

Activity No. 6.1

Course Code: CPE010

Program: Computer Engineering

Course Title: Data Structures and Algorithms

Date Performed: 9/16/2025

Section: CPE21S4

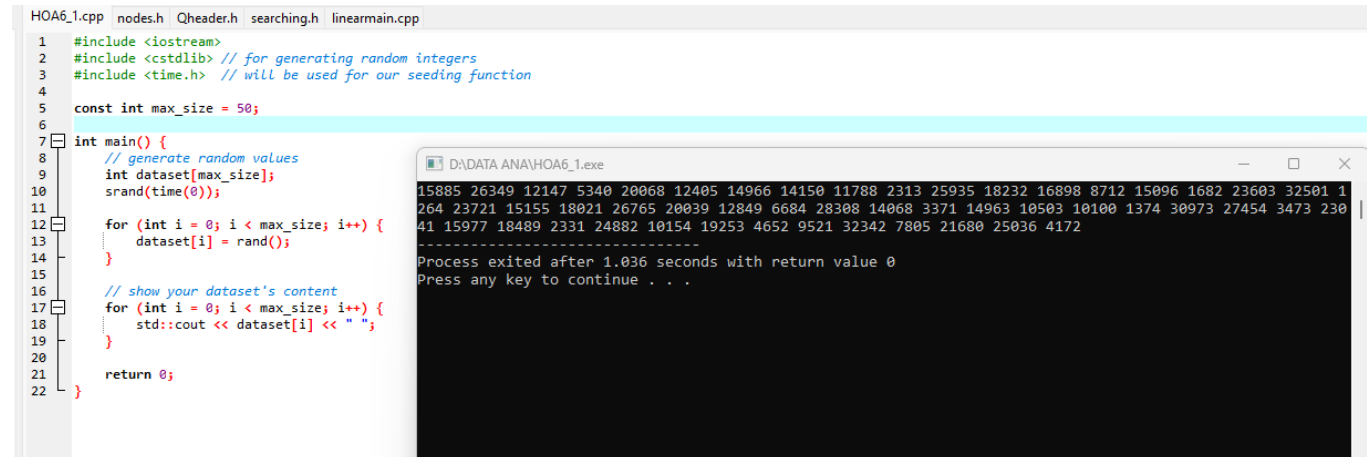
Date Submitted: 9/16/2025

Name(s): Anastacio, Lester Arvid P.

Instructor: Engr. Jimlord M. Quejado

6. Output

Table 6-1:



The screenshot shows the HOA6_1.cpp file in a code editor. The code defines a constant max_size of 50, generates a dataset of random integers, and prints the dataset's content. The execution window shows the output of the program, displaying a single line of 50 random integers.

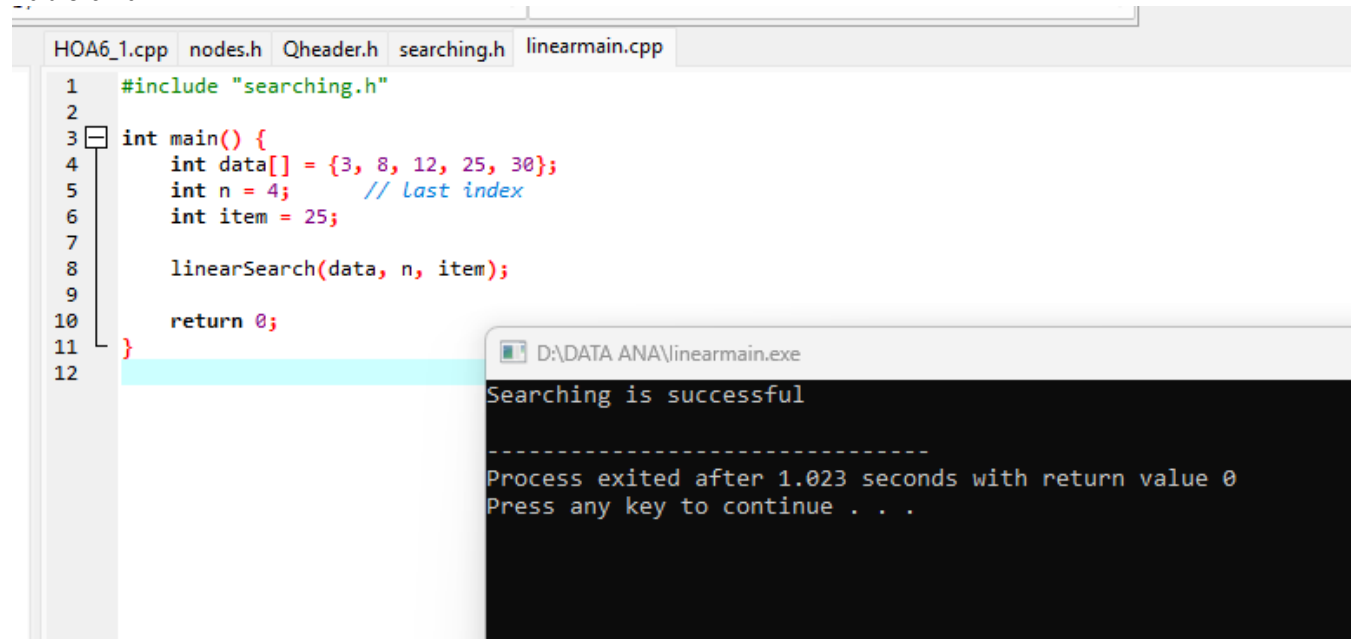
```
HOA6_1.cpp nodes.h Qheader.h searching.h linearmain.cpp
1 #include <iostream>
2 #include <cstdlib> // for generating random integers
3 #include <time.h> // will be used for our seeding function
4
5 const int max_size = 50;
6
7 int main() {
8     // generate random values
9     int dataset[max_size];
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 }
```

