

Activity No. 6	
SEARCHING TECHNIQUES	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed:
Section: CPE21S4	Date Submitted:
Name(s): Magistrado, Aira Pauleen M.	Instructor: Maria Rizette Sayo

6. Output

Screenshot	<div> <div> <div>main.cpp</div> <div> <pre> 1 #include <iostream> 2 #include <cstdlib> // For generating random integers 3 #include <time.h> // For seeding the random number generator 4 5 const int max_size = 50; 6 7 int main() { 8 // Generate random values 9 int dataset[max_size]; 10 srand(time(0)); // Seed the random number generator with the current time 11 12 for(int i = 0; i < max_size; i++) { 13 dataset[i] = rand(); // Generate random integers 14 } 15 16 // Show the dataset content 17 for(int i = 0; i < max_size; i++) { 18 std::cout << dataset[i] << " "; 19 } 20 std::cout << std::endl; 21 22 return 0; 23 } 24 </pre> </div> </div> <div> <div>C:\Users\Aira\Downloads\sea X + v</div> <div> <div>23562 21998 19818 10351 16419 30510 11851 12330 16669 26114 24005 14872 13688 16575 23401 30684 14176 27562 8696 17022 8</div> <div>138 28603 6939 3926 14272 13984 31572 11308 5113 18437 6709 1693 20141 27667 6864 704 5031 22326 30830 9482 12956 17188</div> <div>19332 4975 11195 6427 23266 12746 27269 4457</div> </div> <div> <div>-----</div> <div>Process exited after 0.1161 seconds with return value 0</div> <div>Press any key to continue . . . </div> </div> </div> </div>
Observations	<p>The code generates a list of random numbers which is limited to 50. The code will output different numbers each time it is run because of the random number generation <code>srand(time(0))</code> ;</p>

Table 6-1. Data Generated and Observations

Screenshot	<div> <div>C/C++</div> <div> <pre> #include <iostream> #include <cstdlib> #include <ctime> #include "searching.h" const int max_size = 50; </pre> </div> </div>
------------	---

```

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

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

    std::cout << "50 Dataset: ";
    for(int i = 0; i < max_size; i++) {
        std::cout << dataset[i] << " ";
    }
    std::cout << std::endl;

    int item;
    std::cout << "Number to search for: ";
    std::cin >> item;

    linearSearch(dataset, max_size, item);

    return 0;
}

```

```

C/C++
#include <iostream>
using namespace std;

void linearSearch(int data[], int n, int item) {
    for (int i = 0; i < n; i++) {
        if (data[i] == item) {
            cout << "Searching is successful: Item " << item << " found
at index " << i << "." << endl;
            return;
        }
    }
    cout << "Searching is Unsuccessful: Item " << item << " not found."
<< endl;
}

```

```

Dataset contents: 5551 31870 7657 28242 31885 15146 19545 22246 19810 30260 10977 17311 18563 12846 6935 460 28623 27660
24181 25615 17974 27281 29182 16235 10360 17415 19748 16029 9007 27699 2026 32688 27152 24200 11334 12236 21614 14491 1
4025 6479 767 22743 2689 22558 4835 21874 23583 10302 482 24479
Enter an item to search for: 482
Searching is successful: Item 482 found at index 48.

```

Observation

The code generates 50 random numbers and when it is displayed it asks the users to enter a number to search for. If the number was found it will print Searching is successful along with the index where the number was found. When the number is not found it will print Search is unsuccessful.

Table 6-2a. Linear Search for Arrays

Screenshot

```
C/C++
#include <iostream>
using namespace std;

template <typename T>
struct Node {
    T data;
    Node<T>* 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>
void linearLS(Node<T>* head, T dataFind) {
    Node<T>* current = head;
    int index = 0;
    while (current != nullptr) {
        if (current->data == dataFind) {
            cout << "Searching is successful: Character '" << dataFind
<< "' found at index " << index << "." << endl;
            return;
        }
        current = current->next;
        index++;
    }
    cout << "Searching is Unsuccessful: Character '" << dataFind << "'
not found." << endl;
}

int main() {
    Node<char>* name1 = new_node('A');
    Node<char>* name2 = new_node('i');
    Node<char>* name3 = new_node('r');
    Node<char>* name4 = new_node('a');

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

    char dataFind;
    cout << "Enter a character to search in the linked list: ";
    cin >> dataFind;

    linearLS(name1, dataFind);

    return 0;
}
```

