

Activity No. 6	
Searching Techniques	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: October 14, 2024
Section: CPE21S4	Date Submitted: October 15, 2024
Name(s): Bona, Andrei Nycole So	Instructor: Prof. Maria Rizette Sayo

6. Output

Screenshot	<pre> 1 #include <iostream> 2 #include <cstdlib> 3 #include <ctime> 4 5 using namespace std; 6 7 const int max_size = 50; 8 9 int main() 10 { 11 // generate random values 12 int dataset[max_size]; 13 srand(time(0)); 14 for (int i = 0; i < max_size; i++) 15 { 16 dataset[i] = rand(); 17 } 18 19 // show your datasets content 20 for (int i = 0; i < max_size; i++) 21 { 22 cout << dataset[i] << " "; 23 } 24 25 return 0; 26 } 27 30152 19866 24916 7942 23550 24795 1321 3329 12002 5617 20442 5454 27155 16469 13329 22727 151 17983 304 8975 8724 8463 29228 3607 2 4642 15596 7838 32487 18592 31659 535 24475 18022 24306 3508 15019 24763 7581 30707 7024 10537 10336 16069 22500 28238 25006 4069 10 154 1748 3290 ----- Process exited after 0.04528 seconds with return value 0 Press any key to continue . . . </pre>
Observations	<p>This C++ code creates an array filled with random numbers and displays them. It uses essential libraries to handle input/output and generate random numbers. The array size is set to 50. To make sure the numbers are different each time you run the program, it seeds the random number generator with the current time.</p>

Table 6-1. Data Generated and Observations.

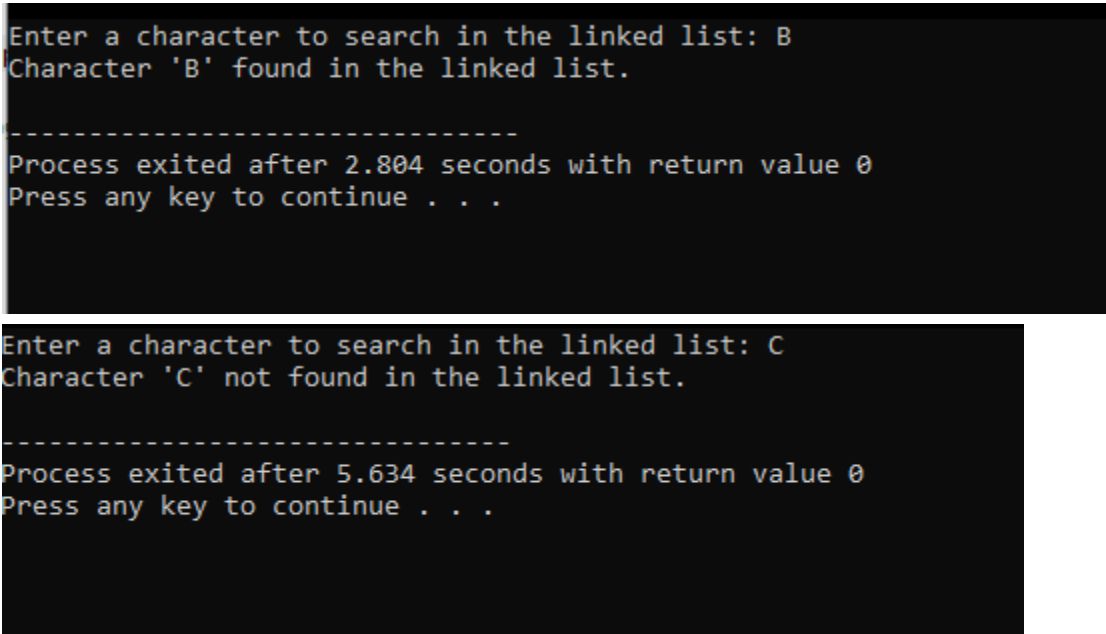
Code	main.cpp	nodes.h	searching.h
	<pre> #include <iostream> #include <cstdlib> #include <time.h> #include "searching.h" #include "nodes.h" using namespace std; const int max_size = 50; int main() { int dataset[max_size]; srand(time(0)); </pre>	<pre> #ifndef NODES_H #define NODES_H template <typename T> class Node { public: T data; Node *next; }; template <typename T> Node<T> *new_node(T newData) { </pre>	<pre> #define SEARCHING_H bool linearSearch(int data[], int size, int item) { for (int i = 0; i < size; i++) { if (data[i] == item) { return true; } } return false; } </pre>