D:\DATA ANA\HOA6_1.exe

15885 26349 12147 5340 20068 12405 14966 14150 11788 2313 25935 18232 16898 8712 15096 1682 23603 32501 1
264 23721 15155 18021 26765 20039 12849 6684 28308 14068 3371 14963 10503 10100 1374 30973 27454 3473 230
41 15977 18489 2331 24882 10154 19253 4652 9521 32342 7805 21680 25036 4172

Process exited after 1.036 seconds with return value 0
Press any key to continue . . .

Table 6-2a:



The screenshot shows the linearmain.cpp file in a code editor. The code defines an array of integers, sets a search item, and calls the linearSearch function. The execution window shows the output of the program, displaying "Searching is successful".

```
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 }
12
```

D:\DATA ANA\linearmain.exe

Searching is successful

Process exited after 1.023 seconds with return value 0
Press any key to continue . . .

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

```

NODES.H

HOA6_1.cpp nodes.h searching.h linearmain.cpp

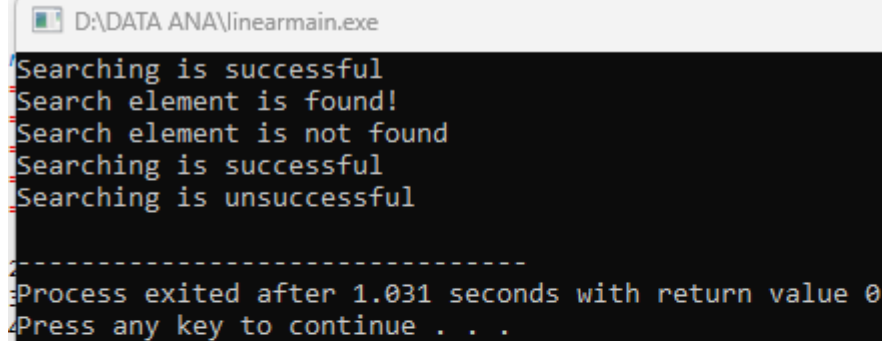
```
1  #ifndef NODES_H
2  #define NODES_H
3
4  template <typename T>
5  struct Node {
6      T data;
7      Node<T>* next;
8  };
9
10 template <typename T>
11 Node<T>* new_node(T newData) {
12     Node<T>* newNode = new Node<T>;
13     newNode->data = newData;
14     newNode->next = nullptr;
15     return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

```

Table 6-3b:

CODE:

MAIN:

```
#include "searching.h"
```

```
#include <iostream>
```

```

int main() {
    // --- Linear search on array ---
    int data[] = {3, 8, 12, 25, 30};
    int n = 4; // last index

    linearSearch(data, n, 25); // Searching is successful
    linearSearch(data, n, 7);  // Searching is unsuccessful