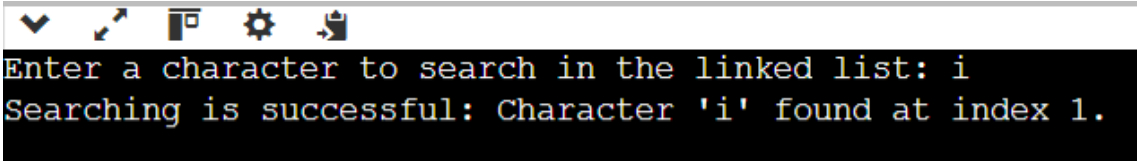
	
Observation	<p>The code creates a linked list with the letters on “Aira”. It will then ask the user which character of the letter to search for. It will perform a linear search to find the letter then if the letter is found it will print the Search was successful and at what index it was found. If it is not found it will output Search is unsuccessful.</p>

Table 6-2b. Linear Search for Linked List

Screenshot	<pre> C/C++ #include <iostream> #include <stdlib> #include <ctime> #include <algorithm> #include "searching.h" const int max_size = 50; int main() { int dataset[max_size]; srand(static_cast<unsigned int>(time(0))); for (int i = 0; i < max_size; i++) { dataset[i] = rand() % 100; } std::cout << "Unsorted dataset:" << std::endl; for (int i = 0; i < max_size; i++) { std::cout << dataset[i] << " "; } std::cout << std::endl; std::sort(dataset, dataset + max_size); std::cout << "Sorted dataset:" << std::endl; for (int i = 0; i < max_size; i++) { std::cout << dataset[i] << " "; } std::cout << std::endl; int target; std::cout << "Enter an integer to search for: "; std::cin >> target; int result = binarySearch(dataset, max_size, target); if (result != -1) { std::cout << "Search element is found at index: " << result << "!" << std::endl; } else { std::cout << "Search element is not found." << std::endl; } } </pre>
------------	--

```

        return 0;
    }

```

```

C/C++
#include <iostream>
using namespace std;

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

    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid;
        }
        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return -1;
}

```

```

Unsorted dataset:
37 6 64 75 61 32 89 4 41 86 90 84 60 88 77 77 51 57 22 68 89 90 28 41 54 31 14 22 39 59 79 49 77 73 65 47 90 84 74 34 48
24 3 29 47 41 8 58 34 27
Sorted dataset:
3 4 6 8 14 22 22 24 27 28 29 31 32 34 34 37 39 41 41 41 47 47 48 49 51 54 57 58 59 60 61 64 65 68 73 74 75 77 77 77 79 8
4 84 86 88 89 89 90 90 90
Enter an integer to search for: 6
Search element is found at index: 2!

```

Observation

The code generates 50 random numbers then it will output and sort them. It will ask the user for the number that they are searching for and it will look for it through binary search on the sorted data. If the number was found it will display search element is found and the index and if it is not found it will display search element not found.

Table 6-3a. Binary Search for Arrays

Screenshot

```

C/C++
#include <iostream>
using namespace std;

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

```

```

};

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

template <typename T>
void displayList(Node<T>* head) {
    Node<T>* currNode = head;
    while (currNode != nullptr) {
        cout << currNode->data << " ";
        currNode = currNode->next;
    }
    cout << endl;
}

template <typename T>
Node<T>* getMiddle(Node<T>* start, Node<T>* end) {
    if (start == nullptr) return nullptr;

    Node<T>* slow = start;
    Node<T>* fast = start;

    while (fast != end && fast->next != end) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}

template <typename T>
Node<T>* binarySearch(Node<T>* head, T target) {
    Node<T>* start = head;
    Node<T>* end = nullptr;

    while (start != end) {
        Node<T>* mid = getMiddle(start, end);

        if (mid->data == target) {
            return mid; // Return the found node
        }
        else if (mid->data > target) {
            end = mid;
        }
        else {
            start = mid->next;
        }
    }
    return nullptr;
}

