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:

HOA6_1.cpp | nodes.h | Qheader.h | searching.h | linearmain.cpp

```cpp
#include <iostream>
#include <cstdlib> // for generating random integers
#include <time.h>  // will be used for our seeding function

const int max_size = 50;

int main() {
    // generate random values
    int dataset[max_size];
    srand(time(0));

    for (int i = 0; i < max_size; i++) {
        dataset[i] = rand();
    }

    // show your dataset's content
    for (int i = 0; i < max_size; i++) {
        std::cout << dataset[i] << " ";
    }

    return 0;
}
```

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:

HOA6_1.cpp | nodes.h | Qheader.h | searching.h | linearmain.cpp

```cpp
#include "searching.h"

int main() {
    int data[] = {3, 8, 12, 25, 30};
    int n = 4;       // last index
    int item = 25;

    linearSearch(data, n, item);

    return 0;
}
```

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

```cpp
#include "searching.h"
#include "nodes.h"

int main() {
    // Array search
    int data[] = {3, 8, 12, 25, 30};
    int n = 4;  // last index
    int item = 25;

    linearSearch(data, n, item);

    // Linked list search for name "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');

    // Link nodes
    name1->next = name2;
    name2->next = name3;
    name3->next = name4;
    name4->next = name5;
    name5->next = nullptr;

    // Search in linked list
    linearLS(name1, 'n');  // Should print "Searching is successful"
    linearLS(name1, 'z');  // Should print "Searching is unsuccessful"

    return 0;
}
```

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

```cpp
#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;
}

#endif
```

NODES.H

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

```cpp
#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
```

Table 6-3a:

CODE:

```cpp

// 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;
}

#endif
```

OUTPUT:

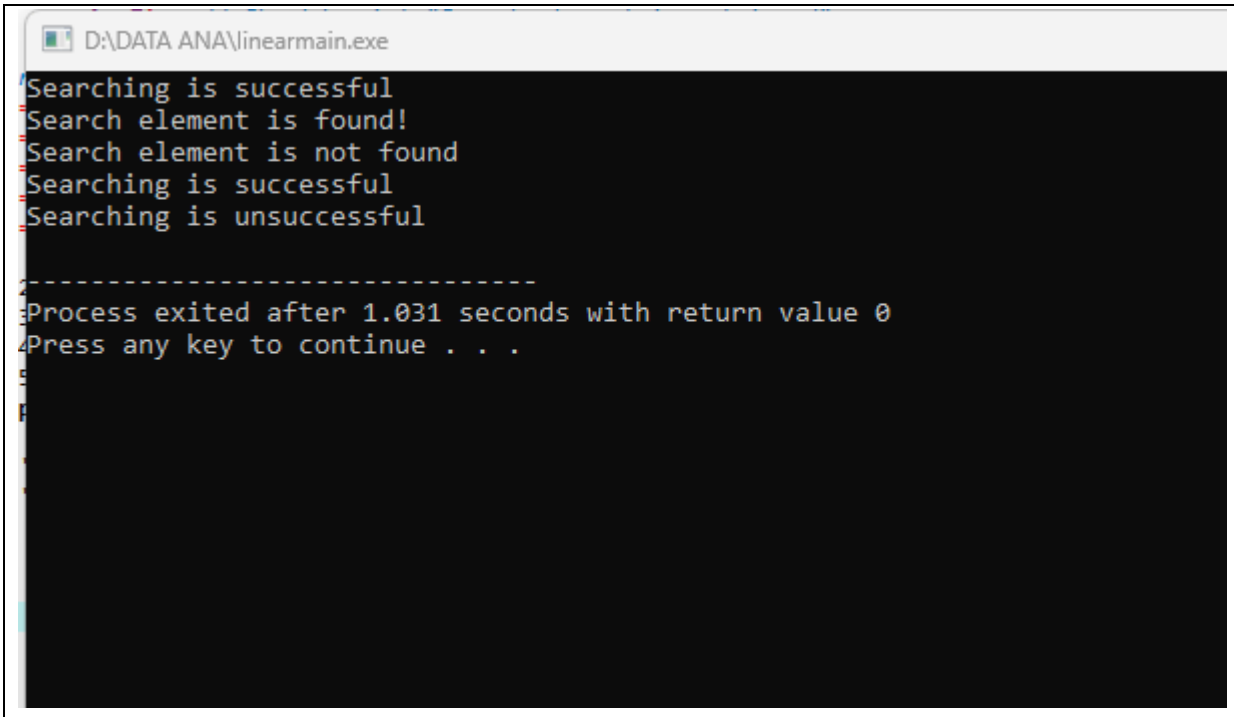


Table 6-3b:
CODE:
MAIN:

```
#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;
}
```

