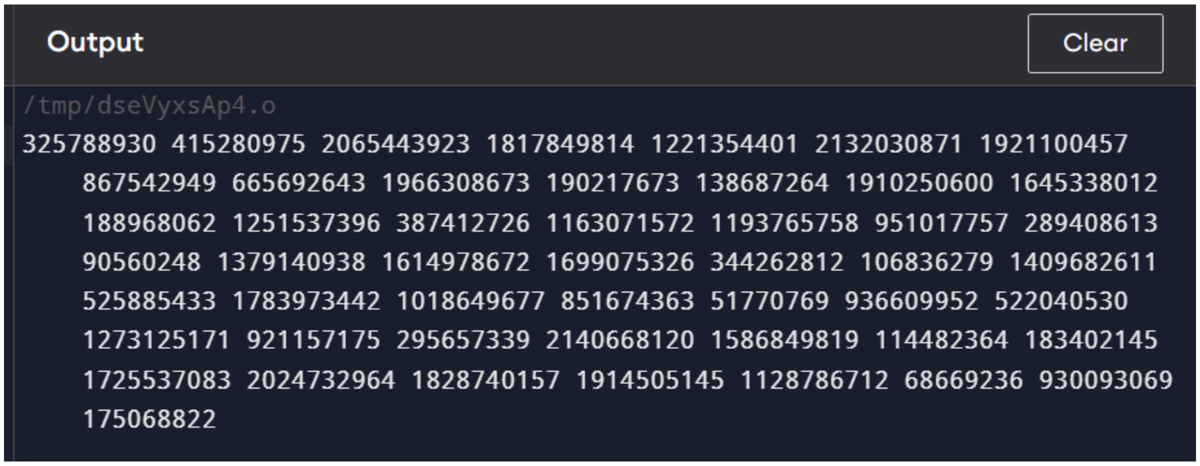
Activity No. 6	
SEARCHING TECHNIQUES	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 10/14/24**
Section: CPE21S4	**Date Submitted: 10/14/24**
Name(s): Dominic Joseph P. Virtucio	**Instructor: Maam Sayo**
6. Output	

Screenshot

```
C/C++
#include <iostream>
#include <cstdlib>
#include <ctime>

const int max_size = 50;
int dataset[max_size];

int main() {
    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;
}
```

Observations



```
/tmp/dseVyxsAp4.o
325788930 415280975 2065443923 1817849814 1221354401 2132030871 1921100457
    867542949 665692643 1966308673 190217673 138687264 1910250600 1645338012
    188968062 1251537396 387412726 1163071572 1193765758 951017757 289408613
    90560248 1379140938 1614978672 1699075326 344262812 106836279 1409682611
    525885433 1783973442 1018649677 851674363 51770769 936609952 522040530
    1273125171 921157175 295657339 2140668120 1586849819 114482364 183402145
    1725537083 2024732964 1828740157 1914505145 1128786712 68669236 930093069
    175068822
```

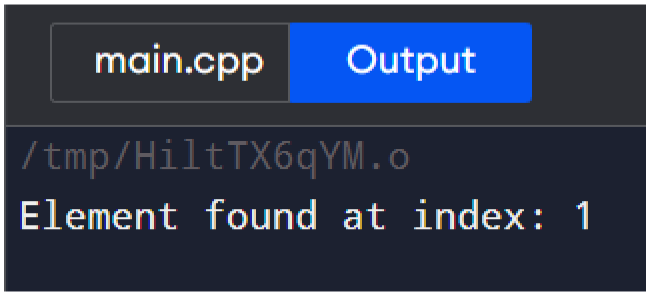
Table 6-1. Data Generated and Observations.

Code	

C/C++

```cpp
#include <iostream>

int linearSearch(int arr[], int n, int key) {
  for (int i = 0; i < n; i++) {
    if (arr[i] == key) {
      return i; // Found the key at index i
    }
  }
  return -1; // Key not found in the array
}

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

  int result = linearSearch(arr, n, key);

  if (result != -1) {
    std::cout << "Element found at index: " << result << std::endl;
  } else {
    std::cout << "Element not found in the array." << std::endl;
  }

  return 0;
}
```

Output



```
/tmp/HiltTX6qYM.o
Element found at index: 1
```

Observations

The code uses a basic linear search method. It loops through the array until it finds the key or reaches the end of it. The function returns the key's index if it is found; else, it returns -1. In this scenario, key 18 is located at index 1.