    // --- Binary search on array ---
    binarySearch(data, n + 1, 25); // Search element is found!
    binarySearch(data, n + 1, 7);  // Search element is not found

    // --- Linear search on linked list ("Roman") ---
    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 = nullptr;
}

```

```

linearLS(name1, 'n'); // Searching is successful
linearLS(name1, 'z'); // Searching is unsuccessful

// --- Create sorted linked list for binary search ---
char choice = 'y';
int count = 1;
int newData;
Node<int>* temp = nullptr;
Node<int>* head = nullptr;
Node<int>* node = nullptr;

std::cout << "\nEnter sorted numbers for linked list (for binary search):\n";
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 = nullptr;
        std::cout << "Successfully added " << node->data << " to the list.\n";
        count++;
    }
    else {
        temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        node = new_node(newData);
        temp->next = node;
        node->next = nullptr;
        std::cout << "Successfully added " << node->data << " to the list.\n";
        count++;
    }

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

// Display the linked list
std::cout << "\nYour linked list data: ";
Node<int>* currNode = head;
while (currNode != nullptr) {
    std::cout << currNode->data << " ";
    currNode = currNode->next;
}

```

```

std::cout << "\n";

// Search in linked list using binary search
std::cout << "\nEnter key to search in linked list: ";
int key;
std::cin >> key;

Node<int>* foundNode = binarySearchLinkedList(head, key);
if (foundNode != nullptr) {
    std::cout << "Found: " << foundNode->data << std::endl;
}
else {
    std::cout << "Not found" << std::endl;
}

return 0;
}

```

SEARCHING.H

```

#ifndef SEARCHING_H
#define SEARCHING_H

```

```

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

```

```

// Linear search for arrays
void linearSearch(int data[], int n, int item) {
    int i = 0;
    while (i <= n) {
        if (data[i] == item) {
            std::cout << "Searching is successful" << std::endl;
            return;
        }
        i++;
    }
    std::cout << "Searching is unsuccessful" << std::endl;
}

```

```

// Linear search for linked lists
template <typename T>
void linearLS(Node<T>* head, T dataFind) {
    Node<T>* current = head;
    while (current != nullptr) {
        if (current->data == dataFind) {
            std::cout << "Searching is successful" << std::endl;
            return;
        }
        current = current->next;
    }
    std::cout << "Searching is unsuccessful" << std::endl;
}

```

```

// Binary search for arrays (array must be sorted)
void binarySearch(int arr[], int n, int no) {
    int low = 0;
    int up = n - 1;

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

// Function to find the middle node between start and last (exclusive) - for linked list binary search
Node<int>* getMiddle(Node<int>* start, Node<int>* last) {
    if (start == nullptr)
        return nullptr;

    Node<int>* slow = start;
    Node<int>* fast = start->next;

    while (fast != last) {
        fast = fast->next;
        if (fast != last) {
            slow = slow->next;
            fast = fast->next;
        }
    }
    return slow;
}

// Binary search on a sorted linked list
Node<int>* binarySearchLinkedList(Node<int>* head, int key) {
    Node<int>* start = head;
    Node<int>* last = nullptr;

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

        if (mid == nullptr)
            return nullptr;

