Activity No. <n>	
<Replace with Title>	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 9/14/25**
Section: CPE21S4	**Date Submitted: 9/4/25**
Name(s): Alcantara, Jason P.	**Instructor: Engr. Jimlord Quejado**

6. Output:

ILO A:

Syntax:

ILO_A.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

const int max_size = 50;

int main() {
    int dataset[max_size];

    // Seed the random number generator
    std::srand(std::time(0));

    // Generate random values and store them in the dataset
    for (int i = 0; i < max_size; ++i) {
        dataset[i] = std::rand();
    }

    // Output the generated dataset
    std::cout << "Generated dataset (" << max_size << " values):\n";
    for (int i = 0; i < max_size; ++i) {
        std::cout << dataset[i] << " ";
    }

    std::cout << std::endl;
    return 0;
}

```

Output:

```
C:\Users\Joshua\Documents\ILO_A.exe
Generated dataset (50 values):
26475 6727 17596 19955 25467 28298 3291 19484 30122 24890 19275 5769 10607 8202 18117 10650 22522 29811 25518 21944 22366 17996
 27728 30346 9461 11418 11883 10860 3840 169 4588 7709 8591 10076 3324 16567 23201 1217 6460 29669 27019 19944 3663 3152 32158
11510 1434 13914 26628 27509


--------------------------------
Process exited after 0.1004 seconds with return value 0
Press any key to continue . . .
```

Explanation:

This showcased the generating of random integers, which can be seen as the basis for the search algorithms.

6.2 A:

Main CPP:

```cpp
#include "searching.h"

const int max_size = 50;

int main() {
    int dataset[max_size];

    srand(time(0));

    // Generate random values
    for (int i = 0; i < max_size; i++) {
        dataset[i] = rand() % 100; // Keep numbers small for easier testing
    }

    // Show dataset
    std::cout << "Dataset: \n";
    for (int i = 0; i < max_size; i++) {
        std::cout << dataset[i] << " ";
    }
    std::cout << "\n\n";

    // Search for a number
    int item;
    std::cout << "Enter number to search: ";
    std::cin >> item;

    int result = linearSearch(dataset, max_size, item);

    if (result != -1) {
        std::cout << "Searching is successful. Item found at index " << result << ".\n";
    } else {
        std::cout << "Searching is unsuccessful. Item not found.\n";
    }

    return 0;
}
```

Header File:

```cpp
// searching.h
#ifndef SEARCHING_H
#define SEARCHING_H

int linearSearch(int dataset[], int size, int item);


// searching.cpp
#include "searching.h"

int linearSearch(int dataset[], int size, int item) {
    for (int i = 0; i < size; i++) {
        if (dataset[i] == item) {
            return i;  // Item found, return the index
        }
    }
    return -1;  // Item not found
}



#endif
```

Output:

```
C:\Users\Joshua\Documents\searching.exe

Dataset:
23 22 27 7 90 69 41 15 87 61 81 86 78 46 2 78 47 20 13 93 28 76 96 76 20 33 19 19 89 35 70 47 18 67 78 7 80 29 67 58 34
69 20 63 47 40 69 15 31 15

Enter number to search: 6
Searching is unsuccessful. Item not found.


--------------------------------
Process exited after 6.293 seconds with return value 0
Press any key to continue . . .
```

Explanation:
Linear search on array checks each element on the data set until the target value is reached.

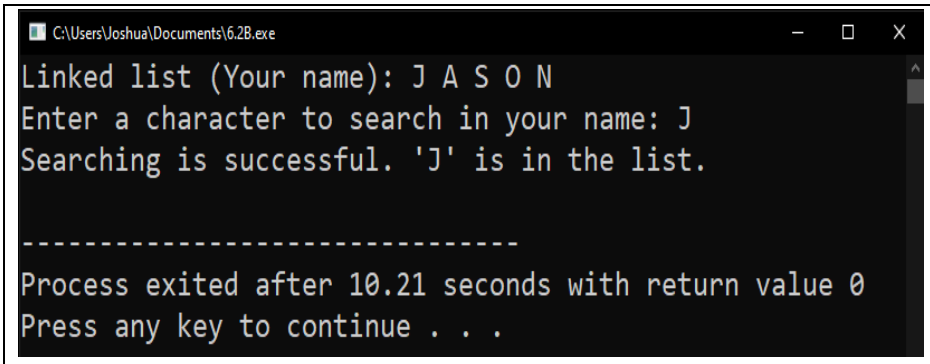
6.2 B:
Main CPP:

