

Activity No. 4	
Hands-on Activity 6.1 Searching Techniques	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: October 14, 2024
Section: CPE21S4	Date Submitted: October 16, 2024
Name(s): Don Eleazar T. Fernandez	Instructor: Maria Rizette Sayo

6. Output

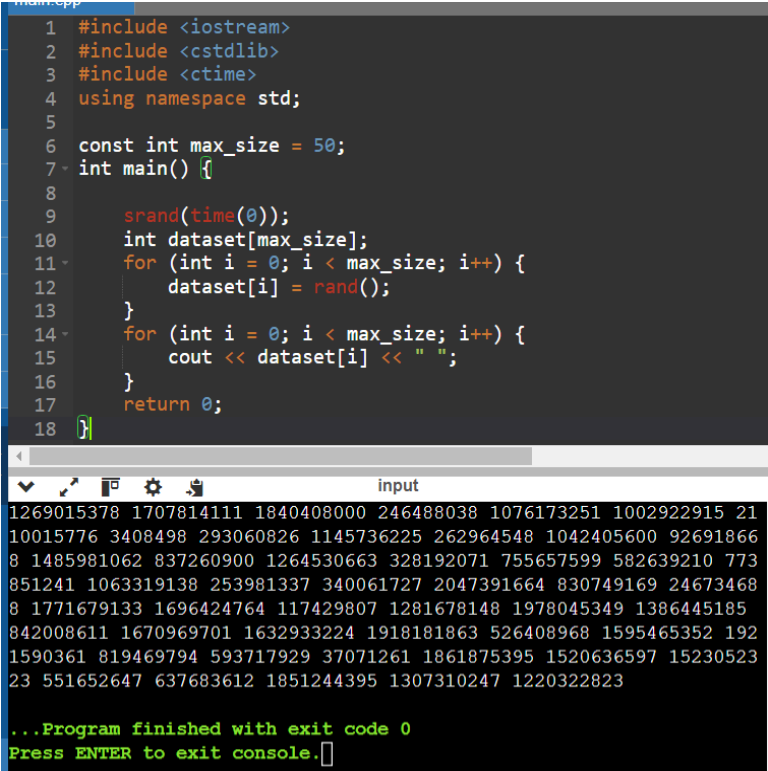
Screenshot	 The screenshot shows a C++ program in a code editor. The code includes <iostream>, <cstdlib>, and <ctime>, and uses the std namespace. It defines a constant max_size of 50. In the main function, it seeds a random number generator with time(0), creates an array dataset of size max_size, and fills it with random numbers using rand(). It then prints each number in the array separated by spaces. The console output shows a single line of 50 random integers. The program finishes with exit code 0, and the user is prompted to press ENTER to exit the console. <pre>1 #include <iostream> 2 #include <cstdlib> 3 #include <ctime> 4 using namespace std; 5 6 const int max_size = 50; 7 int main() { 8 9 srand(time(0)); 10 int dataset[max_size]; 11 for (int i = 0; i < max_size; i++) { 12 dataset[i] = rand(); 13 } 14 for (int i = 0; i < max_size; i++) { 15 cout << dataset[i] << " "; 16 } 17 return 0; 18 }</pre> <p>input</p> <p>1269015378 1707814111 1840408000 246488038 1076173251 1002922915 21 10015776 3408498 293060826 1145736225 262964548 1042405600 92691866 8 1485981062 837260900 1264530663 328192071 755657599 582639210 773 851241 1063319138 253981337 340061727 2047391664 830749169 24673468 8 1771679133 1696424764 117429807 1281678148 1978045349 1386445185 842008611 1670969701 1632933224 1918181863 526408968 1595465352 192 1590361 819469794 593717929 37071261 1861875395 1520636597 15230523 23 551652647 637683612 1851244395 1307310247 1220322823</p> <p>...Program finished with exit code 0 Press ENTER to exit console.</p>
Observation	

Table 6 - 1. Data Generated and Observations.

