Question 1:

Design a program that incorporates a linear search algorithm to find the position of a specific number in an array. Ensure proper user input validation, such as handling non-integer inputs and avoiding array index out-of-bounds errors. Implement a function 'linear_search' that performs the linear search and returns the position if found or -1 otherwise.

Input format:

- Prompts the user to enter the size of the array: [User provides an integer input]
- Prompts the user to enter the elements of the array: [User provides space-separated integers]
- User needs to input a number to search: [User provides an integer input]

Output Format:

- The program output the original Array: [Prints the elements of the array]
- The displays the element found at position: [Prints the position if the element is found, or "Element not found..!" otherwise]

Title for Question 1: Finding Elements in an Array

Solution:

```
#include <iostream>
#include <vector>
// Function to perform linear search in an array
int linear_search(const std::vector<int>& array_nums, int val)
{
    // Iterate through each element of the array
    for (int i = 0; i < array_nums.size(); i++) {
        // Check if the current element is equal to the target value
        if (array_nums[i] == val)
            return i; // Return the position if found
    }
    return -1; // Return -1 if the element is not found in the array
}
// Main function
int main() {
    int n;
    // Input the size of the array
//    std::cout << "Enter the size of the array: ";
    std::cin >> n;
    // Input the elements of the array
//    std::cout << "Enter the elements of the array:" << std::endl;
    std::vector<int> array_nums(n);
    for (int i = 0; i < n; i++)
        std::cin >> array_nums[i];
    // Print the original array
    std::cout << "Original Array: ";
```

```
    for (int i = 0; i < n; i++)
        std::cout << array_nums[i] << " ";
    std::cout << std::endl;
    // Input a number to search
 //   std::cout << "Input a number to search: ";
    std::cin >> n;
    // Perform linear search and print the result
    int index = linear_search(array_nums, n);
    if (index != -1)
        std::cout << "\nElement found at position: " << index << std::end
    else
        std::cout << "\nElement not found..!" << std::endl;
    // Return 0 to indicate successful execution
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	4 20 15 10 5 10	Original Array: 20 15 10 5 Element found at position: 2
2	3 30 25 20 35	Original Array: 30 25 20 Element not found..!
3	6 5 10 15 20 25 30 25	Original Array: 5 10 15 20 25 30 Element found at position: 4
4	0	Original Array: Element not found..!
5	2 10 15 5	Original Array: 10 15 Element not found..!
6	8 2 4 6 8 10 12 14 16 10	Original Array: 2 4 6 8 10 12 14 16 Element found at position: 4

White List:

Black List:

Question 2:

Create a program that aims to identify all occurrences of a specific number within an array. Outline the initial steps for obtaining user input, covering both the array size and its individual elements. Following that, elucidate the fundamental structure of the linear search function, responsible for recognizing and collecting all positions where the specified number is located in the array. Consider and provide solutions for potential challenges related to user input management, such as handling non-integer entries or empty arrays.

Input Format:

- Prompts the user to enter the size of the array.
- Prompts the user to enter the elements of the array.
- Then the user needs to input the number to search.

Output Format:

- The program displays the original Array.

- Then it displays the element found at positions.

Title for Question 2: Array Element Occurrence Finder

Solution:

```
#include <iostream>
#include <vector>


// Function to perform linear search in an array and return all occurrenc
std::vector<int> linear_search_all(const std::vector<int>& array_nums, in
{
    std::vector<int> occurrences;
    // Iterate through each element of the array
    for (int i = 0; i < array_nums.size(); i++) {
        // Check if the current element is equal to the target value
        if (array_nums[i] == val)
            occurrences.push_back(i); // Collect the position if found
    }


    return occurrences; // Return all occurrences
}


// Main function
int main() {
    int n;


    // Input the size of the array
//    std::cout << "Enter the size of the array: ";
    std::cin >> n;
    // Input the elements of the array
 //   std::cout << "Enter the elements of the array:" << std::endl;
    std::vector<int> array_nums(n);
    for (int i = 0; i < n; i++)
        std::cin >> array_nums[i];


    // Print the original array
    std::cout << "Original Array: ";
    for (int i = 0; i < n; i++)
        std::cout << array_nums[i] << " ";
    std::cout << std::endl;
    // Input a number to search
//    std::cout << "Input a number to search: ";
    std::cin >> n;
    // Perform linear search and print the result
    std::vector<int> occurrences = linear_search_all(array_nums, n);
    if (!occurrences.empty()) {
        std::cout << "\nElement found at positions: ";
        for (int pos : occurrences)
            std::cout << pos << " ";
        std::cout << std::endl;
    } else {
```

```
        std::cout << "\nElement not found..!" << std::endl;
    }


    // Return 0 to indicate successful execution
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	7 5 10 15 5 20 25 5 5	Original Array: 5 10 15 5 20 25 5 Element found at positions: 0 3 6
2	4 10 20 30 40 25	Original Array: 10 20 30 40 Element not found..!
3	6 8 15 8 12 20 8 8	Original Array: 8 15 8 12 20 8 Element found at positions: 0 2 5
4	3 5 5 5	Original Array: 5 5 5 Element not found..!
5	5 1 2 3 4 5 6	Original Array: 1 2 3 4 5 Element not found..!
6	4 10 20 30 10 10	Original Array: 10 20 30 10 Element found at positions: 0 3

White List:

Black List:

Question 3:

creating a user-friendly program for efficient array searching. Begin by allowing users to easily input the array size and elements. Craft functions for both linear and binary searches, providing straightforward exploration of search techniques. Consider potential challenges with user inputs, ensuring simplicity and validation.

