

## Activity No. 6.1

<b>Course Code:</b> CPE010	<b>Program:</b> Computer Engineering
<b>Course Title:</b> Data Structures and Algorithms	<b>Date Performed:</b> 9/16/2025
<b>Section:</b> CPE21S4	<b>Date Submitted:</b> 9/16/2025
<b>Name(s):</b> Vince Gabriel Avila	<b>Instructor:</b> Engr. Jimlord M. Quejado

### 6. Output

**Table 6-1:**

```

HOA6_1.cpp nodes.h Qheader.h searching.h linearmain.cpp
1 #include <iostream>
2 #include <cstdlib> // for generating random integers
3 #include <time.h> // will be used for our seeding function
4
5 const int max_size = 50;
6
7 int main() {
8     // generate random values
9     int dataset[max_size];
10    srand(time(0));
11
12    for (int i = 0; i < max_size; i++) {
13        dataset[i] = rand();
14    }
15
16    // show your dataset's content
17    for (int i = 0; i < max_size; i++) {
18        std::cout << dataset[i] << " ";
19    }
20
21    return 0;
22 }

```

**Table 6-2a:**

```

HOA6_1.cpp nodes.h Qheader.h searching.h linearmain.cpp
1 #include "searching.h"
2
3 int main() {
4     int data[] = {3, 8, 12, 25, 30};
5     int n = 4;      // Last index
6     int item = 25;
7
8     linearSearch(data, n, item);
9
10    return 0;
11 }

```

**Table 6-2b:  
LINERAMAIN.CPP**

```

HOA6_1.cpp nodes.h searching.h linearmain.cpp
1 #include "searching.h"
2 #include "nodes.h"
3
4 int main() {
5     // Array search
6     int data[] = {3, 8, 12, 25, 30};
7     int n = 4; // Last index
8     int item = 25;
9
10    linearSearch(data, n, item);
11
12    // Linked List search for name "Roman"
13    Node<char>* name1 = new_node('R');
14    Node<char>* name2 = new_node('o');
15    Node<char>* name3 = new_node('m');
16    Node<char>* name4 = new_node('a');
17    Node<char>* name5 = new_node('n');
18
19    // Link nodes
20    name1->next = name2;
21    name2->next = name3;
22    name3->next = name4;
23    name4->next = name5;
24    name5->next = nullptr;
25
26    // Search in linked list
27    linearLS(name1, 'n'); // Should print "Searching is successful"
28    linearLS(name1, 'z'); // Should print "Searching is unsuccessful"
29
30    return 0;
31 }
32

```

es Compile Log Debug Find Results Close

D:\DATA\ANA\linearmain.exe

Searching is successful  
Searching is successful  
Searching is unsuccessful  
-----  
Process exited after 1.024 seconds with return value 0  
Press any key to continue . . .

## SEARCHING.H

```

HOA6_1.cpp nodes.h searching.h linearmain.cpp
1 ifndef SEARCHING_H
2 define SEARCHING_H
3
4 include <iostream>
5 include "nodes.h"
6
7 // Linear search for arrays
8 void linearSearch(int data[], int n, int item) {
9     int i = 0;
10    while (i <= n) {
11        if (data[i] == item) {
12            std::cout << "Searching is successful" << std::endl;
13            return;
14        }
15        i++;
16    }
17    std::cout << "Searching is unsuccessful" << std::endl;
18 }
19
20 // Linear search for linked lists
21 template <typename T>
22 void linearLS(Node<T>* head, T dataFind) {
23     Node<T>* current = head;
24     while (current != nullptr) {
25         if (current->data == dataFind) {
26             std::cout << "Searching is successful" << std::endl;
27             return;
28         }
29         current = current->next;
30     }
31     std::cout << "Searching is unsuccessful" << std::endl;
32 }
33
34 endif
35

```

NODES.H

HOA6\_1.cpp nodes.h searching.h linearmain.cpp

```
1 #ifndef NODES_H
2 #define NODES_H
3
4 template <typename T>
5 struct Node {
6     T data;
7     Node<T>* next;
8 };
9
10 template <typename T>
11 Node<T>* new_node(T newData) {
12     Node<T>* newNode = new Node<T>;
13     newNode->data = newData;
14     newNode->next = nullptr;
15     return newNode;
16 }
17
18 #endif
19
```

Table 6-3a:

CODE:

```
33
34 // Binary search for arrays (array must be sorted)
35 void binarySearch(int arr[], int n, int no) {
36     int low = 0;
37     int up = n - 1;
38
39     while (low <= up) {
40         int mid = (low + up) / 2;
41         if (arr[mid] == no) {
42             std::cout << "Search element is found!" << std::endl;
43             return;
44         } else if (no < arr[mid]) {
45             up = mid - 1;
46         } else {
47             low = mid + 1;
48         }
49     }
50
51     std::cout << "Search element is not found" << std::endl;
52 }
53
54 #endif
55
```