        if (mid->data == key)
            return mid;
        else if (mid->data < key)

```

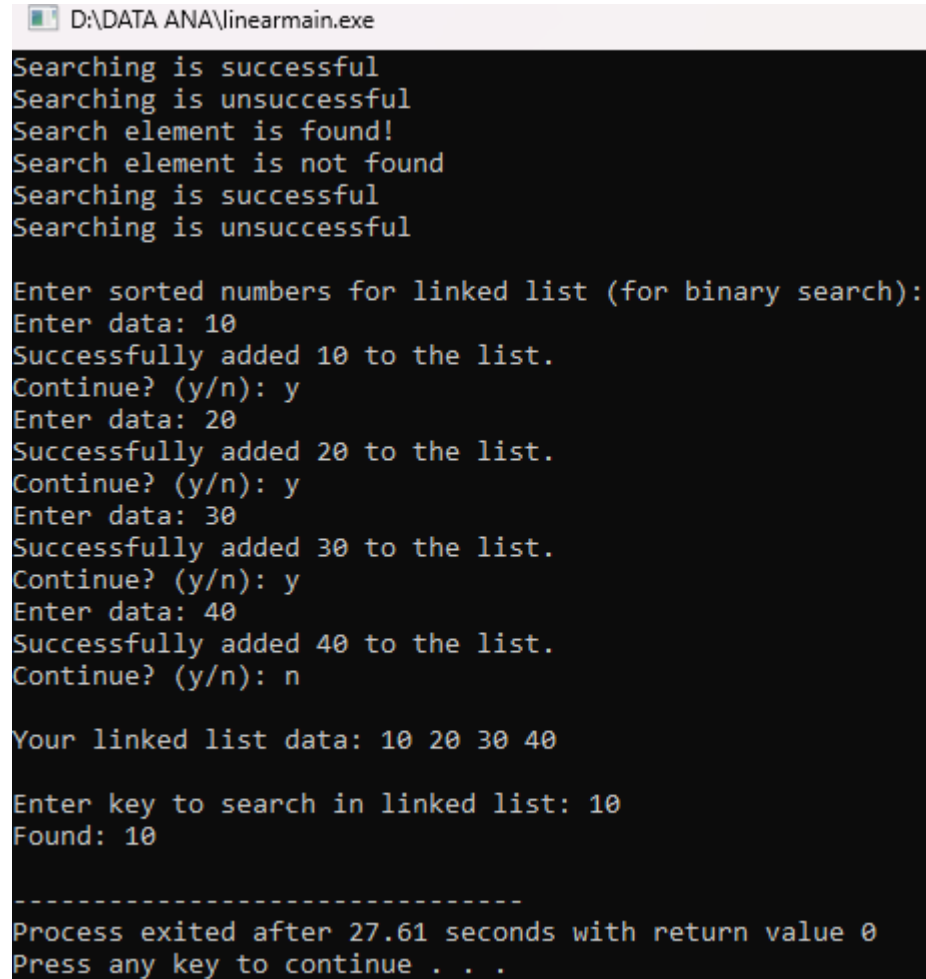
```

        start = mid->next;
    else
        last = mid;
    }
    return nullptr;
}

```

#endif

OUTPUT:



```

D:\DATA ANA\linearmain.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: 10
Successfully added 10 to the list.
Continue? (y/n): y
Enter data: 20
Successfully added 20 to the list.
Continue? (y/n): y
Enter data: 30
Successfully added 30 to the list.
Continue? (y/n): y
Enter data: 40
Successfully added 40 to the list.
Continue? (y/n): n

Your linked list data: 10 20 30 40

Enter key to search in linked list: 10
Found: 10

-----
Process exited after 27.61 seconds with return value 0
Press any key to continue . . .

```

7. Supplementary Activity

PROBLEM 1:

CODE:

MAIN.CPP

```

#include "searching.h"
#include <iostream>

```

```

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

```



```

// --- Array search with count ---
int compsArr = linearSearchWithCount(list, size - 1, 18);
std::cout << "Array comparisons: " << compsArr << std::endl;

// --- Linked list creation ---
Node<int>* head = new_node(list[0]);
Node<int>* temp = head;
for (int i = 1; i < size; ++i) {
    temp->next = new_node(list[i]);
    temp = temp->next;
}

// --- Linked list search with count ---
int compsLL = linearLSWithCount(head, 18);
std::cout << "Linked list comparisons: " << compsLL << std::endl;

return 0;
}

```

SEARCHING.H

```

#ifndef SEARCHING_H
#define SEARCHING_H

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

// --- Original linear search for arrays ---
void linearSearch(int data[], int n, int item) {
    int i = 0;
    while (i <= n) {
        if (data[i] == item) {
            std::cout << "Searching is successful" << std::endl;
            return;
        }
        i++;
    }
    std::cout << "Searching is unsuccessful" << std::endl;
}

// ? NEW: Linear search for arrays with comparison count
int linearSearchWithCount(int data[], int n, int item) {
    int comparisons = 0;
    for (int i = 0; i <= n; ++i) {
        comparisons++;
        if (data[i] == item) {
            std::cout << "Searching is successful\n";
            return comparisons;
        }
    }
    std::cout << "Searching is unsuccessful\n";
    return comparisons;
}

```

```

// --- Original linear search for linked list ---
template <typename T>
void linearLS(Node<T>* head, T dataFind) {
    Node<T>* current = head;
    while (current != nullptr) {
        if (current->data == dataFind) {
            std::cout << "Searching is successful" << std::endl;
            return;
        }
        current = current->next;
    }
    std::cout << "Searching is unsuccessful" << std::endl;
}

