

Activity No. 6	
Searching Techniques	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 10/15/2024
Section:CPE21S4	Date Submitted: 10/16/2024
Name(s): Danica T. Guariño	Instructor: Ma'am Rizette Sayo

6. Output

Screenshot	<div><div>main.cpp</div><pre>1  #include &lt;iostream&gt; 2  #include &lt;cstdlib&gt; //for generating random integers 3  #include &lt;time.h&gt; //will be used for our seeding function 4 5  int main(){ 6 7      const int max_size = 50; 8      //generate random values 9      int dataset[max_size]; 10     srand(time(0)); 11 12     for(int i = 0; i &lt; max_size; i++){ 13         dataset[i] = rand(); 14     } 15 16     //show your datasets content 17     for(int i = 0; i &lt; max_size; i++){ 18         std::cout &lt;&lt; dataset[i] &lt;&lt; " "; 19     } 20     std::cout&lt;&lt;std::endl; 21 22     return 0; 23 }</pre><div><div>C:\Users\TIPQC\OneDrive\Desktop\HAO 6.exe</div><div>567 29893 28918 24836 615 11079 9338 19612 2936 4557 15363 13296 7660 2652 9938 18767 25260 6863 4518 2957 3588 21452 47 ^ 7 1181 29679 9206 11229 22114 12693 27492 2226 2499 4114 25427 28408 21502 1145 11799 13052 17982 28952 9048 4638 7023 3 070 11386 28026 6225 3771 17757 ----- Process exited after 0.01657 seconds with return value 0 Press any key to continue . . .</div></div></div>
Observation	List of random numbers were generated up until 50.

Table 6-1. Data Generated and Observations

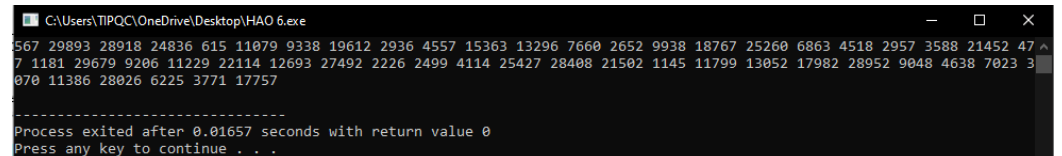
## Code

```
main.cpp  nodes.h
1
2 #ifndef NODES_H
3 #define NODES_H
4
5 template <typename T>
6 class Node {
7 public:
8     T data;
9     Node* next;
10 };
11
12 template <typename T>
13 Node<T>* new_node(T newData) {
14     Node<T>* newNode = new Node<T>;
15     newNode->data = newData;
16     newNode->next = nullptr;
17     return newNode;
18 }
19
20 #endif // NODES_H
21
```

```
main.cpp  nodes.h  searching.h
1
2 #ifndef SEARCHING_H
3 #define SEARCHING_H
4
5 #include <iostream>
6 #include "nodes.h"
7
8 template <typename T>
9 void linearSearch(Node<T>* head, T item) {
10     Node<T>* current = head;
11     int index = 0;
12
13     while (current != nullptr) {
14         if (current->data == item) {
15             std::cout << "Searching is successful at index " << index << std::endl;
16             return;
17         }
18         current = current->next;
19         index++;
20     }
21     std::cout << "Searching is unsuccessful" << std::endl;
22 }
23
24 #endif // SEARCHING_H
28
```

## Output

nodes.h



```
C:\Users\TIPQC\OneDrive\Desktop\HAO 6.exe
567 29893 28918 24836 615 11079 9338 19612 2936 4557 15363 13296 7660 2652 9938 18767 25260 6863 4518 2957 3588 21452 47^
7 1181 29679 9206 11229 22114 12693 27492 2226 2499 4114 25427 28408 21502 1145 11799 13052 17982 28952 9048 4638 7023 3
070 11386 28026 6225 3771 17757
-----
Process exited after 0.01657 seconds with return value 0
Press any key to continue . . .
```

searching.h

Observations

The code generated 50 random numbers and once it displayed it asked the users to enter some number, if the number is found it will print Searching.

