Activity No. 6	
SEARCHING TECHNIQUES	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:15/10/2024**
Section:CPE21S4	**Date Submitted:15/10/2024**
Name(s): Kenn Jie L. Valleser	**Instructor: Engr. Ma. Rizette Sayo**

6. Output

Screenshot

```
#include <iostream>
#include <cstdlib> // for generating random integers
#include <ctime>   // for seeding the random number generator

const int max_size = 50;

int main() {

    srand(time(0));


    int dataset[max_size];
    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;
}
```

Observations

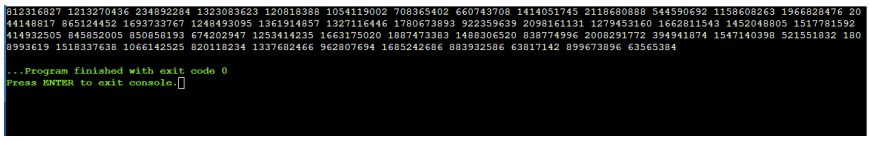


Table 6-1. Data Generated and Observations.

Code

```
#ifndef SEARCHING_H
#define SEARCHING_H

#include <iostream>
```

	template <typename T>
	bool linearSearch(T data[], int n, T item) {
	  for (int i = 0; i < n; i++) {
	  if (data[i] == item) {
	  std::cout << "Searching is successful. Item found at index " << i << std::endl;
	  return true;
	  }
	  }
	  std::cout << "Searching is unsuccessful. Item not found in the array." << std::endl;
	  return false;
	}
	
	#endif

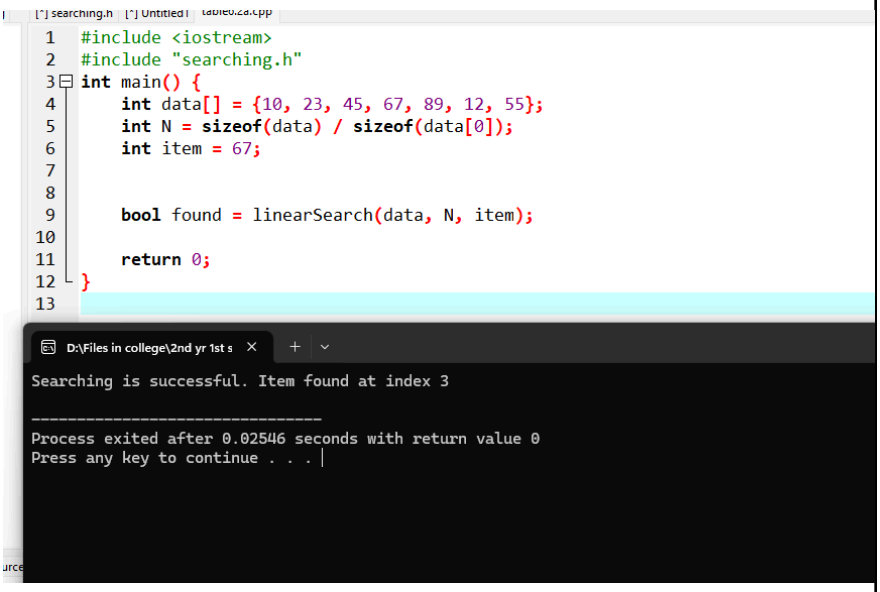
Output	
Observations	The linear search worked well for finding the target element in the array. When the element was present, it confirmed its location with a success message. However, when the element wasn't found, it correctly indicated that the search was unsuccessful, showing that this method can be slow with larger datasets since it has a time complexity of O(N).