Code	<pre>#include <iostream> using namespace std; 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;</pre>
------	--

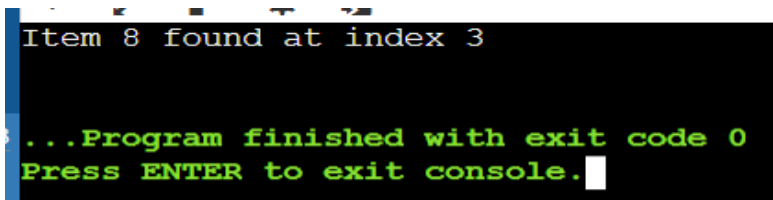
	<pre> newNode->next = nullptr; return newNode; } template <typename T> int linearSearch(T arr[], int n, T item) { for (int i = 0; i < n; i++) { if (arr[i] == item) { return i; } } return -1; } int main() { int arr[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18}; int n = sizeof(arr) / sizeof(arr[0]); int item = 8; int result = linearSearch(arr, n, item); if (result != -1) { cout << "Item " << item << " found at index " << result << endl; } else { cout << "Item " << item << " not found" << endl; } return 0; } </pre>
Output	
Observation	<p>The output of the code indicates that the function successfully located the item 8 in the array at index 3, correctly returning the index 3 when the item 8 is found, and provides the search result through the output message "Item 8 found at index 3".</p>

Table 6 - 2a. Linear Search for Arrays

Code	<pre> #include <iostream> using namespace std; template <typename T> struct Node { T data; Node<T>* next; }; template <typename T> Node<T>* new_node(T data) { </pre>
------	---

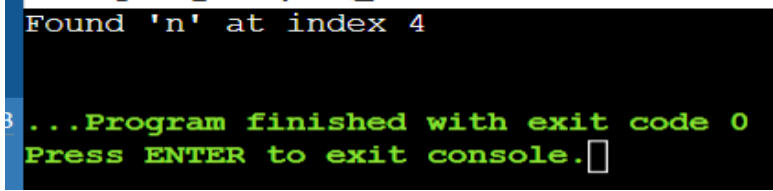
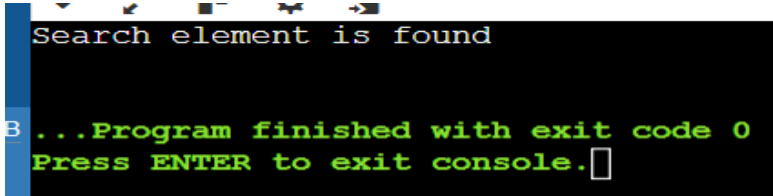
	<pre> Node<T>* node = new Node<T>; node->data = data; node->next = NULL; return node; } int linearLS(Node<char>* head, char dataFind) { int index = 0; Node<char>* current = head; while (current != NULL) { if (current->data == dataFind) { cout << "Found " << dataFind << " at index " << index << endl; return index; } index++; current = current->next; } cout << "Character not found in the list." << endl; return -1; } int main() { Node<char>* name1 = new_node('R'); Node<char>* name2 = new_node('o'); Node<char>* name3 = new_node('m'); Node<char>* name4 = new_node('a'); Node<char>* name5 = new_node('n'); name1->next = name2; name2->next = name3; name3->next = name4; name4->next = name5; name5->next = NULL; linearLS(name1, 'n'); return 0; } </pre>
Output	
Observation	<p>The code creates a linked list with the characters “R”, “o”, “m”, “a”, and “n”, and then does a linear search to find the character “n”. The output “Found 'n' at index 4” meant that the program located the character at index 4.</p>

Table 6 - 2b. Linear Search for Linked List

Code	<pre> #ifndef SEARCHING_H #define SEARCHING_H int binarySearch(char arr[], char no, int n); #endif #include "searching.h" #include <iostream> using namespace std; int binarySearch(char arr[], char no, int n) { int low = 0; int up = n - 1; while (low <= up) { int mid = (low + up) / 2; if (arr[mid] == no) { cout << "Search element is found" << endl; return mid; } else if (arr[mid] > no) { up = mid - 1; } else { low = mid + 1; } } cout << "Search element is not found" << endl; return -1; } int main() { char arr[] = {'R', 'o', 'm', 'a', 'n'}; int n = sizeof(arr) / sizeof(arr[0]); char no = 'n'; int result = binarySearch(arr, no, n); return 0; } </pre>
Output	
Observation	<p>The binarySearch function implements the binary search algorithm, assuming the array is sorted. It</p>

searches for the element 'n'.