Table 6-2a. Linear Search for Arrays

Code

```
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() {
```

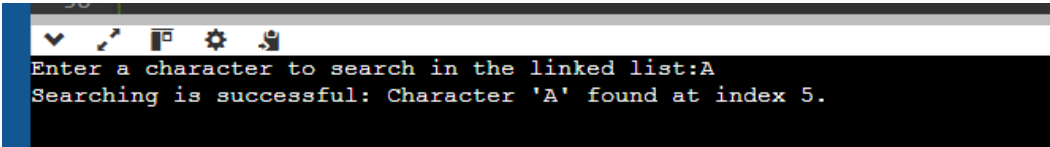
	<pre> Node&lt;char&gt;* name1 = new_node('D'); Node&lt;char&gt;* name2 = new_node('a'); Node&lt;char&gt;* name3 = new_node('n'); Node&lt;char&gt;* name4 = new_node('i'); Node&lt;char&gt;* name5 = new_node('c'); Node&lt;char&gt;* name6 = new_node('A');  name1-&gt;next = name2; name2-&gt;next = name3; name3-&gt;next = name4; name4-&gt;next = name5; name5-&gt;next = name6; name6-&gt;next = nullptr;  char dataFind; cout &lt;&lt; "Enter a character to search in the linked list:"; cin &gt;&gt; dataFind;  linearLS(name1, dataFind);  return 0; } </pre>
Output	
Observations	<p>This code generated a linked list with the letters of my name 'Danica'. It asked to enter a character to search in the linked list, once it was found, it will print the character you entered.</p>

Table 6-2b. Linear Search for Linked List

## 7. Supplementary Activity

### ILO B: Solve different problems utilizing appropriate searching techniques in C++

For each provided problem, give a screenshot of your code, the output console, and your answers to the questions.

**Problem 1.** Suppose you are doing a sequential search of the list [15, 18, 2, 19, 18, 0, 8, 14, 19, 14]. Utilizing both a linked list and an array approach to the list, use sequential search and identify how many comparisons would be necessary to find the key '18'?

```
main.cpp x +
main.cpp > f main

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  Node* newNode(int data) {
10     Node* node = new Node;
11     node->data = data;
12     node->next = nullptr;
13     return node;
14 }
15
16 Node* createLinkedList(int arr[], int size) {
17     Node* head = newNode(arr[0]);
18     Node* current = head;
19     for (int i = 1; i < size; i++) {
20         current->next = newNode(arr[i]);
21         current = current->next;
22     }
23     return head;
24 }
25
26 int sequentialSearchLinkedList(Node* head, int key) {
27     int comparisons = 0;
28     Node* current = head;
29     while (current != nullptr) {
30         comparisons++;
31         if (current->data == key) {
32             return comparisons;
33         }
34         current = current->next;
35     }
36     return comparisons;
37 }
```

```

39 int sequentialSearchArray(int arr[], int size, int key) {
40     int comparisons = 0;
41     for (int i = 0; i < size; i++) {
42         comparisons++;
43         if (arr[i] == key) {
44             return comparisons;
45         }
46     }
47     return comparisons;
48 }
49
50 int main() {
51     int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
52     int size = sizeof(arr) / sizeof(arr[0]);
53     int key = 18;
54
55     Node* head = createLinkedList(arr, size);
56     int comparisonsLinkedList = sequentialSearchLinkedList(head, key);
57     cout << "Comparisons using linked list: " << comparisonsLinkedList << endl;
58
59     int comparisonsArray = sequentialSearchArray(arr, size, key);
60     cout << "Comparisons using array: " << comparisonsArray << endl;
61
62     return 0;
63 }

```

Output:

```

v Run
Comparisons using linked list: 2
Comparisons using array: 2

```

**Problem 2.** Modify your sequential search algorithm so that it returns the count of repeating instances for a given search element 'k'. Test on the same list given in problem 1.

```

C:\main.cpp x +
C:\main.cpp > f main

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  Node* newNode(int data) {
10     Node* node = new Node;
11     node->data = data;
12     node->next = nullptr;
13     return node;
14 }
15
16
17 Node* createLinkedList(int arr[], int size) {
18     Node* head = newNode(arr[0]);
19     Node* current = head;
20     for (int i = 1; i < size; i++) {
21         current->next = newNode(arr[i]);
22         current = current->next;
23     }
24     return head;
25 }
26
27 int sequentialSearchLinkedList(Node* head, int key) {
28     int count = 0;
29     Node* current = head;
30     while (current != nullptr) {
31         if (current->data == key) {
32             count++;
33         }
34         current = current->next;
35     }
36     return count;
37 }

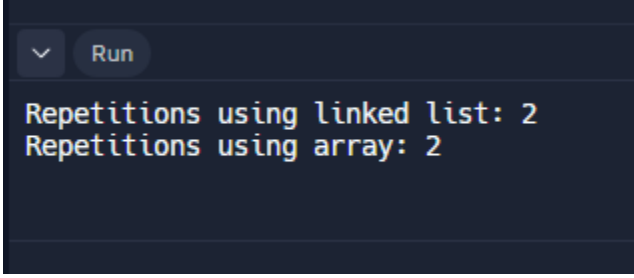
```

```

38
39 int sequentialSearchArray(int arr[], int size, int key) {
40     int count = 0;
41     for (int i = 0; i < size; i++) {
42         if (arr[i] == key) {
43             count++;
44         }
45     }
46     return count;
47 }
48
49 int main() {
50     int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
51     int size = sizeof(arr) / sizeof(arr[0]);
52     int key = 18;
53
54
55     Node* head = createLinkedList(arr, size);
56     int repetitionsLinkedList = sequentialSearchLinkedList(head, key);
57     cout << "Repetitions using linked list: " << repetitionsLinkedList << endl;
58
59
60     int repetitionsArray = sequentialSearchArray(arr, size, key);
61     cout << "Repetitions using array: " << repetitionsArray << endl;
62
63     return 0;
64 }