```cpp
#include <iostream>
#include "linkedlist.h"

int main() {
    Node<char>* name1 = new_node('J');
    Node<char>* name2 = new_node('A');
    Node<char>* name3 = new_node('S');
    Node<char>* name4 = new_node('O');
    Node<char>* name5 = new_node('N');

    name1->next = name2;
    name2->next = name3;
    name3->next = name4;
    name4->next = name5;
    name5->next = nullptr;

    std::cout << "Linked list (Your name): ";
    Node<char>* temp = name1;
    while (temp != nullptr) {
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    std::cout << "\n";

    char findChar;
    std::cout << "Enter a character to search in your name: ";
    std::cin >> findChar;

    if (linearS(name1, findChar)) {
        std::cout << "Searching is successful. '" << findChar << "' is in the list.\n";
    } else {
        std::cout << "Searching is unsuccessful. '" << findChar << "' not found.\n";
    }

    return 0;
}

```

Header File:

```cpp
#ifndef LINKEDLIST_H
#define LINKEDLIST_H

#include <iostream>

template <typename T>
struct Node {
    T data;
    Node* next;
};

template <typename T>
Node<T>* new_node(T data) {
    Node<T>* newNode = new Node<T>();
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

template <typename T>
bool linearS(Node<T>* head, T value) {
    Node<T>* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) {
            return true;  // The Value is found
        }
        temp = temp->next;
    }
    return false;  // the Value not found
}

#endif

```

Output:



Explanation:

Linear search is also the same when it comes to the process but here, it traverses node by node on the array.

6.3 A:

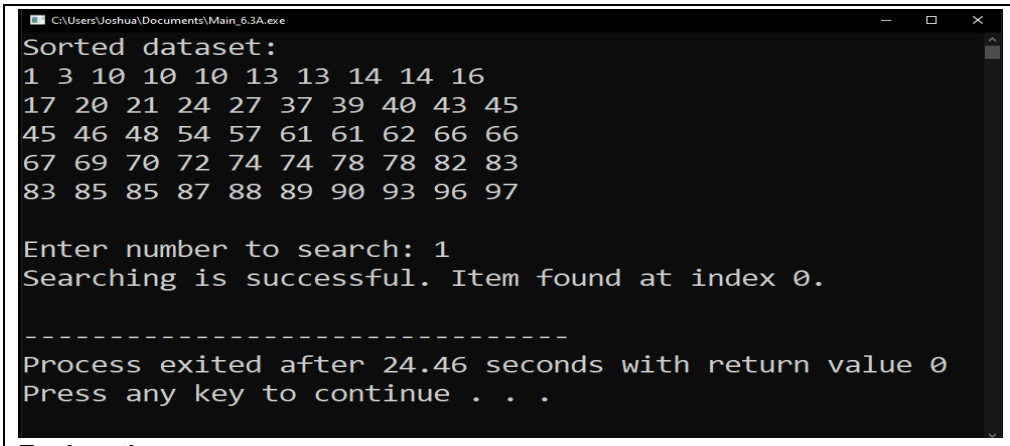
Main CPP:

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include "searching_6.3A.h"

const int max_size = 50;

int main() {
    int dataset[max_size];
    srand(time(0));

    for (int i = 0; i < max_size; i++) {
        dataset[i] = rand() % 100;
    }

    std::sort(dataset, dataset + max_size);

    std::cout << "Sorted dataset: \n";
    for (int i = 0; i < max_size; i++) {
        std::cout << dataset[i] << " ";
        if ((i + 1) % 10 == 0) std::cout << "\n";
    }
    std::cout << "\n";

    int item;
    std::cout << "Enter number to search: ";
    std::cin >> item;

    int result = binarySearch(dataset, max_size, item);

    if (result != -1) {
        std::cout << "Searching is successful. Item found at index " << result << ".\n";
    } else {
        std::cout << "Searching is unsuccessful. Item not found.\n";
    }

    return 0;
}
```

Header File:

```cpp
#ifndef SEARCHING_H
#define SEARCHING_H

int binarySearch(int arr[], int size, int item) {
    int low = 0;
    int high = size - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == item) {
            return mid;
        }

        if (arr[mid] < item) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return -1; // Item not found
}

#endif
```

Output:



Explanation:

Now for the binary search in an array, it just halves the search interval repeatedly until the target value is found. Be mindful that this only works on sorted datasets.