```

```

int main() {
    char choice = 'y';
    int count = 1;
    int newData;
    Node<int>* temp, *head = nullptr, *node = nullptr;

    while (choice == 'y') {
        cout << "Enter data: ";
        cin >> newData;

        if (count == 1) {
            head = new_node(newData);
            cout << "Successfully added " << head->data << " to the
list.\n";
            count++;
        } else {
            node = new_node(newData);
            temp = head;

            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = node;
            cout << "Successfully added " << node->data << " to the
list.\n";
            count++;
        }

        cout << "Continue? (y/n) ";
        cin >> choice;
    }

    cout << "Linked List: ";
    displayList(head);

    int target;
    cout << "Enter target to search: ";
    cin >> target;

    Node<int>* result = binarySearch(head, target);

    if (result != nullptr) {
        cout << "Search element " << target << " is found in the linked
list!" << endl;
    } else {
        cout << "Search element " << target << " is not found in the
linked list." << endl;
    }

    return 0;
}

```

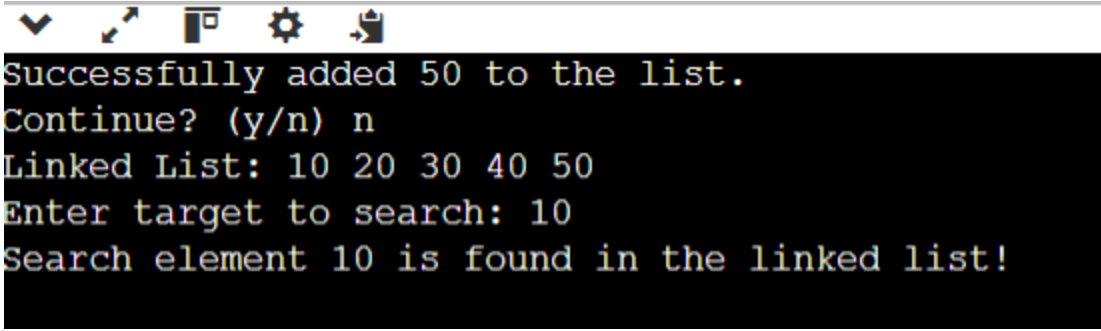
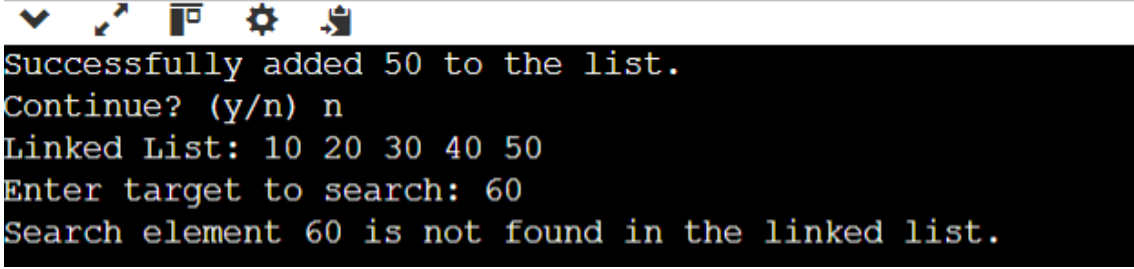
Output	 
Observation	The code will ask the user for number input then it will create a sorted list and will ask the user for the number that they are looking for. If the number is on the list it will display search element is found if not Search element is not found.