Table 6-2a. Linear Search for Arrays

Code	#ifndef SEARCHING2_H #define SEARCHING2_H #include "nodes.h" #include <iostream>

```
template <typename T>
bool linearLS(Node<T>* head, T dataFind) {
    Node<T>* current = head;

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

    std::cout << "Searching is unsuccessful: " << dataFind
<< " not found." << std::endl;
    return false;
}

#endif
```

==========

```
#ifndef NODES_H
#define NODES_H



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

    // Constructor
    Node(T value) : data(value), next(NULL) {}
};

#endif // NODES_H
```

==========

```
#include <iostream>
#include "nodes.h"
#include "searching2.h"

int main() {

    Node<char>* name1 = new Node<char>('R');
    Node<char>* name2 = new Node<char>('o');
```

	Node<char>* name3 = new Node<char>('m');
	Node<char>* name4 = new Node<char>('a');
	Node<char>* name5 = new Node<char>('n');
	
	name1->next = name2;
	name2->next = name3;
	name3->next = name4;
	name4->next = name5;
	name5->next = NULL;
	
	char dataFind = 'n';
	linearLS(name1, dataFind);
	
	delete name1;
	delete name2;
	delete name3;
	delete name4;
	delete name5;
	
	return 0;
	}

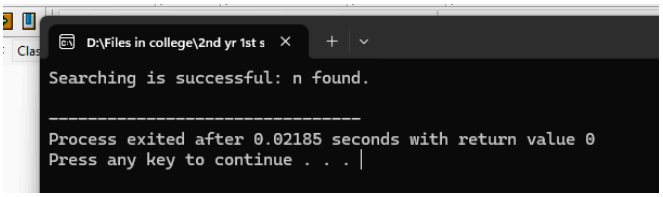
Output	
Observation	In the linked list, the linear search also managed to find the target character when it was there, providing a nice success message. If the character wasn't in the list, it clearly stated that the search failed. Just like with the array, this method has an O(N) time complexity, which means it can take a long time for bigger lists.

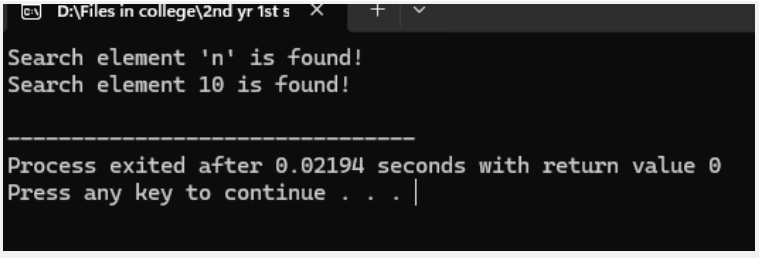
Table 6-2b. Linear Search for Linked List

Code	#ifndef SEARCHING3_H #define SEARCHING3_H #include "nodes.h" #include <iostream>

```
void linearLS(Node<char>* head, char dataFind) {
    Node<char>* current = head;
    while (current != NULL) {
        if (current->data == dataFind) {
            std::cout << "Search element '" << dataFind << "' is
found!" << std::endl;
            return;
        }
        current = current->next;
    }
    std::cout << "Search element '" << dataFind << "' is not
found!" << std::endl;
}


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

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

        if (item == arr[mid]) {
            std::cout << "Search element " << item << " is found!" <<
std::endl;
            return true; // Item found
        }
        else if (item < arr[mid]) {
            up = mid - 1; // Search in the lower half
        }
        else {
            low = mid + 1; // Search in the upper half
        }
    }

    std::cout << "Search element " << item << " is not found!" <<
std::endl;
    return false;
}

#endif

==========================================

#include <iostream>
#include "nodes.h"
#include "searching3.h"
```

```
int main() {

    Node<char>* name1 = new Node<char>('R');
    Node<char>* name2 = new Node<char>('o');
    Node<char>* name3 = new Node<char>('m');
    Node<char>* name4 = new Node<char>('a');
    Node<char>* name5 = new Node<char>('n');


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


    char dataFind = 'n';
    linearLS(name1, dataFind);


    int arr[] = {10, 23, 45, 67, 89};
    int n = sizeof(arr) / sizeof(arr[0]);
    int item = 10;


    binarySearch(arr, n, item);


    delete name1;
    delete name2;
    delete name3;
    delete name4;
    delete name5;

    return 0;
}
```