// ? NEW: Linear search for linked list with comparison count
template <typename T>
int linearLSWithCount(Node<T>* head, T dataFind) {
    int comparisons = 0;
    Node<T>* current = head;
    while (current != nullptr) {
        comparisons++;
        if (current->data == dataFind) {
            std::cout << "Searching is successful\n";
            return comparisons;
        }
        current = current->next;
    }
    std::cout << "Searching is unsuccessful\n";
    return comparisons;
}

// Binary search for arrays (array must be sorted)
void binarySearch(int arr[], int n, int no) {
    int low = 0;
    int up = n - 1;

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

// Function to find the middle node between start and last (exclusive) - for linked list binary search

```

```

Node<int>* getMiddle(Node<int>* start, Node<int>* last) {
    if (start == nullptr)
        return nullptr;

    Node<int>* slow = start;
    Node<int>* fast = start->next;

    while (fast != last) {
        fast = fast->next;
        if (fast != last) {
            slow = slow->next;
            fast = fast->next;
        }
    }
    return slow;
}

// Binary search on a sorted linked list
Node<int>* binarySearchLinkedList(Node<int>* head, int key) {
    Node<int>* start = head;
    Node<int>* last = nullptr;

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

        if (mid == nullptr)
            return nullptr;

        if (mid->data == key)
            return mid;
        else if (mid->data < key)
            start = mid->next;
        else
            last = mid;
    }
    return nullptr;
}

#endif

```

OUTPUT:

D:\DATA ANA\linearmain.exe

```
Searching is successful
Array comparisons: 2
Searching is successful
Linked list comparisons: 2

-----
Process exited after 1.025 seconds with return value 0
Press any key to continue . . .
```

PROBLEM 2:

MAIN:

```
#include "searching.h"
#include <iostream>

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

    int key = 18;

    // --- Count in array ---
    int repeatsArr = countRepeatsArray(list, size - 1, key);
    std::cout << "Occurrences of " << key << " in array: " << repeatsArr << std::endl;

    // --- Create linked list ---
    Node<int>* head = new_node(list[0]);
    Node<int>* temp = head;
    for (int i = 1; i < size; ++i) {
        temp->next = new_node(list[i]);
        temp = temp->next;
    }

    // --- Count in linked list ---
    int repeatsLL = countRepeatsList(head, key);
    std::cout << "Occurrences of " << key << " in linked list: " << repeatsLL << std::endl;

    return 0;
}
```

HEADER:

```
#ifndef SEARCHING_H
```

```

#define SEARCHING_H

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

// --- Original linear search for arrays ---
void linearSearch(int data[], int n, int item) {
    int i = 0;
    while (i <= n) {
        if (data[i] == item) {
            std::cout << "Searching is successful" << std::endl;
            return;
        }
        i++;
    }
    std::cout << "Searching is unsuccessful" << std::endl;
}

// ? NEW: Linear search for arrays with comparison count
int linearSearchWithCount(int data[], int n, int item) {
    int comparisons = 0;
    for (int i = 0; i <= n; ++i) {
        comparisons++;
        if (data[i] == item) {
            std::cout << "Searching is successful\n";
            return comparisons;
        }
    }
    std::cout << "Searching is unsuccessful\n";
    return comparisons;
}

// --- Original linear search for linked list ---
template <typename T>
void linearLS(Node<T>* head, T dataFind) {
    Node<T>* current = head;
    while (current != nullptr) {
        if (current->data == dataFind) {
            std::cout << "Searching is successful" << std::endl;
            return;
        }
        current = current->next;
    }
    std::cout << "Searching is unsuccessful" << std::endl;
}

// ? NEW: Linear search for linked list with comparison count
template <typename T>
int linearLSWithCount(Node<T>* head, T dataFind) {
    int comparisons = 0;
    Node<T>* current = head;
    while (current != nullptr) {
        comparisons++;

```

```

    if (current->data == dataFind) {
        std::cout << "Searching is successful\n";
        return comparisons;
    }
    current = current->next;
}
std::cout << "Searching is unsuccessful\n";
return comparisons;
}

// Binary search for arrays (array must be sorted)
void binarySearch(int arr[], int n, int no) {
    int low = 0;
    int up = n - 1;