OUTPUT:

```
D:\DATA ANA\linarmain.exe
Searching is successful
Search element is found!
Search element is not found
Searching is successful
Searching is unsuccessful

-----
Process exited after 1.031 seconds with return value 0
Press any key to continue . . .
S
```

Table 6-3b:

MAIN:

```
nodes.h  searching.h  Main.cpp
1  #include "searching.h"
2  #include <iostream>
3
4  int main() {
5      // --- Linear search on array ---
6      int data[] = {3, 8, 12, 25, 30};
7      int n = 4; // Last index
8
9      linearSearch(data, n, 25); // Searching is successful
10     linearSearch(data, n, 7); // Searching is unsuccessful
11
12     // --- Binary search on array ---
13     binarySearch(data, n + 1, 25); // Search element is found!
14     binarySearch(data, n + 1, 7); // Search element is not found
15
16     // --- Linear search on Linked List ("Roman") ---
17     Node<char>* name1 = new_node('R');
18     Node<char>* name2 = new_node('o');
19     Node<char>* name3 = new_node('m');
20     Node<char>* name4 = new_node('a');
21     Node<char>* name5 = new_node('n');
22
23     name1->next = name2;
24     name2->next = name3;
25     name3->next = name4;
26     name4->next = name5;
27     name5->next = nullptr;
28
29     linearLS(name1, 'n'); // Searching is successful
30     linearLS(name1, 'z'); // Searching is unsuccessful
31
32     // --- Create sorted linked list for binary search ---
33     char choice = 'y';
34     int count = 1;
```

```
34     int count = 1;
35     int newData;
36     Node<int>* temp = nullptr;
37     Node<int>* head = nullptr;
38     Node<int>* node = nullptr;
39
40     std::cout << "\nEnter sorted numbers for linked list (for binary search):\n";
41     while (choice == 'y') {
42         std::cout << "Enter data: ";
43         std::cin >> newData;
44
45         if (count == 1) {
46             head = new_node(newData);
47             std::cout << "Successfully added " << head->data << " to the list.\n";
48             count++;
49         }
50         else if (count == 2) {
51             node = new_node(newData);
52             head->next = node;
53             node->next = nullptr;
54             std::cout << "Successfully added " << node->data << " to the list.\n";
55             count++;
56         }
57         else {
58             temp = head;
59             while (temp->next != nullptr) {
60                 temp = temp->next;
61             }
62             node = new_node(newData);
63             temp->next = node;
64             node->next = nullptr;
65             std::cout << "Successfully added " << node->data << " to the list.\n";
66             count++;
67         }
68     }
69 }
```

```
68
69         std::cout << "Continue? (y/n): ";
70         std::cin >> choice;
71         if (choice == 'n') break;
72     }
73
74     // Display the linked list
75     std::cout << "\nYour linked list data: ";
76     Node<int>* currNode = head;
77     while (currNode != nullptr) {
78         std::cout << currNode->data << " ";
79         currNode = currNode->next;
80     }
81     std::cout << "\n";
82
83     // Search in Linked List using binary search
84     std::cout << "\nEnter key to search in linked list: ";
85     int key;
86     std::cin >> key;
87
88     Node<int>* foundNode = binarySearchLinkedList(head, key);
89     if (foundNode != nullptr) {
90         std::cout << "Found: " << foundNode->data << std::endl;
91     }
92     else {
93         std::cout << "Not found" << std::endl;
94     }
95
96     return 0;
97 }
```

