ACTIVITY NO. 6	
SEARCHING TECHNIQUES	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:**10/14/2024
Section:CPE21S4	**Date Submitted:**10/16/2024
Name(s): Sanchez,Christan Ray R.	**Instructor:** Prof. 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

```
Random Dataset: 88 80 14 19 15 60 23 28 70 40 18 15 78 35 53 25 19 59 50 10 31 62 78 88 8 46 23 97 98 64 64 82 21 92 79
71 34 22 92 67 67 87 40 10 16 80 33 94 75 50
--------------------------------
```

Table 6-1. Data Generated and Observations.

Screenshot	main.cpp | searching.h | nodes.h 1 template <typename T> 2 class Node { 3 public: 4 T data; 5 Node* next; 6 }; 7 8 template <typename T> 9 Node<T>* new_node(T newData) { 10 Node<T>* newNode = new Node<T>; 11 newNode->data = newData; 12 newNode->next = NULL; 13 return newNode; 14 } 15
Output	Searching for item: 573 14974 30497 27728 23471 28730 23632 26076 16127 29876 26857 573 29338 20965 12947 9673 6546 28262 22031 18578 8385 17531 25126 1419 8542 2641 28384 9452 20943 12164 13559 9687 30910 21476 14731 2875 5743 7033 25111 28665 14417 965 1188 9797 10963 7752 18493 26191 32120 21578 27042 Searching for item: 573 Searching is successful.
Observations	If the item exists in the dataset, the search is successful; otherwise, it confirms that the item is not present.

Table 6-2a. Linear Search for Arrays

Screenshot	main.cpp | searching.h | nodes.h 1 void linearSearch(int data[], int n, int item) { 2 for (int i = 0; i < n; i++) { 3 if (data[i] == item) { 4 std::cout << "Searching is successful." << std::endl; 5 return; 6 } 7 } 8 std::cout << "Searching is unsuccessful." << std::endl; 9 } 10
Output	Searching is successful. 14974 30497 27728 23471 28730 23632 26076 16127 29876 26857 573 29338 20965 12947 9673 6546 28262 22031 18578 8385 17531 25126 1419 8542 2641 28384 9452 20943 12164 13559 9687 30910 21476 14731 2875 5743 7033 25111 28665 14417 965 1188 9797 10963 7752 18493 26191 32120 21578 27042 Searching for item: 573 Searching is successful.
Observations	Successfully searches through the linked list nodes; outputs a message indicating the search status.

Table 6-2b. Linear Search for Linked List

Screenshot

main.cpp | searching.h | nodes.h

```
37      int searchItem = dataset[10]; // For example, search for the 11th random number
38      std::cout << "Searching for item: " << searchItem << std::endl;
39      linearSearch(dataset, max_size, searchItem); // Call the linear search function
40
41      // Create linked list for your first name
42      Node<char>* name1 = new_node('R'); // R
43      Node<char>* name2 = new_node('o'); // o
44      Node<char>* name3 = new_node('m'); // m
45      Node<char>* name4 = new_node('a'); // a
46      Node<char>* name5 = new_node('n'); // n
47
48      // Link each node together
49      name1->next = name2;
50      name2->next = name3;
51      name3->next = name4;
52      name4->next = name5;
53      name5->next = nullptr; // End of the list
54
55      // Example item to search in the linked list
56      char searchChar = 'n'; // For example, search for 'n'
57      std::cout << "Searching for character: " << searchChar << std::endl;
58      linearLS(name1, searchChar); // Call the linear search function for the linked list
59
60      return 0;
61  }
```

Output

```
16708 3501 10883 21571 745 11574 7619 1840 15707 6265 20339 19959 12996 17222 30000 5264 14537 8193 7650 15377 17085 207
36 11668 30509 23572 2357 19331 20475 10011 24589 30324 15097 13287 7319 13131 23377 17291 15337 2522 23643 19343 14016
17399 1122 19869 19046 8913 25771 22396 18649
Searching for item: 20339
Searching is successful.
Searching for character: n
Searching in linked list is successful.
```

Observations

The output confirms if the search element exists, emphasizing the efficiency of the binary search algorithm on sorted arrays.

Table 6-3a. Binary Search for Arrays

Screenshot

main.cpp | searching.h | nodes.h

```
59              node = new_node(newData);
60              temp->next = node;
61              std::cout << "Successfully added " << node->data << " to the list." << std::endl;
62              count++;
63          }
64
65          std::cout << "Continue? (y/n): ";
66          std::cin >> choice;
67      }
68
69      // Display linked list
70      temp = head;
71      std::cout << "Linked List: ";
72      while (temp != NULL) {
73          std::cout << temp->data << " ";
74          temp = temp->next;
75      }
76      std::cout << std::endl;
77
78      // Part 5: Binary Search in Linked List
79      int itemToSearchLinkedList = 5; // Example item to search in linked list
80      binarySearchLinkedList(head, itemToSearchLinkedList);
```

Output	
Observations	Highlights the ability to search through a sorted linked list, showcasing the utility of the `getMiddle` function to facilitate efficient searching.