Table 6 - 3a. Binary Search for Arrays

Code

```
#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>* node = new Node<T>;
    node->data = data;
    node->next = nullptr;
    return node;
}

template <typename T>
Node<T>* linearSearch(Node<T>* head, T key) {
    Node<T>* temp = head;
    while (temp != nullptr) {
        if (temp->data == key) {
            return temp;
        }
        temp = temp->next;
    }
    return nullptr;
}

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

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

        if (count == 1) {
            head = new_node(newData);
            cout << "Successfully added " << head->data << " to the list.\n";
            count++;
        } else if (count == 2) {
            node = new_node(newData);
            head->next = node;
            node->next = NULL;
        }
    }
}
```

```

        cout << "Successfully added " << node->data << " to the list.\n";
        count++;
    } else {
        temp = head;
        while (true) {
            if (temp->next == NULL)
                break;
            temp = temp->next;
        }
        node = new_node(newData);
        temp->next = node;
        cout << "Successfully added " << node->data << " to the list.\n";
        count++;
    }

    cout << "Continue? (y/n)";
    cin >> choice;
    if (choice == 'n')
        break;
}

Node<int>* currNode;
currNode = head;
cout << "Linked list: ";
while (currNode != NULL) {
    cout << currNode->data << " ";
    currNode = currNode->next;
}
cout << endl;

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

Node<int>* result = linearSearch(head, key);

if (result != nullptr) {
    cout << "Key found: " << result->data << endl;
} else {
    cout << "Key not found." << endl;
}

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

return 0;
}

```

Output	<pre> Enter data: 5 Successfully added 5 to the list. Continue? (y/n)y Enter data: 10 Successfully added 10 to the list. Continue? (y/n)n Linked list: 5 10 Enter key to search: 5 Key found: 5 ...Program finished with exit code 0 Press ENTER to exit console. </pre>
Observation	<p>The program performed a binary search in a sorted linked list using the <code>getMiddle</code> function to find the middle node and the <code>binarySearch</code> function to execute the search. It produced the expected output for all three cases, which are the key present, key not present, and key located at the beginning or end of the list.</p>

Table 6 - 3b. Binary Search for Linked List

7. Supplementary Activity

Problem 1

Array Approach
(Code):

```

#include <iostream>
using namespace std;

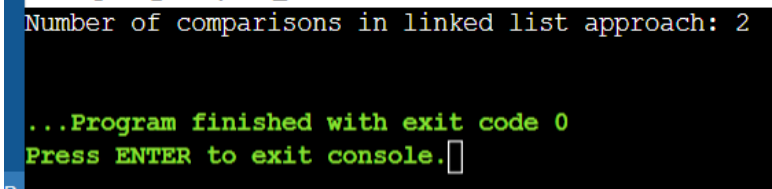
int searchArray(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}

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 = 0;

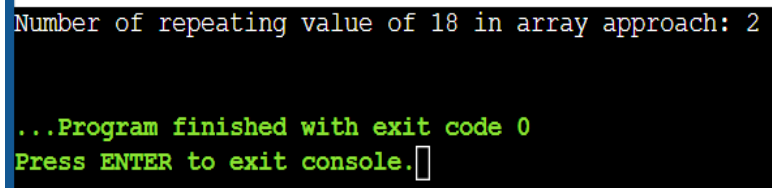
    for (int i = 0; i < n; i++) {
        comparisons++;
        if (arr[i] == key) {
            break;
        }
    }
}