SEARCHING.H

nodes.h searching.h Main.cpp

```
1  #ifndef SEARCHING_H
2  #define SEARCHING_H
3
4  #include <iostream>
5  #include "nodes.h"
6
7  // Linear search for arrays
8  void linearSearch(int data[], int n, int item) {
9      int i = 0;
10     while (i <= n) {
11         if (data[i] == item) {
12             std::cout << "Searching is successful" << std::endl;
13             return;
14         }
15         i++;
16     }
17     std::cout << "Searching is unsuccessful" << std::endl;
18 }
19
20 // Linear search for linked lists
21 template <typename T>
22 void linearLS(Node<T>* head, T dataFind) {
23     Node<T>* current = head;
24     while (current != nullptr) {
25         if (current->data == dataFind) {
26             std::cout << "Searching is successful" << std::endl;
27             return;
28         }
29         current = current->next;
30     }
31     std::cout << "Searching is unsuccessful" << std::endl;
32 }
33
34 // Binary search for arrays (array must be sorted)
35 void binarySearch(int arr[], int n, int no) {
36     int low = 0;
37     int up = n - 1;
38
39     while (low <= up) {
40         int mid = (low + up) / 2;
41         if (arr[mid] == no) {
42             std::cout << "Search element is found!" << std::endl;
43             return;
44         } else if (no < arr[mid]) {
45             up = mid - 1;
46         } else {
47             low = mid + 1;
48         }
49     }
50
51     std::cout << "Search element is not found" << std::endl;
52 }
53
54 // Function to find the middle node between start and last (exclusive) - for linked list binary search
55 Node<int>* getMiddle(Node<int>* start, Node<int>* last) {
56     if (start == nullptr)
57         return nullptr;
58
59     Node<int>* slow = start;
60     Node<int>* fast = start->next;
61
62     while (fast != last) {
63         fast = fast->next;
64         if (fast != last) {
65             slow = slow->next;
66             fast = fast->next;
67         }
68     }
69 }
```

```
66     fast = fast->next;
67 }
68 }
69 return slow;
70 }
71
72 // Binary search on a sorted linked list
73 Node<int>* binarySearchLinkedList(Node<int>* head, int key) {
74     Node<int>* start = head;
75     Node<int>* last = nullptr;
76
77     while (start != last) {
78         Node<int>* mid = getMiddle(start, last);
79
80         if (mid == nullptr)
81             return nullptr;
82
83         if (mid->data == key)
84             return mid;
85         else if (mid->data < key)
86             start = mid->next;
87         else
88             last = mid;
89     }
90     return nullptr;
91 }
92
93 #endif
```

#### OUTPUT:

```
D:\DATA ANA\Main.exe
Searching is successful
Searching is unsuccessful
Search element is found!
Search element is not found
Searching is successful
Searching is unsuccessful

Enter sorted numbers for linked list (for binary search):
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): y3
Enter data: Successfully added 3 to the list.
Continue? (y/n): 4

Your linked list data: 1 2 3

Enter key to search in linked list: 2
Found: 2

-----
Process exited after 29.61 seconds with return value 0
Press any key to continue . . .
```

## 7. Supplementary Activity

### Searching.h

nodes.h searching.h Main.cpp

```
1 #ifndef SEARCHING_H
2 #define SEARCHING_H
3
4 #include <iostream>
5 #include "nodes.h"
6
7 // Problem 1: Linear search on array with comparison count
8 int linearSearchWithCount(int data[], int n, int item) {
9     int comparisons = 0;
10    for (int i = 0; i <= n; ++i) {
11        comparisons++;
12        if (data[i] == item) {
13            std::cout << "Searching is successful" << std::endl;
14            return comparisons;
15        }
16    }
17    std::cout << "Searching is unsuccessful" << std::endl;
18    return comparisons;
19 }
20
21 // Problem 2: Count repeating instances in array
22 int countRepeatsArray(int data[], int n, int item) {
23     int count = 0;
24     for (int i = 0; i <= n; ++i) {
25         if (data[i] == item) {
26             count++;
27         }
28     }
29     return count;
30 }
31
32 // Problem 1: Linear search on linked list with comparison count
33 template <typename T>
34 int linearLSWithCount(Node<T>* head, T dataFind) {
```

```
34  int linearLSWithCount(Node<T>* head, T dataFind) {
35      int comparisons = 0;
36      Node<T>* current = head;
37      while (current != nullptr) {
38          comparisons++;
39          if (current->data == dataFind) {
40              std::cout << "Searching is successful" << std::endl;
41              return comparisons;
42          }
43          current = current->next;
44      }
45      std::cout << "Searching is unsuccessful" << std::endl;
46      return comparisons;
47  }
48
49 // Problem 2: Count repeating instances in Linked List
50 template <typename T>
51 int countRepeatsList(Node<T>* head, T item) {
52     int count = 0;
53     Node<T>* current = head;
54     while (current != nullptr) {
55         if (current->data == item) {
56             count++;
57         }
58         current = current->next;
59     }
60     return count;
61 }
62
63 // Problem 3: Binary search with iteration output for arrays
64 void binarySearchVerbose(int arr[], int n, int no) {
65     int low = 0;
66     int up = n - 1;
67     int iteration = 1;
```

```

    while (low <= up) {
        int mid = (low + up) / 2;
        std::cout << "Iteration " << iteration++
            << ": low=" << low
            << ", up=" << up
            << ", mid=" << mid
            << ", arr[mid] =" << arr[mid] << std::endl;

        if (arr[mid] == no) {
            std::cout << "Search element is found!" << std::endl;
            return;
        } else if (no < arr[mid]) {
            up = mid - 1;
        } else {
            low = mid + 1;
        }
    }

    std::cout << "Search element is not found" << std::endl;
}

// Problem 4: Recursive binary search on array
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 (key < arr[mid]) {
        return recursiveBinarySearch(arr, low, mid - 1, key);
    } else {
        return recursiveBinarySearch(arr, mid + 1, high, key);
    }
}