Output



Observation

The binary search was really efficient in the sorted array; it quickly found the target element and let me know with a success message. If

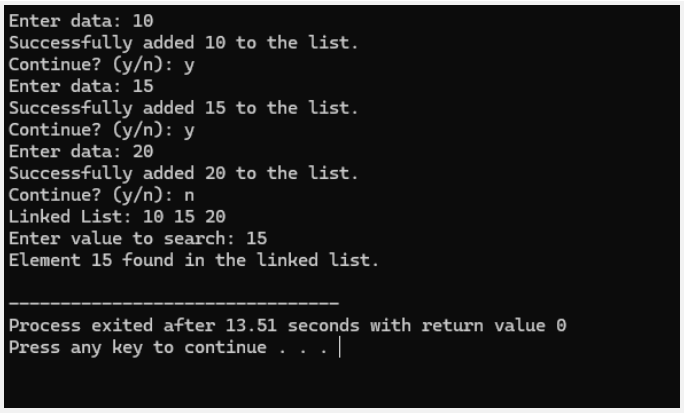
	the element wasn't found, the program clearly stated that it wasn't there. This method is way faster than linear search, operating with a time complexity of O(log N), which is great for searching in large arrays.

Table 6-3a. Binary Search for Arrays

Code	#include <iostream>
	
	template <typename T>
	struct Node {
	  T data;
	  Node* next;
	
	  Node(T value) : data(value), next(NULL) {}
	};
	
	template <typename T>
	Node<T>* new_node(T value) {
	  return new Node<T>(value);
	}
	
	template <typename T>
	void displayList(Node<T>* head) {
	  Node<T>* currNode = head;
	  while (currNode != NULL) {
	  std::cout << currNode->data << " ";
	  currNode = currNode->next;
	  }
	  std::cout << std::endl;
	}
	
	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>* binarySearch(Node<T>* head, T key) {
    Node<T>* start = head;
    Node<T>* end = NULL;

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


        if (mid->data == key) {
            return mid;
        }

        else if (mid->data > key) {
            end = mid;
        }

        else {
            start = mid->next;
        }
    }
    return NULL;
}

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

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


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

        else if (count == 2) {
            node = new_node(newData);
            head->next = node;
            node->next = NULL;
            std::cout << "Successfully added " << node->data
```

```
<< " to the list.\n";
            count++;
        }

        else {
            temp = head;
            while (temp->next != NULL) {
                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 << "Linked List: ";
    displayList(head);


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

    Node<int>* result = binarySearch(head, key);
    if (result != NULL) {
        std::cout << "Element " << key << " found in the
linked list.\n";
    } else {
        std::cout << "Element " << key << " not found in the
linked list.\n";
    }


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

    return 0;
}
```

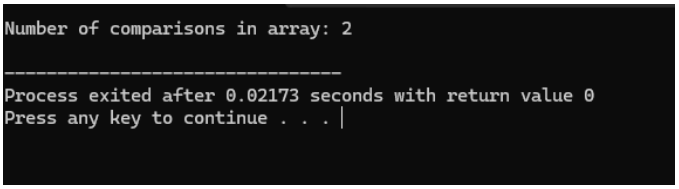
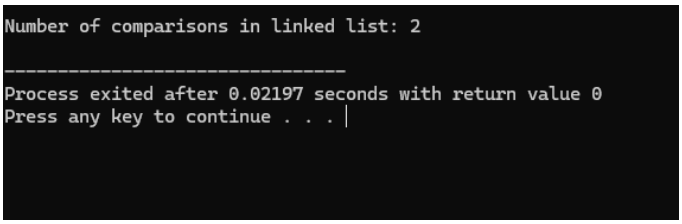
Output	
Observation	When I used binary search on the sorted linked list, it also found the target value if it was present, giving me a success message. If the value was missing, it let me know that it couldn't be found. However, even though this method is more efficient with an O(log N) time complexity, it felt a bit slower because of the extra overhead involved in dealing with linked lists compared to arrays.

7. Supplementary Activity

Problem 1 code	#include <iostream>
	
	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;
	}
	
	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 comparisons = sequentialSearchArray(arr, size, key);
	  std::cout << "Number of comparisons in array: " <<

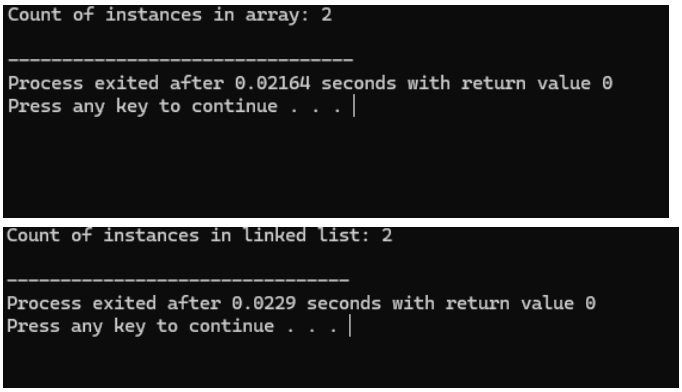
```
comparisons << std::endl;