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

// Function to find the middle node between start and last (exclusive) - for linked list binary search
Node<int>* getMiddle(Node<int>* start, Node<int>* last) {
    if (start == nullptr)
        return nullptr;

    Node<int>* slow = start;
    Node<int>* fast = start->next;

    while (fast != last) {
        fast = fast->next;
        if (fast != last) {
            slow = slow->next;
            fast = fast->next;
        }
    }
    return slow;
}

// Binary search on a sorted linked list
Node<int>* binarySearchLinkedList(Node<int>* head, int key) {
    Node<int>* start = head;
    Node<int>* last = nullptr;

```

```

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

    if (mid == nullptr)
        return nullptr;

    if (mid->data == key)
        return mid;
    else if (mid->data < key)
        start = mid->next;
    else
        last = mid;
}
return nullptr;
}

int countRepeatsArray(int data[], int n, int item) {
    int count = 0;
    for (int i = 0; i <= n; ++i) {
        if (data[i] == item) {
            count++;
        }
    }
    return count;
}

template <typename T>
int countRepeatsList(Node<T>* head, T item) {
    int count = 0;
    Node<T>* current = head;
    while (current != nullptr) {
        if (current->data == item) {
            count++;
        }
        current = current->next;
    }
    return count;
}

#endif

```

OUTPUT:

```
D:\DATA ANA\linearmain.exe
Occurrences of 18 in array: 2
Occurrences of 18 in linked list: 2

-----
Process exited after 1.027 seconds with return value 0
Press any key to continue . . .
```

PROBLEM 3:

MAIN:

```
#include "searching.h"
#include <iostream>

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

    std::cout << "Binary Search (Verbose) for key " << key << ":\n";
    binarySearchVerbose(sortedData, size, key);

    return 0;
}
```

HEADER:

```
#ifndef SEARCHING_H
#define SEARCHING_H

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

// --- Linear search for arrays ---
void linearSearch(int data[], int n, int item) {
    int i = 0;
    while (i <= n) {
        if (data[i] == item) {
            std::cout << "Searching is successful" << std::endl;
            return;
        }
        i++;
    }
    std::cout << "Searching is unsuccessful" << std::endl;
}
```



```

// --- Linear search with comparison count (for Problem 1) ---
int linearSearchWithCount(int data[], int n, int item) {
    int comparisons = 0;
    for (int i = 0; i <= n; ++i) {
        comparisons++;
        if (data[i] == item) {
            std::cout << "Searching is successful\n";
            return comparisons;
        }
    }
    std::cout << "Searching is unsuccessful\n";
    return comparisons;
}

// --- Count repeating instances in array (for Problem 2) ---
int countRepeatsArray(int data[], int n, int item) {
    int count = 0;
    for (int i = 0; i <= n; ++i) {
        if (data[i] == item) {
            count++;
        }
    }
    return count;
}

// --- Linear search for linked list ---
template <typename T>
void linearLS(Node<T>* head, T dataFind) {
    Node<T>* current = head;
    while (current != nullptr) {
        if (current->data == dataFind) {
            std::cout << "Searching is successful" << std::endl;
            return;
        }
        current = current->next;
    }
    std::cout << "Searching is unsuccessful" << std::endl;
}

// --- Linked list linear search with comparison count (for Problem 1) ---
template <typename T>
int linearLSWithCount(Node<T>* head, T dataFind) {
    int comparisons = 0;
    Node<T>* current = head;
    while (current != nullptr) {
        comparisons++;
        if (current->data == dataFind) {
            std::cout << "Searching is successful\n";
            return comparisons;
        }
        current = current->next;
    }
    std::cout << "Searching is unsuccessful\n";
}

```

```

    return comparisons;
}

// --- Count repeating instances in linked list (for Problem 2) ---
template <typename T>
int countRepeatsList(Node<T>* head, T item) {
    int count = 0;
    Node<T>* current = head;
    while (current != nullptr) {
        if (current->data == item) {
            count++;
        }
        current = current->next;
    }
    return count;
}

// --- Binary search for arrays ---
void binarySearch(int arr[], int n, int no) {
    int low = 0;
    int up = n - 1;

    while (low <= up) {
        int mid = (low + up) / 2;
        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 3: Binary search with verbose iteration output ---
void binarySearchVerbose(int arr[], int n, int no) {
    int low = 0;
    int up = n - 1;
    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;
}