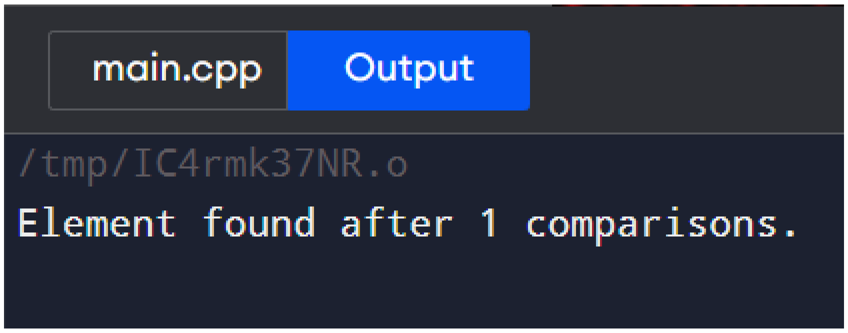
Table 6-2a. Linear Search for Arrays

Code

C/C++

```cpp
#include <iostream>

template <typename T>
class Node {
```

	```cpp
	public:
	  T data;
	  Node* next;

	  Node(T data) {
	    this->data = data;
	    this->next = nullptr;
	  }
	};

	template <typename T>
	int linearSearch(Node<T>* head, T key) {
	  int count = 0;
	  while (head != nullptr) {
	    if (head->data == key) {
	      return count; // Found the key after count comparisons
	    }
	    head = head->next;
	    count++;
	  }
	  return -1; // Key not found in the list
	}

	int main() {
	  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 key = 18;

	  int result = linearSearch(head, key);

	  if (result != -1) {
	    std::cout << "Element found after " << result << " comparisons." <<
	std::endl;
	  } else {
	    std::cout << "Element not found in the list." << std::endl;
	  }

	  return 0;
	}
	```

Output	 main.cpp Output
	```
	/tmp/IC4rmk37NR.o
	Element found after 1 comparisons.
	```

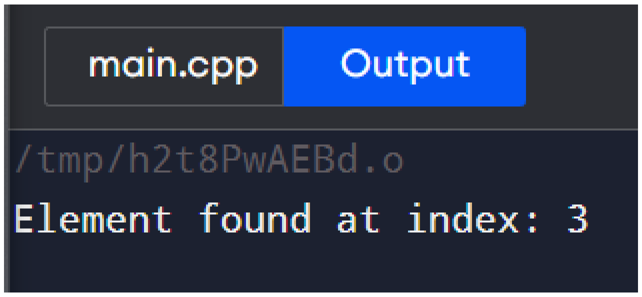
Observations	The code applies a linear search method to a linked list. It loops through the linked list until it finds the key or reaches the end of the list. The function returns the number of comparisons performed before locating the key, or -1 if no key is discovered. In this situation, the key 18 was discovered after one comparison.

Table 6-2b. Linear Search for Linked List

Code

C/C++

```cpp
#include <iostream>

int binarySearch(int arr[], int left, int right, int key) {
  while (left <= right) {
    int mid = left + (right - left) / 2; // Calculate midpoint

    if (arr[mid] == key) {
      return mid; // Key found at index mid
    } else if (arr[mid] < key) {
      left = mid + 1; // Search in the right half
    } else {
      right = mid - 1; // Search in the left half
    }
  }
  return -1; // Key not found in the array
}

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

  int result = binarySearch(arr, 0, n - 1, key);

  if (result != -1) {
    std::cout << "Element found at index: " << result << std::endl;
  } else {
    std::cout << "Element not found in the array." << std::endl;
  }

  return 0;
}
```

Output	
	`/tmp/h2t8PwAEBd.o`
	`Element found at index: 3`

Observations	The code uses a binary search method. It repeatedly splits the search interval in half until the key is discovered or the interval is empty. The function returns the key's index if it is found; else, it returns -1. In this situation, the key 8 is located at index 3.

Table 6-3a. Binary Search for Arrays

Code

C/C++

```cpp
#include <iostream>

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

  Node(T data) {
    this->data = data;
    this->next = nullptr;
  }
};

template <typename T>
Node<T>* getMiddle(Node<T>* head) {
  Node<T>* slow = head;
  Node<T>* fast = head;

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

  return slow;
}

template <typename T>
int binarySearch(Node<T>* head, T key) {
  Node<T>* start = head;
  Node<T>* last = nullptr;

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

    if (mid->data == key) {
      return 1; // Key found
    } else if (mid->data < key) {
      start = mid->next; // Search in the right half
    } else {
      last = mid; // Search in the left half
    }
  }

  return 0; // Key not found
}

int main() {
  Node<int>* head = new Node<int>(3);
  head->next = new Node<int>(5);
  head->next->next = new Node<int>(6);
  head->next->next->next = new Node<int>(8);
  head->next->next->next->next = new Node<int>(11);
  head->next->next->next->next->next = new Node<int>(12);
  head->next->next->next->next->next->next = new Node<int>(14);
  head->next->next->next->next->next->next->next = new Node<int>(15);
  head->next->next->next->next->next->next->next->next = new Node<int>(17);
  head->next->next->next->next->next->next->next->next->next = new
Node<int>(18);

  int key = 8;

  int result = binarySearch(head, key);

  if (result) {
    std::cout << "Element found in the list." << std::endl;
  } else {
    std::cout << "Element not found in the list." << std::endl;
  }

  return 0;
}
```