6.3 B:

Main CPP:

```cpp
#include <iostream>
#include "nodes.h"
#include "searching_6.3B.h"

int main() {
    Node<int>* head = new_node(10);
    Node<int>* node2 = new_node(20);
    Node<int>* node3 = new_node(30);
    Node<int>* node4 = new_node(40);
    Node<int>* node5 = new_node(50);

    head->next = node2;
    node2->next = node3;
    node3->next = node4;
    node4->next = node5;

    std::cout << "Linked list content: ";
    Node<int>* temp = head;
    while (temp != NULL) {
        std::cout << temp->data << " ";
        temp = temp->next;
    }
    std::cout << "\n";

    int key;
    std::cout << "Enter number to search: ";
    std::cin >> key;

    Node<int>* result = binarySearchLinkedList(head, key);

    if (result != NULL) {
        std::cout << "Searching is successful. Item " << key << " found in linked list.\n";
    } else {
        std::cout << "Searching is unsuccessful. Item " << key << " not found in linked list.\n";
    }

    return 0;
}
```

searching_6.3B.h

```cpp
#ifndef SEARCHING_H
#define SEARCHING_H

template <typename T>
Node<T>* binarySearchLinkedList(Node<T>* head, T key) {
    Node<T>* low = head;
    Node<T>* high = nullptr;
    Node<T>* mid = nullptr;

    while (low != nullptr) {
        mid = low;
        high = low->next;

        // Check if mid is the correct match
        if (mid != nullptr && mid->data == key) {
            return mid;
        }

        if (mid->data < key) {
            low = mid->next;
        } else {
            high = mid;
        }
    }

    return nullptr; // Item not found
}

#endif
```

nodes.h

```cpp
#ifndef NODES_H
#define NODES_H

template <typename T>
struct Node {
    T data;
    Node* next;

    Node(T data) : data(data), next(nullptr) {}
};

template <typename T>
Node<T>* new_node(T data) {
    return new Node<T>(data);
}

#endif
```

Output:

```
C:\Users\Joshua\Documents\Main_6.3B.exe
Linked list content: 10 20 30 40 50
Enter number to search: 20
Searching is successful. Item 20 found in linked list.

--------------------------------
Process exited after 9.159 seconds with return value 0
Press any key to continue . . .
```

Explanation:

Now binary search for linked list, it still works correctly but it is much slower here because it needs to traverse nodes to find the middle.

7. Supplementary Activity:

Main CPP:

[*] Supplementary_problem#1.cpp

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

struct Node {
    int data;
    Node* next;
};

Node* createList(int arr[], int size) {
    if (size == 0) return nullptr;
    Node* head = new Node{arr[0], nullptr};
    Node* current = head;
    for (int i = 1; i < size; i++) {
        current->next = new Node{arr[i], nullptr};
        current = current->next;
    }
    return head;
}

int searchArray(int arr[], int size, int key, int &count) {
    count = 0;
    for (int i = 0; i < size; i++) {
        count++;
        if (arr[i] == key) return i;
    }
    return -1;
}

int searchList(Node* head, int key, int &count) {
    count = 0;
    int index = 0;
    Node* current = head;
    while (current) {
        count++;
        if (current->data == key) return index;
        current = current->next;
        index++;
    }
    return -1;
}
}

int main() {
    int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key = 18;
    int comparisonsArray, comparisonsList;

    Node* head = createList(arr, size);

    int posArray = searchArray(arr, size, key, comparisonsArray);
    int posList = searchList(head, key, comparisonsList);

    if (posArray != -1)
        cout << "Array: Found " << key << " at index " << posArray << " after " << comparisonsArray << " comparisons.\n";
    else
        cout << "Array: " << key << " not found after " << comparisonsArray << " comparisons.\n";

    if (posList != -1)
        cout << "Linked List: Found " << key << " at node " << posList << " after " << comparisonsList << " comparisons.\n";
    else
        cout << "Linked List: " << key << " not found after " << comparisonsList << " comparisons.\n";

    // Clean up linked list
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }

    return 0;
}
```

Output:

```
C:\Users\Joshua\Documents\Supplementary_problem#1.exe
Array: Found 18 at index 1 after 2 comparisons.
Linked List: Found 18 at node 1 after 2 comparisons.


--------------------------------
Process exited after 0.08246 seconds with return value 0
Press any key to continue . . .
```

Explanation:

We can learn more about searching methods in arrays and linked lists by finding a number (the key) and counting comparisons in each case.

Supplementary_Act#2

Main CPP:

Supplementary_problem#1.cpp | Supplementary_problem#2.cpp

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

struct Node {
    int data;
    Node* next;
};

Node* createList(int arr[], int size) {
    if (size == 0) return nullptr;
    Node* head = new Node{arr[0], nullptr};
    Node* current = head;
    for (int i = 1; i < size; i++) {
        current->next = new Node{arr[i], nullptr};
        current = current->next;
    }
    return head;
}

// Count occurrences in array
int countOccurrencesArray(int arr[], int size, int key, int &comparisons) {
    comparisons = 0;
    int count = 0;
    for (int i = 0; i < size; i++) {
        comparisons++;
        if (arr[i] == key) count++;
    }
    return count;
}

// Count occurrences in linked list
int countOccurrencesList(Node* head, int key, int &comparisons) {
    comparisons = 0;
    int count = 0;
    Node* current = head;
    while (current) {
        comparisons++;
        if (current->data == key) count++;
        current = current->next;
    }
    return count;
}
```

```
}

int main() {
    int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key = 18;
    int comparisonsArray, comparisonsList;

    Node* head = createList(arr, size);

    int countArr = countOccurrencesArray(arr, size, key, comparisonsArray);
    int countList = countOccurrencesList(head, key, comparisonsList);

    cout << "Array: Key " << key << " occurs " << countArr << " times with " << comparisonsArray << " comparisons.\n";
    cout << "Linked List: Key " << key << " occurs " << countList << " times with " << comparisonsList << " comparisons.\n";

    // Clean up linked list
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }

    return 0;
}
```