Table 6-3b. Binary Search for Linked List

7. Supplementary Activity

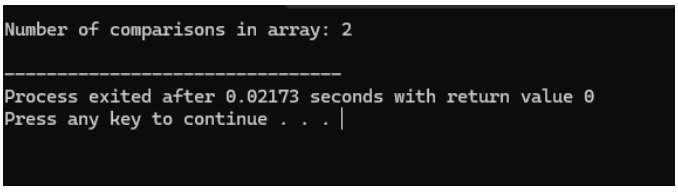
PROBLEM 1:

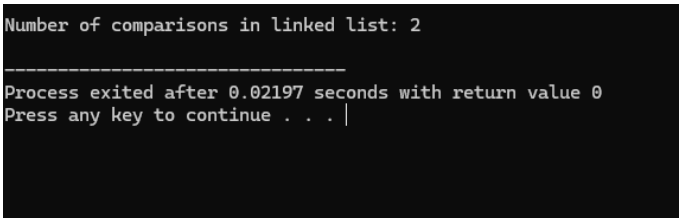
SOURCE CODE

```
#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:

SOURCE CODE

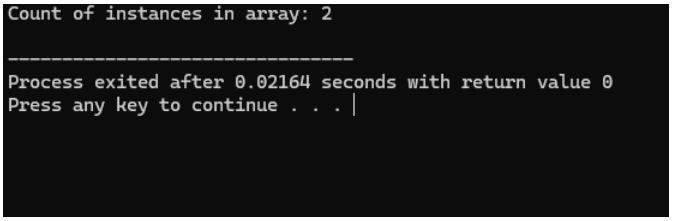
```
#include <iostream>

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;
}
```

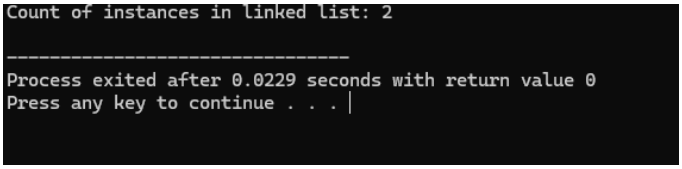
==========

```
#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

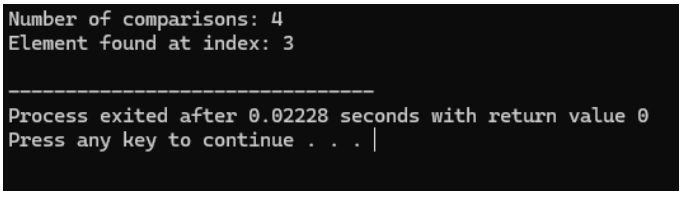




PROBLEM 3:

SOURCE 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

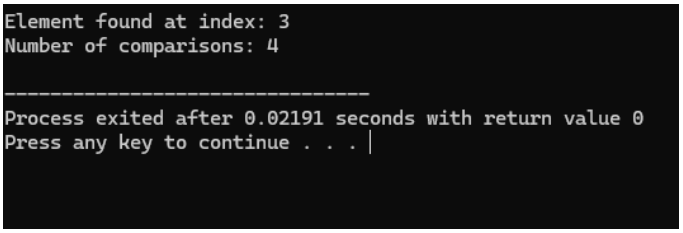
	New State: Low = 0, High = 3 Second Iteration: Low = 0, High = 3 Mid Index Calculation: mid = (0 + 3) / 2 = 1 (Value = 5) Second Iteration: Compare: arr[mid] (5) < key (8) → Move to the right New State: Low = 2, High = 3 Third Iteration: Low = 2, High = 3 Mid Index Calculation: mid = (2 + 3) / 2 = 2 (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:

SOURCE 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;
}
```

OUTPUT



8. Conclusion

Implementing sequential and binary search algorithms allowed me to understand the efficiency of these algorithms in relation to data structures like arrays and linked lists. This experience underscored the importance of choosing the right algorithm for the task and revealed the benefits of recursion, despite its complexities. Additionally, working with linked lists highlighted their flexibility for dynamic data storage compared to static arrays. While I successfully executed the algorithms and gained confidence in my programming skills, I recognize the need to improve my debugging abilities and deepen my understanding of recursion. Overall, this activity enhanced my knowledge of data structures and algorithms, motivating me to tackle future challenges in computer science.

9. Assessment Rubric