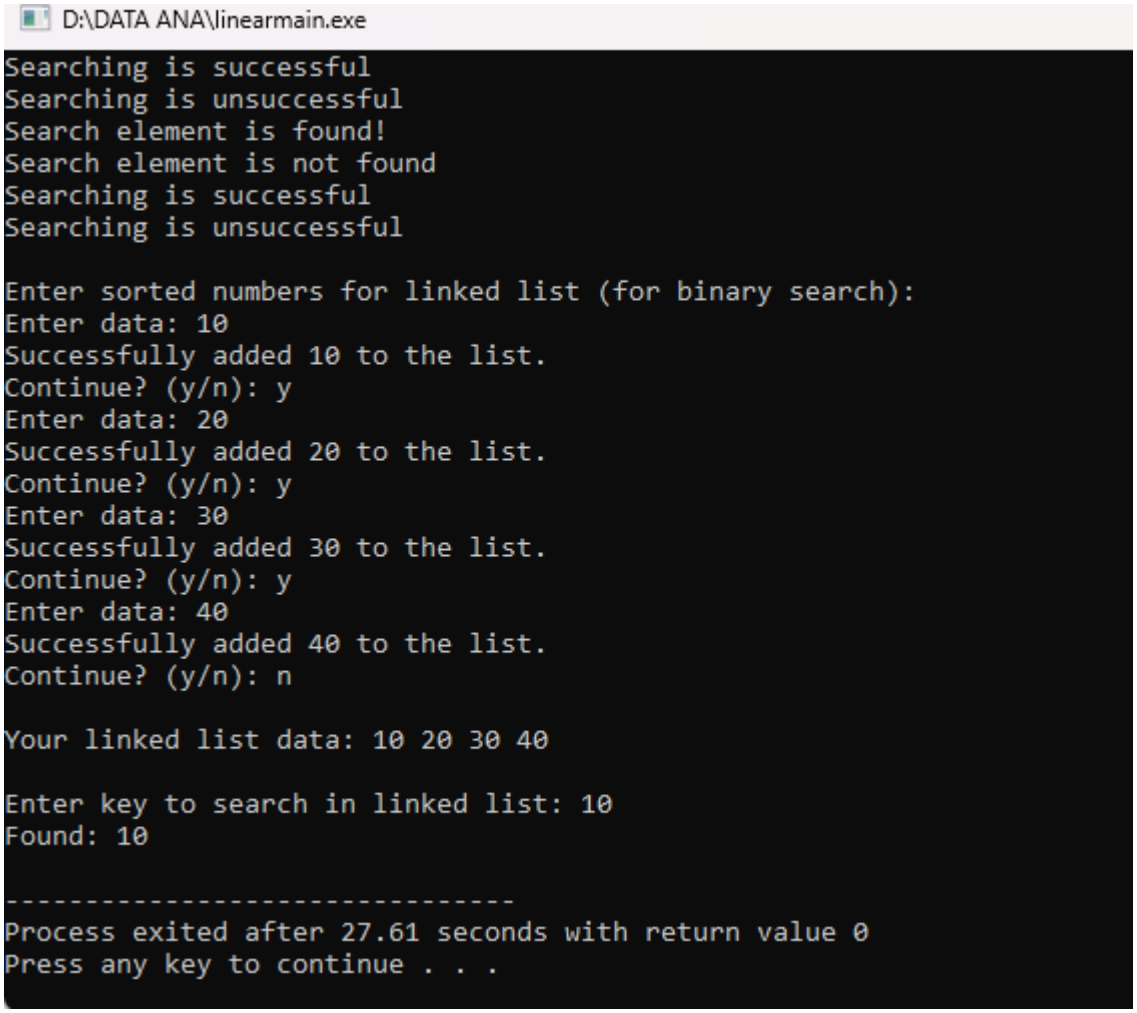
SEARCHING.H

```
#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:



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;
}
```

