Activity No. 6.1	
Searching Techniques	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:15/09/2025**
Section: CPE21S4	**Date Submitted:16/09/2025**
Name(s): Crishen Luper S. Pulgado	**Instructor: Sir Jimlord Quejado**

6. Output

Table 6-1

HOA6.1.cpp

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

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();
    }

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

    return 0;
}

```

```
Z:\Crishen\cpp\HOA6.1.exe
24761 28139 18546 20216 25542 11310 32431 880 16910 6783 9085 9214 6332 15791 21374 6540 12446 28837 7688 5610 5335 2874
4 29048 7885 468 24224 8649 18967 16513 311 15717 3928 16828 16070 9274 21141 3594 8205 5393 3145 23944 22531 9499 12970
 27685 5935 21965 18477 24079 11862
--------------------------------
Process exited after 0.08083 seconds with return value 0
Press any key to continue . . .
```

Observation:

I observe that the code includes a “cstdlib” which allows to access the rand() function and srand() function. This allows to generate random numbers and sets the “seed” starting point for those random numbers. The time allows to access the time() function. The code creates an array with 50 as its max capacity and we use the srand(time(0)) to get different random numbers each time the code has run.

Table 6-2a

HOA6.1.cpp | node.h | [*] searching.h

```
#ifndef SEARCHING_H
#define SEARCHING_H

#include <iostream>
int linearSearch(int data[], int n, int item) {
    int i = 0;

    while (i < n) {
        if (item == data[i]) {
            std::cout << "Searching is successful. Item found at index " << i << std::endl;
            return i;
        }
        i++;
    }

    std::cout << "Searching is unsuccessful. Item not found." << std::endl;
    return -1;
}

#endif

```

```
Z:\Crishen\cpp\HOA6.1.exe
32491 8881 32320 27295 27980 32536 26802 18837 7620 3299 19439 15168 13019 23
03 4732 15863 30828 28235 18726 16030 22991 11359 17265 8297 30634 26975 1024
949 6728 16314 2274 6181 9369 8503 4958 23321
enter a number to search: 30828
Searching is successful. Item found at index 23

--------------------------------
Process exited after 13.01 seconds with return value 0
Press any key to continue . . .
```

HOA6.1.cpp | node.h | [*] searching.h

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "searching.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();
    }

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

    std::cout << std::endl;
    int item;
    std::cout << "enter a number to search: ";
    std::cin >> item;

    linearSearch(dataset, max_size, item);

    return 0;
}

```

Observation:

Here, I created a function linear search that search the data[] array and runs if the item is equal to the element that the user is searching for. If it does not find the specific element, it will print that the item is not found and returns -1 to end.

In the main program, I've included the searching.h file and I ask the user to input a specific number to find and calls the linearSearch function to find the number.

Table 6-2b

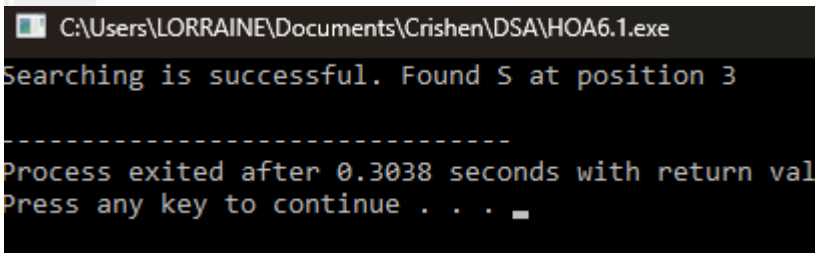
HOA6.1.cpp searching.h node.h

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "searching.h"
#include "node.h"

int main() {
    //create linked list for linear search
    Node<char> *name1 = new_node('C');
    Node<char> *name2 = new_node('R');
    Node<char> *name3 = new_node('I');
    Node<char> *name4 = new_node('S');
    Node<char> *name5 = new_node('H');
    Node<char> *name6 = new_node('E');
    Node<char> *name7 = new_node('N');