    return 0;
}

==========================================

#include <iostream>

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

Node* newNode(int data) {
    Node* node = new Node();
    node->data = data;
    node->next = NULL;
    return node;
}

int sequentialSearchLinkedList(Node* head, int key) {
    int comparisons = 0;
    Node* current = head;

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

int main() {
    Node* head = newNode(15);
    head->next = newNode(18);
    head->next->next = newNode(2);
    head->next->next->next = newNode(19);
    head->next->next->next->next = newNode(18);
    head->next->next->next->next->next = newNode(0);
    head->next->next->next->next->next->next =
newNode(8);
    head->next->next->next->next->next->next->next =
newNode(14);

head->next->next->next->next->next->next->next->next =
newNode(19);

head->next->next->next->next->next->next->next->next->
```

	next = newNode(14);
	
	int key = 18;
	int comparisons = sequentialSearchLinkedList(head, key);
	std::cout << "Number of comparisons in linked list: " << comparisons << std::endl;
	
	return 0;
	}

Output	 
Answer	Both are 2

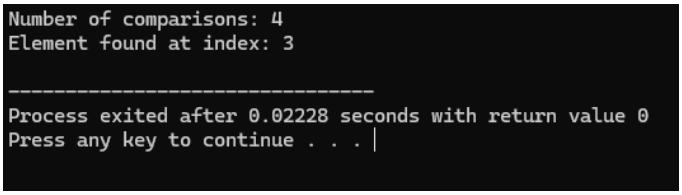
Problem 2 code	#include <iostream>
	
	int countInstancesArray(int arr[], int size, int key) {
	int count = 0;
	for (int i = 0; i < size; i++) {
	if (arr[i] == key) {
	count++;
	}
	}
	return count;
	}
	
	int main() {
	int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
	int size = sizeof(arr) / sizeof(arr[0]);
	int key = 18;
	
	int count = countInstancesArray(arr, size, key);
	std::cout << "Count of instances in array: " << count << std::endl;
	
	return 0;
	}

```
==========
#include <iostream>

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

Node* newNode(int data) {
    Node* node = new Node();
    node->data = data;
    node->next = NULL;
    return node;
}

int countInstancesLinkedList(Node* head, int key) {
    int count = 0;
    Node* current = head;

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

int main() {
    Node* head = newNode(15);
    head->next = newNode(18);
    head->next->next = newNode(2);
    head->next->next->next = newNode(19);
    head->next->next->next->next = newNode(18);
    head->next->next->next->next->next = newNode(0);
    head->next->next->next->next->next->next =
newNode(8);
    head->next->next->next->next->next->next->next =
newNode(14);

head->next->next->next->next->next->next->next->next =
newNode(19);

head->next->next->next->next->next->next->next->next->
next = newNode(14);
    int key = 18;
    int count = countInstancesLinkedList(head, key);
    std::cout << "Count of instances in linked list: " <<
count << std::endl;