	<pre> for (int i = 0; i < max_size; i++) { dataset[i] = rand(); } /* // Display dataset content cout << "Generated dataset: ", for (int i = 0; i < max_size; i++) { cout << dataset[i] << " "; } cout << endl; */ int search_item; cout << "Enter an item to search: "; cin >> search_item; bool result = linearSearch(dataset, max_size, search_item); if (result) { cout << "Searching is successful" << endl; } else { cout << "Searching is unsuccessful" << endl; } return 0; } </pre>	<pre> Node<T> *newNode = new Node<T>; newNode->data = newData; newNode->next = NULL; return newNode; } #endif </pre>	
--	---	---	--

Screenshot	<pre> Enter an item to search: 2773 Searching is unsuccessful ----- Process exited after 8.202 seconds with return value 0 Press any key to continue . . . _ Enter an item to search: 26124 Searching is successful ----- Process exited after 5.416 seconds with return value 0 Press any key to continue . . . </pre>
Observations	<p>The C++ project is made up of three files: main.cpp, nodes.h, and searching.h. In main.cpp, the program creates a dataset of random integers and asks the user to enter a number they want to search for. It then uses a linear search function from searching.h to see if that number is in the dataset, giving feedback on whether it was found. The nodes.h file contains a template class for a linked list node and a function to create new nodes. Meanwhile, searching.h includes implementing the linear search function, which goes through the array to find a match.</p>

Table 6-2a. Linear Search for Arrays

Code	main.cpp	nodes.h	searching.h
	<pre> #include <iostream> #include "nodes.h" #include "searching.h" using namespace std; int main() { Node<char> *name1 = new_node('B'); Node<char> *name2 = new_node('O'); Node<char> *name3 = new_node('N'); Node<char> *name4 = new_node('A'); name1->next = name2; name2->next = name3; name3->next = name4; </pre>	<pre> #ifndef NODES_H #define NODES_H template <typename T> class Node { public: T data; Node *next; }; template <typename T> Node<T> *new_node(T newData) { Node<T> *newNode = new Node<T>; newNode->data = newData; newNode->next = NULL; return newNode; </pre>	<pre> #define SEARCHING_H #include "nodes.h" template <typename T> bool linearLS(Node<T> *head, T dataFind) { Node<T> *current = head; while (current != NULL) { if (current->data == dataFind) { return true; } current = current->next; } return false; } </pre>

	<pre> name4->next = NULL; char searchChar; cout << "Enter a character to search in the linked list: "; cin >> searchChar; bool found = linearLS(name1, searchChar); if (found) { cout << "Character '" << searchChar << "' found in the linked list." << endl; } else { cout << "Character '" << searchChar << "' not found in the linked list." << endl; } return 0; } </pre>	<pre> } #endif </pre>	
Screenshot	 <p>Enter a character to search in the linked list: B Character 'B' found in the linked list.</p> <p>----- Process exited after 2.804 seconds with return value 0 Press any key to continue . . .</p> <p>Enter a character to search in the linked list: C Character 'C' not found in the linked list.</p> <p>----- Process exited after 5.634 seconds with return value 0 Press any key to continue . . .</p>		
Observations	<p>This C++ project includes three files: main.cpp, nodes.h, and searching.h, which work together to create a simple linked list with search functionality. In main.cpp, four nodes are created to spell the name "BONA," and these nodes are linked together. The program then asks the user to enter a character to search for in the list. It uses the linearLS function from searching.h to check if the character is present, providing feedback on whether it was found. The nodes.h file defines a template class for the linked list nodes and includes a function to create new nodes. Meanwhile, searching.h</p>		

	contains the implementation of the linear search function, which goes through the linked list to find a match.
--	--