```

## Nodes.h

```

#ifndef NODES_H
#define NODES_H

template <typename T>
struct Node {
    T data;
    Node<T>* next;
};

template <typename T>
Node<T>* new_node(T newData) {
    Node<T>* newNode = new Node<T>;
    newNode->data = newData;
    newNode->next = nullptr;
    return newNode;
}

#endif

```

## Main.cpp

nodes.h searching.h Main.cpp

```
1 #include "searching.h"
2 #include <iostream>
3 #include "nodes.h"
4
5 // Helper function to build Linked List from array
6 template <typename T>
7 Node<T>* buildLinkedList(T arr[], int size) {
8     if (size == 0) return nullptr;
9     Node<T>* head = new_node(arr[0]);
10    Node<T>* current = head;
11    for (int i = 1; i < size; ++i) {
12        current->next = new_node(arr[i]);
13        current = current->next;
14    }
15    return head;
16}
17
18 int main() {
19     // Problem 1 & 2 data setup
20     int data[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
21     int n = sizeof(data)/sizeof(data[0]) - 1;
22
23     // Problem 1: Linear search on array with comparisons
24     std::cout << "Problem 1 - Linear search in array for '18':\n";
25     int compsArray = linearSearchWithCount(data, n, 18);
26     std::cout << "Comparisons made: " << compsArray << "\n\n";
27
28     // Problem 1: Linear search on Linked List with comparisons
29     Node<int>* head = buildLinkedList(data, n+1);
30     std::cout << "Problem 1 - Linear search in linked list for '18':\n";
31     int compsList = linearLSWithCount(head, 18);
32     std::cout << "Comparisons made: " << compsList << "\n\n";
33
34     // Problem 2: Count repeating instances in array
```

```
34 // Problem 2: Count repeating instances in array
35 int repeatsArray = countRepeatsArray(data, n, 18);
36 std::cout << "Problem 2 - Count repeats of '18' in array: " << repeatsArray << "\n";
37
38 // Problem 2: Count repeating instances in linked list
39 int repeatsList = countRepeatsList(head, 18);
40 std::cout << "Problem 2 - Count repeats of '18' in linked list: " << repeatsList << "\n\n";
41
42 // Problem 3: Binary search verbose
43 int sortedData[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
44 int sortedSize = sizeof(sortedData)/sizeof(sortedData[0]);
45 std::cout << "Problem 3 - Iteration verbose binary search for key '8':\n";
46 binarySearchVerbose(sortedData, sortedSize, 8);
47 std::cout << "\n";
48
49 // Problem 4: Recursive binary search
50 std::cout << "Problem 4 - Recursive binary search for key '8':\n";
51 int index = recursiveBinarySearch(sortedData, 0, sortedSize - 1, 8);
52 if (index != -1) {
53     std::cout << "Search element found at index " << index << std::endl;
54 } else {
55     std::cout << "Search element not found" << std::endl;
56 }
57
58 // Clean Linked List memory
59 Node<int>* temp;
60 while (head != nullptr) {
61     temp = head;
62     head = head->next;
63     delete temp;
64 }
65
66
67 }
```

Output:

D:\DATA ANA\Main.exe

```
Problem 1 - Linear search in array for '18':  
Searching is successful  
Comparisons made: 2  
  
Problem 1 - Linear search in linked list for '18':  
Searching is successful  
Comparisons made: 2  
  
Problem 2 - Count repeats of '18' in array: 2  
Problem 2 - Count repeats of '18' in linked list: 2  
  
Problem 3 - Iteration verbose binary search for key '8':  
Iteration 1: low=0, up=9, mid=4, arr[mid]=11  
Iteration 2: low=0, up=3, mid=1, arr[mid]=5  
Iteration 3: low=2, up=3, mid=2, arr[mid]=6  
Iteration 4: low=3, up=3, mid=3, arr[mid]=8  
Search element is found!  
  
Problem 4 - Recursive binary search for key '8':  
Search element found at index 3  
  
-----  
Process exited after 1.03 seconds with return value 0  
Press any key to continue . . .
```

## 8. Conclusion

## 9. Assessment Rubric