linear-search/**

3. **GeeksforGeeks. (2025b, July 23). *Applications, Advantages and Disadvantages of Binary Search Tree*. GeeksforGeeks.**

https://www.geeksforgeeks.org/dsa/applications-advantages-and-disadvantages-of-binary-search-tree/

4. ***Binary Search Tree and its operations with algorithm and Examples*. (n.d.).** **https://techskillguru.com/ds/binary-search-tree**

5. **Loner, L. (n.d.). *Linear and Binary Searching Algorithm using C++ | DSA*. Learn Loner.** **https://learnloner.com/linear-and-binary-searching-algorithm-using-c/**

7. Supplementary Activity
8. Conclusion
9. Assessment Rubric