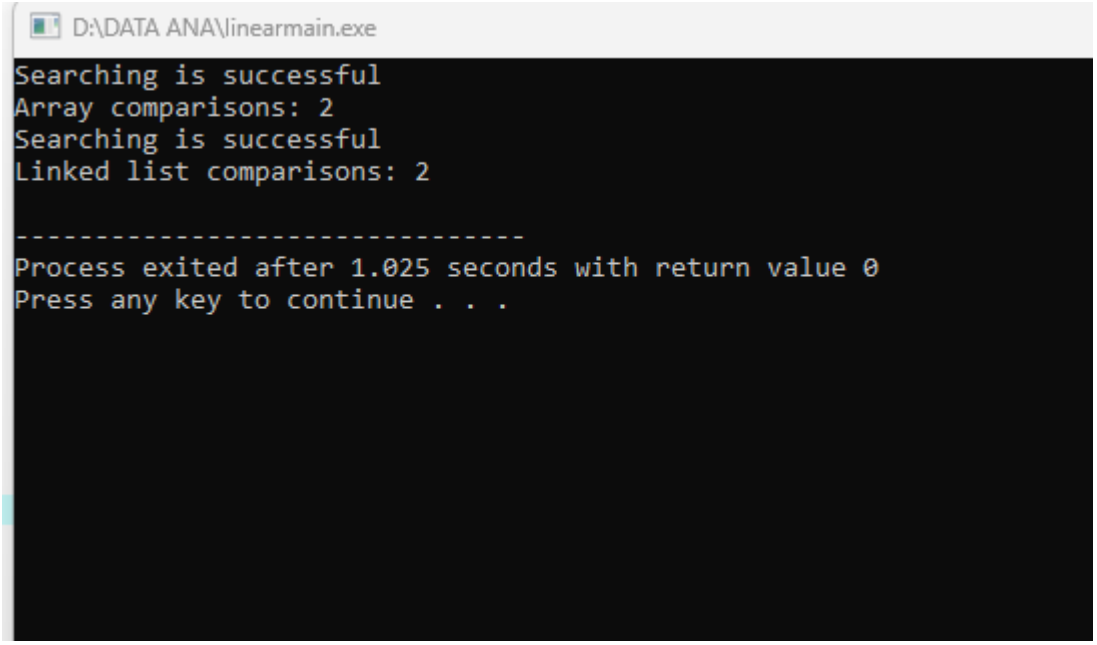
SEARCHING.H

```
#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:



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:

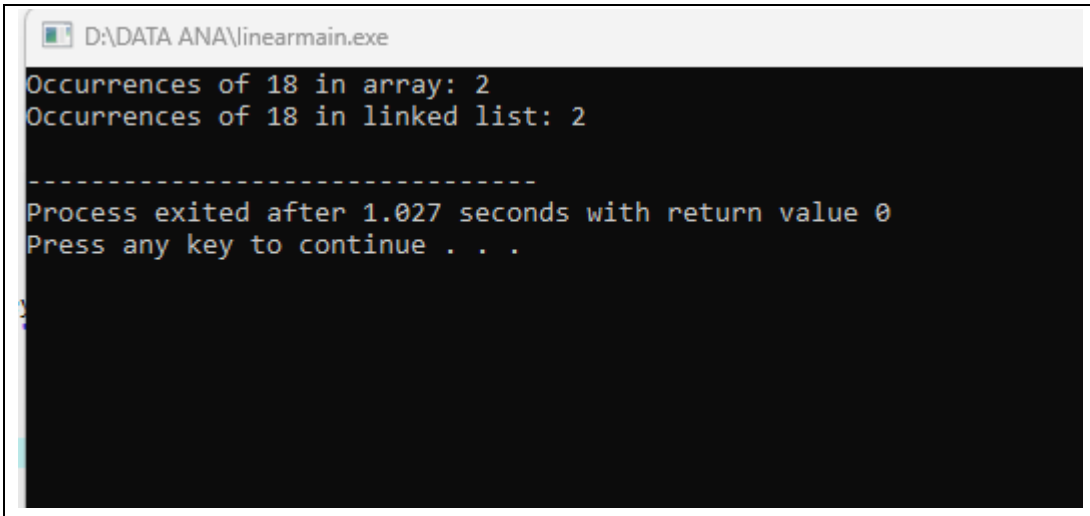
```
#ifndef SEARCHING_H
```