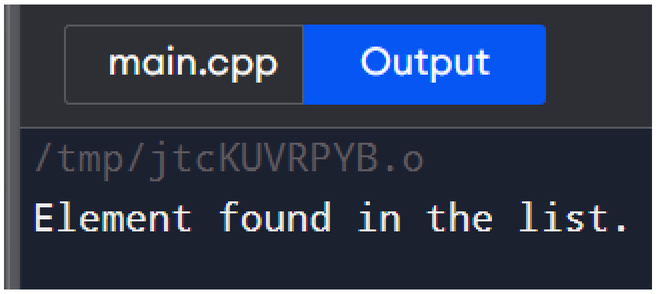
Output	 `/tmp/jtcKUVRPYB.o` `Element found in the list.`
Observations	The code executes a binary search algorithm on a sorted linked list. It uses the getMiddle method to locate the linked list's middle node. The algorithm cuts the search interval in half until the key is located or the interval is empty. The method returns 1 if the key is discovered; else, it returns 0. In this situation, key 8 is located in the linked list.

Table 6-3b. Binary Search for Linked Lis

7. Supplementary Activity

PROBLEM 1

```
C/C++
#include <iostream>

int linearSearch(int arr[], int n, int key) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    count++;
    if (arr[i] == key) {
      return count;
    }
  }
  return -1;
}

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

  int comparisons = linearSearch(arr, n, key);

  if (comparisons != -1) {
    std::cout << "Key found after " << comparisons << "
comparisons." << std::endl;
  } else {
    std::cout << "Key not found in the array." << std::endl;
  }

  return 0;
}
```

To identify the key '18' in the list [15, 18, 2, 19, 18, 0, 8, 14, 19, 14] using a sequential search, I discovered that two comparisons are required. The search begins at the beginning of the list, with the first element (15) compared to the key '18', resulting in a mismatch. Moving on to the following element (index 1), I discovered that the second element (18) matches the key, which ended the search after two comparisons.

PROBLEM 2

```
C/C++
#include <iostream>

int linearSearch(int arr[], int n, int key) {
  int count = 0;
  int repetitions = 0;
  for (int i = 0; i < n; i++) {
    count++;
    if (arr[i] == key) {
      repetitions++;
```

When changing the sequential search method to count repetitions of a particular key 'k', I would take the following steps: First, I set the variable repetitions to zero. Then I iterate through the list, checking each entry against the key 'k'. If an

```
    }
  }
  return repetitions;
}

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

  int repetitions = linearSearch(arr, n, key);

  std::cout << "Number of repetitions of " << key << ": " <<
repetitions << std::endl;

  return 0;
}
```

element matches, I increase the repetitions variable. After traversing the full list, the updated method yields 2 as the number of repetitions for key '18' in the list.

PROBLEM 3

C/C++
```
#include <iostream>

int binarySearch(int arr[], int left, int right, int key) {
  while (left <= right) {
    int mid = left + (right - left) / 2;

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

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

  int result = binarySearch(arr, 0, n - 1, key);

  if (result != -1) {
    std::cout << "Element found at index: " << result << std::endl;
  } else {
```

To show the binary search strategy for finding the key '8' in the sorted list [3, 5, 6, 8, 11, 12, 14, 15, 17, 18], I begin with the complete range from index 0 to 9. The midpoint is estimated as 4, and the element is 11. Because 11 is greater than 8, I limit the search to the left side (index 0 to 3). The new midpoint is at index 1 (element 5), which is less than 8, thus I go to the right half (index 2–3). The following midpoint is at index 2 (element 6), which is still less than 8, resulting in a final narrowing down to index 3, where the element is 8, successfully matching the key.

```
    std::cout << "Element not found in the array." << std::endl;
  }

  return 0;
}
```

PROBLEM 4

C/C++

```cpp
#include <iostream>

int binarySearchRecursive(int arr[], int left, int right, int key)
{
  if (left > right) {
    return -1;
  }
  int mid = left + (right - left) / 2;

  if (arr[mid] == key) {
    return mid;
  } else if (arr[mid] < key) {
    return binarySearchRecursive(arr, mid + 1, right, key);
  } else {
    return binarySearchRecursive(arr, left, mid - 1, key);
  }
}

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

  int result = binarySearchRecursive(arr, 0, n - 1, key);

  if (result != -1) {
    std::cout << "Element found at index: " << result << std::endl;
  } else {
    std::cout << "Element not found in the array." << std::endl;
  }

  return 0;
}
```

8. Conclusion

This assignment taught me two essential search techniques: linear search and binary search, as well as how to implement them in arrays and linked lists. The approach entailed coding these algorithms in C++, as well as counting

comparisons and repetitions, which I found to be a difficult yet satisfying experience that helped me understand. The supplemental exercise provided actual applications for these search strategies, which improved my abilities to solve real-world situations. Overall, I believe I fared well, but I acknowledge the need to increase my understanding of temporal complexity, investigate advanced search methods, and improve my code documentation for readability.
9. Assessment Rubric