    //linked them together
    name1->next = name2;
    name2->next = name3;
    name3->next = name4;
    name4->next = name5;
    name5->next = name6;
    name6->next = name7;
    name7->next = NULL;

    linearLS(name1, 'S');

    return 0;
}
```

```cpp
#ifndef SEARCHING_H
#define SEARCHING_H

#include <iostream>
#include "node.h"

template <typename T>
void linearLS(Node<T> *head, T dataFind) {
    Node<T>* current = head;
    int position = 0;


    while (current != NULL) {
        if (current->data == dataFind) {
            std::cout << "Searching is successful. F
                      << dataFind
                      << " at position " << position
                      << std::endl;
            return;
        }
        current = current->next;
        position++;
    }

    std::cout << "Searching is unsuccessful. "
              << dataFind
              << " not found in the list."
              << std::endl;
}

#endif
```

HOA6.1.cpp | searching.h | node.h

```
1   #ifndef NODES_H
2   #define NODES_H
3   template <typename T>
4   class Node{
5       public:
6           T data;
7           Node *next;
8       };
9   template <typename T>
10  Node<T> *new_node(T newData){
11      Node<T> *newNode = new Node<T>;
12      newNode->data = newData;
13      newNode->next = NULL;
14      return newNode;
15  }
16  #endif
```



Observation:

In the node.h file, it is where the nodes will be created and has the pointer next to it. For the searching.h file, I include the node.h file so that the compiler knows what a node is. I created the function linearLS and the parameters contains the head, which it will start, and dataFind is what we are looking for. The current is a pointer that goes throughout the list one node at a time and the position keeps track where you are. The while loop will keep running until there are no more nodes. If the current node matches the data that we are finding, it prints success and returns to stop searching. Current moves and the position++ increase the position counter. Else, it will print the “data not found”. For the result, I use CRISHEN and find the letter S which is at position 3 because we started at 0, which is the C.

Table 6-3a

```
template <typename T>
int binarySearch(T arr[], int n, T key) {
    int low = 0;
    int up = n - 1;

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

        if (key == arr[mid]) {
            std::cout << "Search element " << key
                      << " is found at index " << mid << std::endl;
            return mid;
        }
        else if (key < arr[mid]) {
            up = mid - 1;
        }
        else {
            low = mid + 1;
        }
    }

    std::cout << "Search element " << key << " is not found." << std::endl;
    return -1;
}
#endif
```

```
    linearLS(name1, 'S');

    int arr[] = {1, 3, 5, 7, 9, 11};
    int size = sizeof(arr) / sizeof(arr[0]);

    binarySearch(arr, size, 7);
    binarySearch(arr, size, 8);

    return 0;

    return 0;
}
```

```
Search element 7 is found at index 3
Search element 8 is not found.
```

Observation:

I've made the similar to the other code as template <typename t> so that it can input any datatypes. In the binarySearch, the parameters contains arr[], which is the sorted array that we are searching in, n as the number of elements in the array, key is the value that we wanted to find, low starts at first index 0, and the up starts at the last index which is n-1. The loop will run as long as the up is greater than or equal to low and mid is the middle index between low and up. If the middle element is equals to key, it will print found index. If the key is smaller than the middle element, we will search for the left of the array so we move up to mid-1. If the key is greater than the middle element we ignore the left half so we move low to mid+1. If the loop ends, it means the data that is searching has not been found. So in step 1, low = 0, up = 5 which mid = 2 which is arr[2] = 5, 7>5. Then the step 2, low = 3, up = 5, mid = 4 which is arr[4] =9, 7<9. Lastly, step 3 is low =3, up =3 which is mid=3 which arr[3] =7 and that is found at index 3.

Table 3-2b

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

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

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

template <typename T>
Node<T>* binarySearchLL(Node<T>* head, T key) {
    Node<T>* start = head;
    Node<T>* end = NULL;

    do {
        Node<T>* mid = getMiddle(start, end);
        if (mid == NULL) return NULL;

        if (mid->data == key) {
            return mid;
        }
        else if (mid->data < key) {
            start = mid->next;
        }
        else {
            end = mid;
        }
    } while (end == NULL || end != start);

    return NULL;
}
#endif
```

```
    char choice = 'y';
    int count = 1;
    int newData;
    Node<int> *temp, *head, *node;

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

        if(count == 1){
            head = new_node(newData);
            std::cout << "Succesfully added" << head->data << "to the list.\n";
            count++;

        } else if(count==2){
            node = new_node(newData);
            head->next = node;
            node->next = NULL;
            std::cout << "Succesfully added" << node->data << "to the list. \n";
            count++;

        }else {
            temp = head;
            while(true){
                if(temp->next == NULL) break;
                temp = temp->next;
            } node = new_node(newData);
            temp->next = node;
            std::cout << "Successfully added" << node->data << "to the list. \n";
            count++;
        }
        std::cout << "Continue (y/n): ";
        std::cin>> choice;
    }
    std::cout << "Final linked list: ";
    Node<int>* currNode = head;
    while (currNode != NULL) {
        std::cout << currNode->data << " ";
        currNode = currNode->next;
    }
    std::cout << std::endl;


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