```

## Output:



```

  Run
Repetitions using linked list: 2
Repetitions using array: 2

```

**Problem 3.** Suppose you have the following sorted list [3, 5, 6, 8, 11, 12, 14, 15, 17, 18] and are using the binary search algorithm. If you wanted to find the key 8, draw a diagram that shows how the searching works per iteration of the algorithm. Prove that your drawing is correct by implementing the algorithm and showing a screenshot of the code and the output console.

## Diagram:

Iteration 1:

Low = 0, High = 9, Mid = 4

List: [3, 5, 6, 8, 11, 12, 14, 15, 17, 18]

^ (11 is too high, search left half)

Iteration 2:

Low = 0, High = 3, Mid = 1

List: [3, 5, 6, 8, 11, 12, 14, 15, 17, 18]

^ (5 is too low, search right half)

Iteration 3:

Low = 2, High = 3, Mid = 2

List: [3, 5, 6, 8, 11, 12, 14, 15, 17, 18]

^ (6 is too low, search right half)

Iteration 4:

Low = 3, High = 3, Mid = 3

List: [3, 5, 6, 8, 11, 12, 14, 15, 17, 18]

^ (Key found)

## Code:



```

1  #include <iostream>
2
3  int binarySearch(int arr[], int size, int key) {
4      int low = 0;
5      int high = size - 1;
6      int comparisons = 0;
7
8      while (low <= high) {
9          comparisons++;
10         int mid = low + (high - low) / 2;
11         std::cout << "Iteration " << comparisons << ": Low = " << low << ", High = " << high << ",
Mid = " << mid << "\n";
12
13         if (arr[mid] == key) {
14             std::cout << "Number of comparisons: " << comparisons << std::endl;
15             return mid; // Key found
16         }
17         if (arr[mid] < key) {
18             low = mid + 1;
19         } else {
20             high = mid - 1;
21         }
22     }
23     std::cout << "Number of comparisons: " << comparisons << std::endl;
24     return -1; // Key not found
25 }
26
27 int main() {
28     int arr[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
29     int size = sizeof(arr) / sizeof(arr[0]);
30     int key = 8;
31     int result = binarySearch(arr, size, key);
32     if (result != -1) {
33         std::cout << "Key " << key << " found at index " << result << std::endl;
34     } else {
35         std::cout << "Key " << key << " not found" << std::endl;
36     }
}

```

Output:

```

Run

Iteration 1: Low = 0, High = 9, Mid = 4
Iteration 2: Low = 0, High = 3, Mid = 1
Iteration 3: Low = 2, High = 3, Mid = 2
Iteration 4: Low = 3, High = 3, Mid = 3
Number of comparisons: 4
Key 8 found at index 3

```

**Problem 4.** Modify the binary search algorithm so that the algorithm becomes recursive. Using this new recursive binary search, implement a solution to the same problem for problem 3.

```

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

```

Output:

```

  Run
Key 8 found at index: 3

```

## 8. Conclusion

In conclusion, this activity is hard since it is confusing and also requires a lot of time to do. I find it difficult to generate the codes needed for this activity. I learned that there is a difference between linear search and binary. The algorithm for the binary search divides the list in half with each comparison using the fact that the list should be ordered for quick elimination of half possibilities at each step, and its time complexity is much faster. Though simple, linear search is

suitable for unsorted or short lists; however, the binary search excels in large and sorted lists making it the strategic and efficient choice.

## **9. Assessment Rubric**