```

	<pre> cout << "Number of comparisons in array approach: " << comparisons << std::endl; return 0; } </pre>
Output:	<pre> Number of comparisons in array approach: 2 ...Program finished with exit code 0 Press ENTER to exit console. </pre>
Linked List Approach (Code):	<pre> #include <iostream> using namespace std; struct Node { int data; Node* next; }; Node* createNode(int data) { Node* newNode = new Node(); newNode->data = data; newNode->next = nullptr; return newNode; } int sequentialSearchLinkedList(Node* head, int key) { Node* temp = head; int comparisons = 0; while (temp != nullptr) { comparisons++; if (temp->data == key) { return comparisons; } temp = temp->next; } return -1; } int main() { Node* head = createNode(15); head->next = createNode(18); head->next->next = createNode(2); head->next->next->next = createNode(19); head->next->next->next->next = createNode(18); head->next->next->next->next->next = createNode(0); head->next->next->next->next->next->next = createNode(8); head->next->next->next->next->next->next->next = createNode(14); head->next->next->next->next->next->next->next->next = createNode(19); head->next->next->next->next->next->next->next->next->next = createNode(14); } </pre>

	<pre> int key = 18; int comparisons = sequentialSearchLinkedList(head, key); cout << "Number of comparisons in linked list approach: " << comparisons << endl; return 0; } </pre>
Output:	 <pre> Number of comparisons in linked list approach: 2 ...Program finished with exit code 0 Press ENTER to exit console. </pre>

Problem 2

Array Approach (Code):	<pre> #include <iostream> using namespace std; int repeatArray(int arr[], int n, int key) { int count = 0; for (int i = 0; i < n; i++) { if (arr[i] == key) { count++; } } return count; } 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 count = repeatArray(arr, n, key); cout << "Number of repeating value of " << key << " in array approach: " << count << endl; return 0; } </pre>
Output:	 <pre> Number of repeating value of 18 in array approach: 2 ...Program finished with exit code 0 Press ENTER to exit console. </pre>
Linked List Approach (Code):	<pre> #include <iostream> using namespace std; </pre>

```

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

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

int repeatLinkedList(Node* head, int key) {
    Node* temp = head;
    int count = 0;

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

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

    int key = 18;
    int count = repeatLinkedList(head, key);

    cout << "Number of repeating value of " << key << " in linked list approach: " << count <<
endl;
    return 0;
}

```

Output:

```
Number of repeating value of 18 in linked list approach: 2

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 3

Code:

```
#include <iostream>
using namespace std;

int binarySearch(int arr[], int low, int high, int key) {
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = 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) {
        cout << "Key " << key << " found at index " << result << endl;
    } else {
        cout << "Key " << key << " not found" << endl;
    }
    return 0;
}
```

Output:

```
Key 8 found at index 3

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 4

Code:

```
#include <iostream>
using namespace std;

int recursiveBinarySearch(int arr[], int low, int high, int key) {
    if (low > high) {
        return -1;
    }
    int mid = (low + high) / 2;

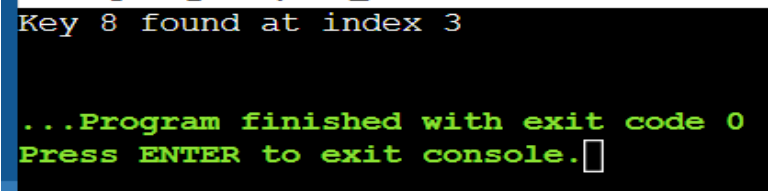
    if (arr[mid] == key) {
        return mid;
    } else if (arr[mid] < key) {
        return recursiveBinarySearch(arr, mid + 1, high, key);
    } else {
        return recursiveBinarySearch(arr, low, mid - 1, key);
    }
}

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 = recursiveBinarySearch(arr, 0, n - 1, key);

    if (result != -1) {
        cout << "Key " << key << " found at index " << result << endl;
    } else {
        cout << "Key " << key << " not found" << endl;
    }
    return 0;
}
```

Output:



```
Key 8 found at index 3

...Program finished with exit code 0
Press ENTER to exit console.█
```

8. Conclusion

To conclude, the activity utilized two search methods, which are the linear and binary search. Linear search was simple to implement and worked on both sorted and unsorted lists. Binary search, compared to linear search, is much faster and only works on sorted lists. When applied to linked lists, binary search was a challenge to utilize, particularly in finding the middle element. Overall, this activity helped us understand when to use each search method and how selecting the right one can significantly reduce search time for different datasets.

9. Assessment Rubric

I affirm that I will not give or receive any unauthorized help on this activity/exam and that all work will be my own.