```
Enter data: 1
Succesfully added1to the list.
Continue (y/n): y
Enter data: 2
Succesfully added2to the list.
Continue (y/n): y
Enter data: 3
Successfully added3to the list.
Continue (y/n): y
Enter data: 4
Successfully added4to the list.
Continue (y/n): y
Enter data: 5
Successfully added5to the list.
Continue (y/n): y
Enter data: 6
Successfully added6to the list.
Continue (y/n): y
Enter data: 7
Successfully added7to the list.
Continue (y/n): n
Final linked list: 1 2 3 4 5 6 7
Enter value to search: 5
Found 5 in the list.
```

Observation:

In this procedure, I added the getMiddle function which finds the middle node between start and end since in linked list we cannot jump directly to the middle unlike arrays. The slow pointer moves one step at a time and the fast pointer moves two steps at a time. In the while loop, it traverses until it reaches the end. Under the while loop, it moves fast by one (fast = fast->next) and checks before moving further and returns slow. For the binarySearchLL function, it searches the value of key in a sorted linked list. The (Node<T>* start = head) is the start of the head of the list. The (Node<T>* end = NULL)is the end marker initilally set to the NULL. Inside the do while loop, the (Node<T>* mid = getMiddle(start, end)) finds the middle node in the current range. If the data == key, it has found the element. If data < key, it searches the right side of the list, so it moves start to mid->next. If mid->data > key, it searches the left side and move end to mid. The while loop stops when the search range collapses and returns null if no match.

7. Supplementary Activity

```
int sequentialSearchComparisons(int arry[], int n, int key) {
    int comparisons = 0;
    for (int i = 0; i < n; i++) {
        comparisons++;
        if (arry[i] == key) {
            return comparisons;
        }
    }
    return comparisons;
}
```

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

int main() {
    int arry[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
    int n = sizeof(arry) / sizeof(arry[0]);

    int key = 18;


    int compsArr = sequentialSearchComparisons(arry, n, key);
    std::cout << "Array: Found key " << key
              << " after " << compsArr << " comparisons.\n";


    Node<int>* head = NULL;
    Node<int>* tail = NULL;
    for (int i = 0; i < n; i++) {
        Node<int>* node = new_node(arry[i]);
        if (head == NULL) {
            head = node;
            tail = node;
        } else {
            tail->next = node;
            tail = node;
        }
    }


    int compsLL = linearLSComparisons(head, key);
    std::cout << "Linked List: Found key " << key
              << " after " << compsLL << " comparisons.\n";

    return 0;
}
```

```
C:\Users\LORRAINE\Documents\Crishen\DSA\supplementary.exe
Array: Found key 18 after 2 comparisons.
Linked List: Found key 18 after 2 comparisons.

--------------------------------
Process exited after 0.2925 seconds with return value 0
Press any key to continue . . .
```

The number of comparisons made when searching the list [15, 18, 2, 19, 18, 0, 8, 14, 19, 14] with the key 18, is dependent on the location of the key in the list. Throughout the array and the linked list method, the search process sequentially examines element after element starting at the start until one finds the key or gets to the end. As the first instance of 18 is at index 1, the algorithm will first compare the key with 15 on step 2, then the next step is 18. This implies that there are two comparisons that have to be made to perform the search of the key. This reasoning applies equally to the cases when the data is stored in an array or in a linked list because to search sequentially these algorithms move through elements sequentially.

8. Conclusion

In conclusion, this activity taught me to use array and linked list in searching algorithms. It taught me how to implement in binary and linear search using arrays and linked list. I did good in this activity and I had a little hard time for the supplementary and other parts of the procedure.

9. Assessment Rubric