Table 6-2b. Linear Search for Linked List

Code	main.cpp <pre> #include <iostream> #include "searching.h" using namespace std; int main() { int dataset[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; int size = sizeof(dataset) / sizeof(dataset[0]); int searchItem; cout << "Enter a number to search: "; cin >> searchItem; binarySearch(dataset, size, searchItem); return 0; } </pre>	searching.h <pre> #ifndef SEARCHING_H #define SEARCHING_H #include <iostream> using namespace std; bool binarySearch(int arr[], int size, int no) { int low = 0; int up = size - 1; while (low <= up) { int mid = low + (up - low) / 2; if (arr[mid] == no) { cout << "Search element is found!" << endl; return true; } else if (no < arr[mid]) { up = mid - 1; } else { low = mid + 1; } } cout << "Search element is not found." << endl; return false; } #endif </pre>
------	---	--

Screenshot	<pre> Enter a number to search: 1 Search element is found! ----- Process exited after 4.149 seconds with return value 0 Press any key to continue . . . Enter a number to search: 0 Search element is not found. ----- Process exited after 1.339 seconds with return value 0 Press any key to continue . . . </pre>
Observations	<p>This C++ project includes two files: main.cpp and searching.h, which work together to implement a binary search algorithm on an array of integers from 1 to 10. In main.cpp, the program asks the user to enter a number to search for and then calls the binarySearch function from searching.h. This function looks for the entered number by comparing it to the middle element of the array and adjusting the search range based on that comparison. If it finds the number, it displays a success message; if not, it lets the user know that the number isn't in the array.</p>

Table 6-3a. Binary Search for Arrays

Code	main.cpp	nodes.h	searching.h
	<pre> #include <iostream> #include "nodes.h" #include "searching.h" using namespace std; int main() { char choice = 'y'; int count = 0; int newData; Node<int>* head = NULL; Node<int>* temp; while (choice == 'y') { cout << "Enter data: "; cin >> newData; </pre>	<pre> #ifndef NODES_H #define NODES_H using namespace std; template <typename T> class Node { public: T data; Node *next; }; template <typename T> Node<T> *new_node(T newData) { </pre>	<pre> #ifndef SEARCHING_H #define SEARCHING_H #include <iostream> #include "nodes.h" using namespace std; template <typename T> Node<T>* getMiddle(Node<T>* start, Node<T>* last) { if (start == NULL) return NULL; Node<T>* slow = start; Node<T>* fast = start->next; </pre>

	<pre> Node<int>* node = new_node(newData); if (head == NULL) { head = node; cout << "Successfully added " << head->data << " to the list.\n"; } else { temp = head; while (temp->next != NULL) { temp = temp->next; } temp->next = node; cout << "Successfully added " << node->data << " to the list.\n"; } count++; cout << "Continue? (y/n): "; cin >> choice; } Node<int>* currNode = head; cout << "Linked list contents: ", while (currNode != NULL) { cout << currNode->data << " "; currNode = currNode->next; } cout << endl; int searchKey; cout << "Enter a number to search: "; cin >> searchKey; binarySearch(head, searchKey); return 0; } </pre>	<pre> Node<T> *newNode = new Node<T>; newNode->data = newData; newNode->next = NULL; return newNode; } #endif </pre>	<pre> while (fast != last) { fast = fast->next; if (fast != last) { slow = slow->next; fast = fast->next; } } return slow; } template <typename T> bool binarySearch(Node<T>* head, T key) { Node<T>* start = head; Node<T>* last = NULL; while (start != last) { Node<T>* mid = getMiddle(start, last); if (mid->data == key) { cout << "Search element is found: " << mid->data << endl; return true; } else if (mid->data > key) { last = mid; } else { start = mid->next; } } cout << "Search element is not found." << endl; return false; } #endif </pre>
--	---	---	--