Table 6-3b. Binary Search for Linked List

7. Supplementary Activity
Problem 1:


```

main.cpp
1  #include <iostream>
2  using namespace std;
3
4  int linearSearch(int arr[], int size, int target, int& comparisons) {
5      comparisons = 0;
6      for (int i = 0; i < size; i++) {
7          comparisons++;
8          if (arr[i] == target) {
9              return i;
10         }
11     }
12     return -1;
13 }
14
15 int main() {
16     int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
17     int size = sizeof(arr) / sizeof(arr[0]);
18     int target = 18;
19     int comparisons;
20
21     int index = linearSearch(arr, size, target, comparisons);
22
23     if (index != -1) {
24         cout << "Element " << target << " found at index " << index << " after " << comparisons << " comparisons." << endl;
25     } else {
26         cout << "Element " << target << " not found after " << comparisons << " comparisons." << endl;
27     }
28 }

```

input

Element 18 found at index 1 after 2 comparisons.

Problem 2:

```

main.cpp
1  #include <iostream>
2  using namespace std;
3
4  int linearSearch(int arr[], int size, int target, int& comparisons) {
5      comparisons = 0;
6      int count = 0;
7      for (int i = 0; i < size; i++) {
8          comparisons++;
9          if (arr[i] == target) {
10             count++;
11         }
12     }
13     return count;
14 }
15
16 int main() {
17     int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
18     int size = sizeof(arr) / sizeof(arr[0]);
19     int target = 18;
20     int comparisons;
21
22     int count = linearSearch(arr, size, target, comparisons);
23
24     if (count > 0) {
25         cout << "Element " << target << " found " << count << " times after " << comparisons << " comparisons." << endl;
26     } else {
27         cout << "Element " << target << " not found after " << comparisons << " comparisons." << endl;
28     }
29
30     return 0;
31 }
32

```

input

Element 18 found 2 times after 10 comparisons.

Problem 3:

Initial State:

| 3 | 5 | 6 | 8 | 11 | 12 | 14 | 15 | 17 | 18 |

First Last

Midpoint: $M = (First + Last) / 2 = (0 + 9) / 2 = 4$ $M = 4$: Value = 11

8 < 11 so move left

(8) is less than the middle value (11).

| 3 | 5 | 6 | 8 |

First Last

Midpoint: $M = (First + Last) / 2 = (0 + 3) / 2 = 1$ $M = 1$: Value = 5

8 > 5 so move right

(8) is greater than the middle value (5).

| 6 | 8 |

First Last

Midpoint: $M = (First + Last) / 2 = (2 + 3) / 2 = 2$ $M = 2$: Value = 6

8 > 6 so move right

(8) is greater than the middle value (6).

| 8 |

First Last

Midpoint: $M = (First + Last) / 2 = (3 + 3) / 2 = 3$ $M = 3$: Value = 8

8 = 8

Problem 4:

main.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int binarySearch(int arr[], int left, int right, int target) {
5     if (left > right) {
6         return -1;
7     }
8     int mid = left + (right - left) / 2;
9     if (arr[mid] == target) {
10         return mid;
11     } else if (arr[mid] > target) {
12         return binarySearch(arr, left, mid - 1, target);
13     } else {
14         return binarySearch(arr, mid + 1, right, target);
15     }
16 }
17
18 int main() {
19     int arr[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
20     int size = sizeof(arr) / sizeof(arr[0]);
21     int target = 8;
22
23     int result = binarySearch(arr, 0, size - 1, target);
24
25     if (result != -1) {
26         cout << "The key " << target << " is found at index " << result << "." << endl;
27     } else {
28         cout << "The key " << target << " is not found in the array." << endl;
29     }
30
31     return 0;
32 }
33
```

input

The key 8 is found at index 3.

8. Conclusion

In conclusion, the lab activity taught me the difference of linear search and binary. The linearSearch function searches a list for a given number. It searches through the lists, number by number, comparing each one to the number it's looking for. When it detects a match, it stops and notifies you of its location. It indicates that the item wasn't there if it gets to the end of the list without discovering the number. When using binary search, the first number in a sorted list to be examined is the middle number. If the number being searched for is smaller than the middle number, it must be in the first half of the list. If it discovers that the number it's searching for is greater, it must be in the second half of the list. It then goes through this process again, focusing on the half of the list where the number might be until it finds the number. Overall, I think I did great and I understood the difference between the two.

9. Assessment Rubric