// --- Get middle node for linked list binary search ---
Node<int>* getMiddle(Node<int>* start, Node<int>* last) {
    if (start == nullptr)
        return nullptr;

    Node<int>* slow = start;
    Node<int>* fast = start->next;

    while (fast != last) {
        fast = fast->next;
        if (fast != last) {
            slow = slow->next;
            fast = fast->next;
        }
    }
    return slow;
}

// --- Binary search on sorted linked list ---
Node<int>* binarySearchLinkedList(Node<int>* head, int key) {
    Node<int>* start = head;
    Node<int>* last = nullptr;

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

        if (mid == nullptr)
            return nullptr;

        if (mid->data == key)
            return mid;
        else if (mid->data < key)
            start = mid->next;
        else
            last = mid;
    }
    return nullptr;
}

#endif

```

OUTPUT:

```
D:\DATA ANA\linearmain.exe
Binary Search (Verbose) 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!

-----
Process exited after 1.028 seconds with return value 0
Press any key to continue . . .
```

PROBLEM 4:

MAIN:

```
#include "searching.h"
#include <iostream>

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

    std::cout << "Recursive Binary Search for key " << key << ":\n";
    int index = recursiveBinarySearch(sortedData, 0, size - 1, key);

    if (index != -1) {
        std::cout << "Search element is found at index " << index << std::endl;
    } else {
        std::cout << "Search element is not found" << std::endl;
    }

    return 0;
}
```

HEADER:

```
#include "searching.h"
#include <iostream>

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

```

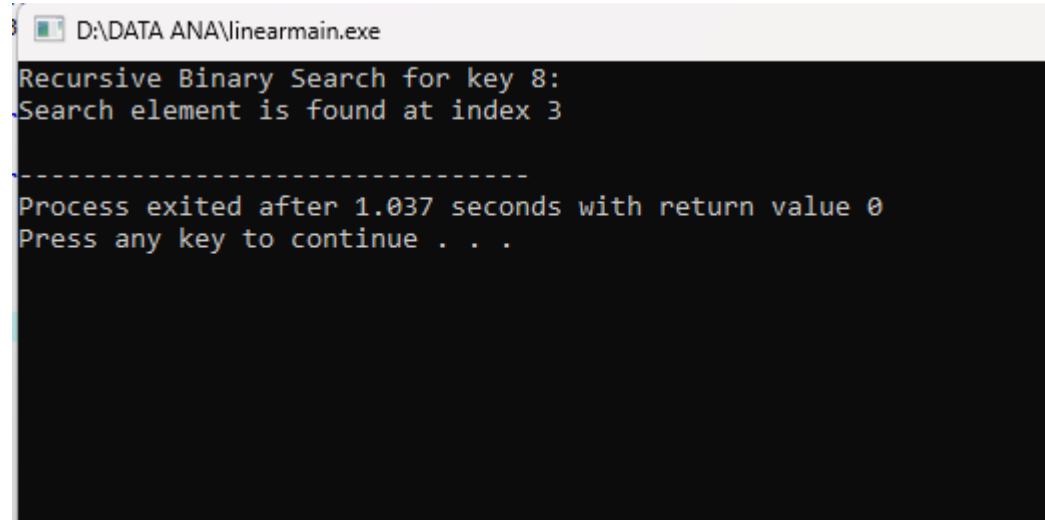
std::cout << "Recursive Binary Search for key " << key << "\n";
int index = recursiveBinarySearch(sortedData, 0, size - 1, key);

if (index != -1) {
    std::cout << "Search element is found at index " << index << std::endl;
} else {
    std::cout << "Search element is not found" << std::endl;
}

return 0;
}

```

OUTPUT:



```

D:\DATA ANA\linearmain.exe
Recursive Binary Search for key 8:
Search element is found at index 3
-----
Process exited after 1.037 seconds with return value 0
Press any key to continue . . .

```

8. Conclusion

This activity has broadened my knowledge on searching algorithms as I executed linear and binary search on both arrays and linked lists. I was able to extend a sequential search to count duplicates, which provided me with a sense on how different data structures impact search outcomes. Executing recursive binary search also made me realize how effective recursion can be in crafting solutions to computational problems.

9. Assessment Rubric