```
#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:



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:

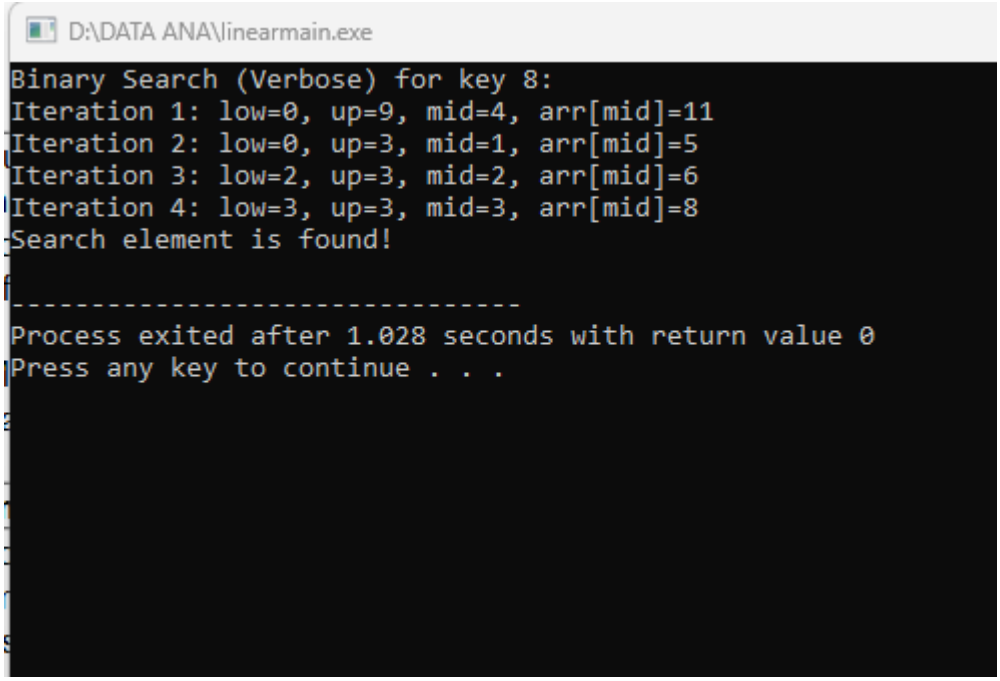
```
#ifndef SEARCHING_H
#define SEARCHING_H

#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:



PROBLEM 4:

MAIN:

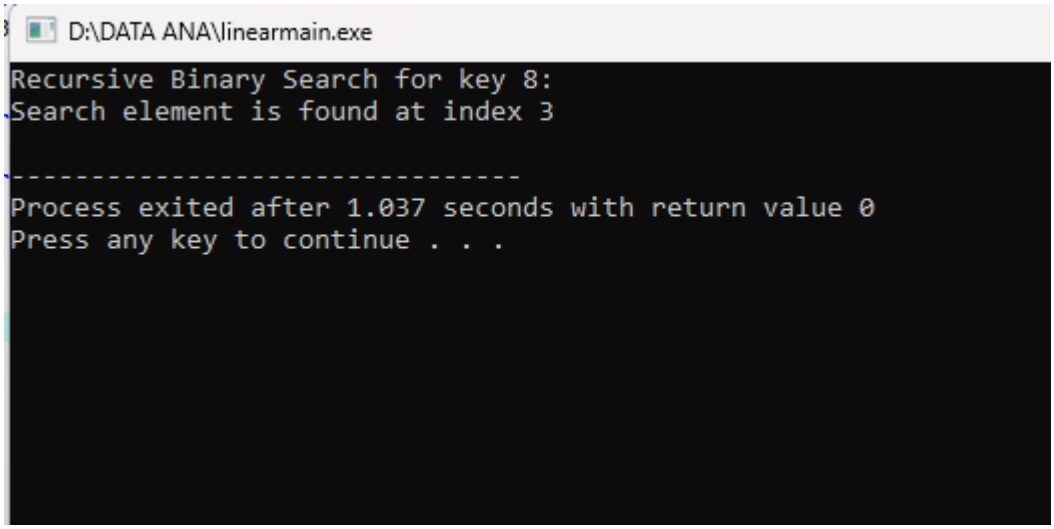
```
#include "searching.h"
#include <iostream>

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:



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