    return 0;
```

	}
Output	

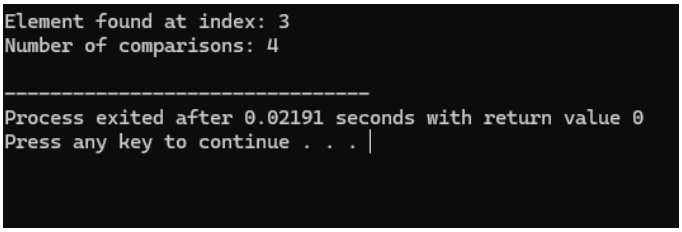
Problem 3 Code	#include <iostream>
	
	int binarySearch(int arr[], int size, int key) {
	int low = 0;
	int high = size - 1;
	int mid;
	int comparisons = 0;
	
	while (low <= high) {
	mid = low + (high - low) / 2;
	comparisons++;
	
	if (arr[mid] == key) {
	std::cout << "Number of comparisons: " <<
	comparisons << std::endl;
	return mid;
	}
	else if (arr[mid] < key) {
	low = mid + 1;
	}
	else {
	high = mid - 1;
	}
	}
	std::cout << "Number of comparisons: " <<
	comparisons << std::endl;
	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) {
	std::cout << "Element found at index: " << result << std::endl;
	} else {
	std::cout << "Element not found." << std::endl;
	}
	
	return 0;
	}

Output	
Diagram	Initial State: Low = 0, High = 9 Mid Index Calculation: mid = (0 + 9) / 2 = 4 (Value = 11) First Iteration: Compare: arr[mid] (11) > key (8) ⟶ Move to the left New State: Low = 0, High = 3 Second Iteration: Low = 0, High = 3 Mid Index Calculation: mid = (0 + 3) / 2 = 1 (Value = 5) Second Iteration: Compare: arr[mid] (5) < key (8) ⟶ Move to the right New State: Low = 2, High = 3 Third Iteration: Low = 2, High = 3 Mid Index Calculation: mid = (2 + 3) / 2 = 2 (Value = 6) Third Iteration: Compare: arr[mid] (6) < key (8) ⟶ Move to the right New State: Low = 3, High = 3 Fourth Iteration: Low = 3, High = 3

	Mid Index Calculation: mid = (3 + 3) / 2 = 3 (Value = 8) Compare: arr[mid] (8) == key (8) → Element found!

Problem 4 Code	#include <iostream>
	
	int recursiveBinarySearch(int arr[], int low, int high, int key, int &comparisons) {
	  if (low > high) {
	  return -1;
	  }
	
	  int mid = low + (high - low) / 2;
	  comparisons++;
	
	  if (arr[mid] == key) {
	  return mid;
	  }
	  else if (arr[mid] < key) {
	  return recursiveBinarySearch(arr, mid + 1, high, key, comparisons);
	  }
	  else {
	  return recursiveBinarySearch(arr, low, mid - 1, key, comparisons);
	  }
	}
	
	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 comparisons = 0;
	
	  int result = recursiveBinarySearch(arr, 0, size - 1, key, comparisons);
	  if (result != -1) {
	  std::cout << "Element found at index: " << result << std::endl;
	  std::cout << "Number of comparisons: " << comparisons << std::endl;
	  } else {
	  std::cout << "Element not found." << std::endl;
	  }
	
	  return 0;
	}



8. Conclusion

I learned the differences between sequential and binary search algorithms and how their efficiency varies based on data structure, like arrays and linked lists. Implementing these algorithms helped me grasp the importance of algorithmic complexity and the role of recursion in simplifying code. Additionally, I understood how linked lists can be utilized effectively for dynamic data storage.I believe I performed well in this activity, successfully implementing both searching algorithms and understanding their differences. However, I realize I need to improve my debugging skills and deepen my understanding of recursive functions. Overall, this exercise enhanced my programming abilities and confidence in working with data structures and algorithms.

9. Assessment Rubric