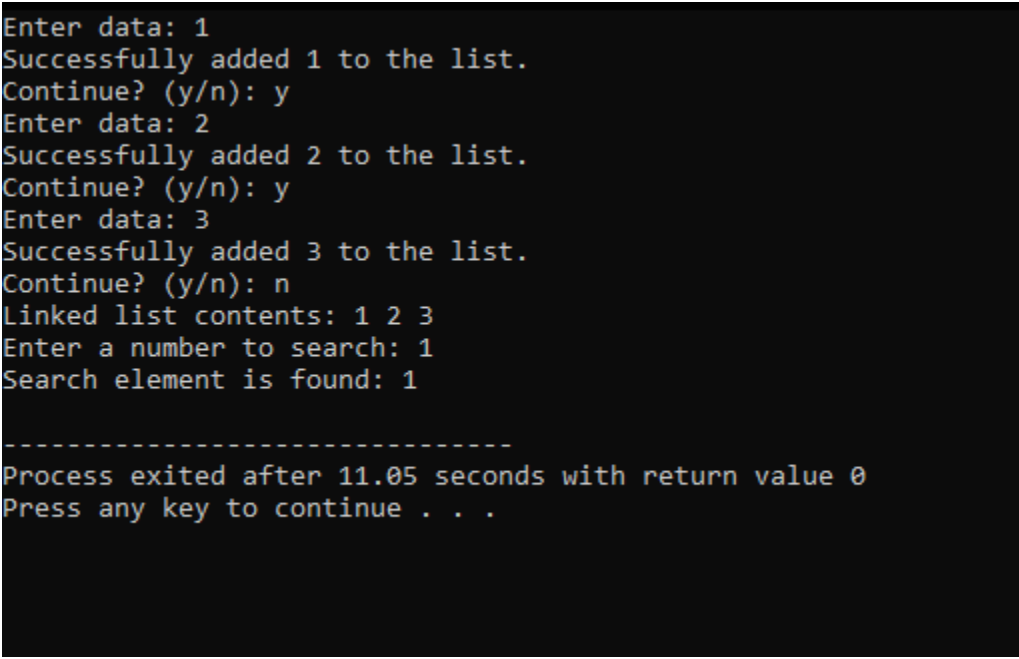
Screenshot	 <pre> Enter data: 1 Successfully added 1 to the list. Continue? (y/n): y Enter data: 2 Successfully added 2 to the list. Continue? (y/n): y Enter data: 3 Successfully added 3 to the list. Continue? (y/n): n Linked list contents: 1 2 3 Enter a number to search: 1 Search element is found: 1 ----- Process exited after 11.05 seconds with return value 0 Press any key to continue . . . </pre>		
Observations	<p>This C++ project includes three files: main.cpp, nodes.h, and searching.h, which work together to create a linked list that allows users to add integers and search for them using binary search. In main.cpp, users can input numbers to build the linked list, which dynamically adds nodes at the end. After all the numbers are entered, the program displays the contents of the list and prompts the user to enter a number to search for. The searching.h file contains a function that finds the middle node of the list and implements the binary search algorithm to check if the entered number exists in the list.</p>		

Table 6-3b. Binary Search for Linked List

7. Supplementary Activity

Problem1

```

#include <iostream>

using namespace std;

template <typename T>
class Node {
public:
    T data;
    Node *next;

    Node(T newData) {
        data = newData;
        next = NULL;
    }
};

```



```

int sequentialSearchArray(int arr[], int size, int key) {
    int comparisons = 0;
    for (int i = 0; i < size; i++) {
        comparisons++;
        if (arr[i] == key) {
            return comparisons;
        }
    }
    return comparisons;
}

```

```

template <typename T>
int sequentialSearchLinkedList(Node<T>* head, T key) {
    int comparisons = 0;
    Node<T>* current = head;

    while (current != NULL) {
        comparisons++;
        if (current->data == key) {
            return comparisons;
        }
        current = current->next;
    }

    return comparisons;
}

```

```

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 arrayComparisons = sequentialSearchArray(arr, size, key);
    cout << "Comparisons in array: " << arrayComparisons << endl;

    Node<int>* head = new Node<int>(15);
    head->next = new Node<int>(18);
    head->next->next = new Node<int>(2);
    head->next->next->next = new Node<int>(19);
    head->next->next->next->next = new Node<int>(18);
    head->next->next->next->next->next = new Node<int>(0);
    head->next->next->next->next->next->next = new Node<int>(8);
    head->next->next->next->next->next->next->next = new Node<int>(14);
    head->next->next->next->next->next->next->next->next = new Node<int>(19);
    head->next->next->next->next->next->next->next->next->next = new Node<int>(14);

    int linkedListComparisons = sequentialSearchLinkedList(head, key);
    cout << "Comparisons in linked list: " << linkedListComparisons << endl;

    Node<int>* current = head;
}

```

```

while (current != NULL) {
    Node<int>* temp = current;
    current = current->next;
    delete temp;
}

return 0;
}

```

```

Comparisons in array: 2
Comparisons in linked list: 2

-----
Process exited after 0.04861 seconds with return value 0
Press any key to continue . . .