Output:

```
C:\Users\Joshua\Documents\Supplementary_problem#2.exe
Array: Key 18 occurs 2 times with 10 comparisons.
Linked List: Key 18 occurs 2 times with 10 comparisons.

--------------------------------
Process exited after 0.08917 seconds with return value 0
Press any key to continue . . .
```

Explanation:

We can learn more about searching methods in arrays and linked lists by finding a number (the key) and counting comparisons in each case.

Supplementary_Act#3:

Main CPP:

Supplementary_problem#1.cpp | Supplementary_problem#2.cpp | Supplementary_problem#3.cpp

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

int binarySearch(int arr[], int size, int key) {
    int left = 0;
    int right = size - 1;

    while (left <= right) {
        int mid = (left + right) / 2;
        cout << "Check index " << mid << " with value " << arr[mid] << endl;

        if (arr[mid] == key) {
            return mid;  // Found the key
        }
        else if (arr[mid] < key) {
            cout << "Value is smaller than " << key << ", look right side" << endl;
            left = mid + 1;
        }
        else {
            cout << "Value is bigger than " << key << ", look left side" << endl;
            right = mid - 1;
        }
    }

    return -1;  // Key not found
}

int main() {
    int sortedList[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
    int size = 10;
    int key = 8;

    cout << "Start searching for " << key << " in the list." << endl;

    int result = binarySearch(sortedList, size, key);

    if (result != -1) {
        cout << "Found " << key << " at index " << result << "!" << endl;
    }
    else {
        cout << key << " is not in the list." << endl;
    }

    return 0;
}
```

Output:

```
C:\Users\Joshua\Documents\Supplementary_problem#3.exe
Start searching for 8 in the list.
Check index 4 with value 11
Value is bigger than 8, look left side
Check index 1 with value 5
Value is smaller than 8, look right side
Check index 2 with value 6
Value is smaller than 8, look right side
Check index 3 with value 8
Found 8 at index 3!

--------------------------------
Process exited after 0.08346 seconds with return value 0
Press any key to continue . . .
```

Explanation:

This program implements a binary search to locate a number within a sorted list and displays the work done at each step of the search.

Supplementary_Act#4:

Main CPP:

Supplementary_problem#1.cpp | Supplementary_problem#2.cpp | Supplementary_problem#3.cpp | Supplementary_problem#4.cpp

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

int binarySearch(int arr[], int size, int key) {
    int left = 0;
    int right = size - 1;

    while (left <= right) {
        int mid = (left + right) / 2;
        cout << "Check index " << mid << " with value " << arr[mid] << endl;

        if (arr[mid] == key) {
            return mid;
        }
        else if (arr[mid] < key) {
            cout << "Value is smaller than " << key << ", look right side" << endl;
            left = mid + 1;
        }
        else {
            cout << "Value is bigger than " << key << ", look left side" << endl;
            right = mid - 1;
        }
    }

    return -1;
}

int main() {
    int sortedList[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
    int size = 10;
    int key = 8;

    cout << "Start searching for " << key << " in the list." << endl;

    int result = binarySearch(sortedList, size, key);

    if (result != -1) {
        cout << "Found " << key << " at index " << result << "!" << endl;
    }
    else {
        cout << key << " is not in the list." << endl;
    }

    return 0;
}
```

Output:

```
C:\Users\Joshua\Documents\Supplementary_problem#4.exe
Start searching for 8 in the list.
Check index 4 with value 11
Value is bigger than 8, look left side
Check index 1 with value 5
Value is smaller than 8, look right side
Check index 2 with value 6
Value is smaller than 8, look right side
Check index 3 with value 8
Found 8 at index 3!

--------------------------------
Process exited after 0.1021 seconds with return value 0
Press any key to continue . . .
```

Explanation:

This program implements a binary search to find a specified number in a sorted list, and in addition to the search result, it also displays the steps of the search process.

8. Conclusion:

- **During that activity I learned in C++ how the line and binary search really work on a computer, I mean I know that linear searches the elements one by one but binary checks/looks for the target by splitting up what it has until finding it. All in all this activity will enable me to see how these searching algorithms work on a clear and practical way, as for my programming I also have to develop better organization in storing it as well as reducing my errors while coding.**

9. Assessment Rubric