```

Problem2

```

#include <iostream>

using namespace std;

template <typename T>
class Node {
public:
    T data;
    Node *next;

    Node(T newData) {
        data = newData;
        next = NULL;
    }
};

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

template <typename T>
int countOccurrencesLinkedList(Node<T>* head, T key) {
    int count = 0;
    Node<T>* current = head;

```

```

while (current != NULL) {
    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 arrayCount = countOccurrencesArray(arr, size, key);
    cout << "Occurrences of " << key << " in array: " << arrayCount << endl;

    Node<int>* head = new Node<int>(15);
    head->next = new Node<int>(18);
    head->next->next = new Node<int>(2);
    head->next->next->next = new Node<int>(19);
    head->next->next->next->next = new Node<int>(18);
    head->next->next->next->next->next = new Node<int>(0);
    head->next->next->next->next->next->next = new Node<int>(8);
    head->next->next->next->next->next->next->next = new Node<int>(14);
    head->next->next->next->next->next->next->next->next = new Node<int>(19);
    head->next->next->next->next->next->next->next->next->next = new Node<int>(14);

    int linkedListCount = countOccurrencesLinkedList(head, key);
    cout << "Occurrences of " << key << " in linked list: " << linkedListCount << endl;

    Node<int>* current = head;
    while (current != NULL) {
        Node<int>* temp = current;
        current = current->next;
        delete temp;
    }

    return 0;
}

```

```

Occurrences of 18 in array: 2
Occurrences of 18 in linked list: 2

-----
Process exited after 0.05657 seconds with return value 0
Press any key to continue . . .

```

Problem 3

```
#include <iostream>

using namespace std;

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

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

        if (arr[mid] == key) {
            return mid;
        }
        else if (arr[mid] < key) {
            low = mid + 1;
        }
        else {
            up = mid - 1;
        }
    }
    return -1;
}

int main() {
    int arr[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key = 8;

    int result = binarySearch(arr, size, key);
    if (result != -1) {
        cout << "Element found at index: " << result << endl;
    } else {
        cout << "Element not found!" << endl;
    }

    return 0;
}
```

```
Element found at index: 3
```

```
-----
Process exited after 0.04634 seconds with return value 0
Press any key to continue . . .
```

Problem4

```
#include <iostream>

using namespace std;

int recursiveBinarySearch(int arr[], int low, int up, int key) {
    if (low > up) {
        return -1;
    }

    int mid = low + (up - low) / 2;

    if (arr[mid] == key) {
        return mid;
    }
    else if (arr[mid] < key) {
        return recursiveBinarySearch(arr, mid + 1, up, key);
    }
    else {
        return recursiveBinarySearch(arr, low, mid - 1, key);
    }
}

int main() {
    int arr[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key = 8;

    int result = recursiveBinarySearch(arr, 0, size - 1, key);
    if (result != -1) {
        cout << "Element found at index: " << result << endl;
    } else {
        cout << "Element not found!" << endl;
    }

    return 0;
}
```

```
Element found at index: 3
```

```
-----
Process exited after 0.04258 seconds with return value 0
Press any key to continue . . .
```

.
8. Conclusion
<p>In this activity, I learned key concepts about searching algorithms, specifically sequential and binary search. I gained practical experience implementing these algorithms using arrays and linked lists, which helped me understand their mechanics and efficiency. The process involved writing code, testing it on datasets, and debugging errors, such as issues with nullptr and class definitions. I explored both iterative and recursive methods, appreciating how recursion can simplify code. The supplementary activity deepened my understanding of algorithm efficiency and data handling, as I modified the algorithms to count instances and transition from iterative to recursive implementations. Overall, I performed well but identified areas for improvement, including better error handling, optimization, and documentation. This experience has enriched my knowledge of fundamental algorithms and data structures, which I look forward to applying in future challenges.</p>
9. Assessment Rubric