Input Format:

- Prompts the user to enter the size of the array.
- Prompts the user to enter the elements of the array.
- Prompts the user to enter the target for linear search.
- Prompts the user to enter the target for binary search.

Output Format:

- The Program outputs the result of Linear Search: Element found at index number (or) Element not found.
- Then it outputs the result of Binary Search: Element found at index number (or) Element not found.

Title for Question 3: Linear and Binary Adventures

Solution:

```cpp
#include <iostream>
#include <vector>


// Linear Search
int linear_search(const std::vector<int>& array, int target)
 {
    for (int i = 0; i < array.size(); ++i) {
        if (array[i] == target) {
            return i; // Return the index if target is found
        }
    }
    return -1; // Return -1 if target is not found
}


// Binary Search (Assumes array is sorted)
int binary_search(const std::vector<int>& array, int target) {
    int low = 0;
    int high = array.size() - 1;


    while (low <= high) {
        int mid = low + (high - low) / 2;


        if (array[mid] == target) {
            return mid; // Return the index if target is found
        } else if (array[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }


    return -1; // Return -1 if target is not found
}
int main() {
    int size;
//    std::cout << "Enter the size of the array: ";
    std::cin >> size;
    std::vector<int> userArray(size);
  //  std::cout << "Enter the elements of the array:" << std::endl;
    for (int i = 0; i < size; ++i) {
        std::cin >> userArray[i];
    }
    int target;
    // Linear Search
//    std::cout << "Enter the target for linear search: ";
    std::cin >> target;
    int linearResult = linear_search(userArray, target);
    if (linearResult != -1) {
        std::cout << "Linear Search: Element found at index " << linearRe
    } else {
        std::cout << "Linear Search: Element not found." << std::endl;
    }
    // Binary Search
//    std::cout << "Enter the target for binary search: ";
    std::cin >> target;
```

```
    int binaryResult = binary_search(userArray, target);
    if (binaryResult != -1) {
        std::cout << "Binary Search: Element found at index " << binaryRe
    } else {
        std::cout << "Binary Search: Element not found." << std::endl;
    }
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 10 5 20 15 25 20 15	Linear Search: Element found at index 2 Binary Search: Element not found.
2	4 30 25 30 40 25 30	Linear Search: Element found at index 1 Binary Search: Element found at index 2
3	6 8 15 8 12 20 8 8 15	Linear Search: Element found at index 0 Binary Search: Element not found.
4	3 5 5 5 5 3	Linear Search: Element found at index 0 Binary Search: Element not found.
5	5 1 2 3 4 5 6 3	Linear Search: Element not found. Binary Search: Element found at index 2
6	4 10 20 30 10 10 25	Linear Search: Element found at index 0 Binary Search: Element not found.

White List:

Black List:

Question 4:

Develop a program implementing the binary search algorithm to locate the index of a target element within a sorted array. This program dynamically acquires user input for the array size and elements, showcasing an interactive and adaptable design. Integrate technical terms such as "algorithmic efficiency," "sorted data structure," and "logarithmic time complexity" in your inquiry.

Input Format:

- Prompts the user to enter the size of the sorted array: [user inputs array size]
- Prompts the user to enter the elements of the sorted array:
- Prompts the user to enter the target element: [user inputs target element]

Output Format:

- The program outputs the lement found at index: [index] OR Element not found.

Title for Question 4: Finding the Index of a Target Element in a Sorted Array

Solution:

```
#include <iostream>
#include <vector>
int binarySearch(const std::vector<int>& sortedArray, int target) {
    int low = 0;
    int high = sortedArray.size() - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (sortedArray[mid] == target)
            return mid; // Target found
        else if (sortedArray[mid] < target)
            low = mid + 1; // Adjust the search range to the right
        else
            high = mid - 1; // Adjust the search range to the left
    }
    return -1; // Target not found
}
int main() {
    int size;
    // Input the size of the sorted array
//    std::cout << "Enter the size of the sorted array: ";
    std::cin >> size;
    // Input the elements of the sorted array
    std::vector<int> sortedArray(size);
//    std::cout << "Enter the elements of the sorted array:" << std::endl
    for (int i = 0; i < size; ++i) {
        std::cin >> sortedArray[i];
    }
    int target;
    // Input the target element
//    std::cout << "Enter the target element: ";
    std::cin >> target;
    // Perform binary search and print the result
    int result = binarySearch(sortedArray, target);
    if (result != -1)
        std::cout << "Element found at index: " << result << std::endl;
    else
        std::cout << "Element not found." << std::endl;
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 10 20 30 40 50 30	Element found at index: 2
2	4 5 15 25 35 20	Element not found.
3	3 10 20 30 15	Element not found.
4	6 2 4 6 8 10 12 7	Element not found.
5	5 100 200 300 400 500 500	Element found at index: 4
6	2 30 40 20	Element not found.

White List:

Black List:

Question 5:

Embark on implementing a program that simulates a number-guessing game. Your program should prompt the user to input a range for the game. Subsequently, it randomly selects a target number within that range. Employ the binary search algorithm to guide the user in guessing the correct number. Utilize vectors for efficient storage and manipulation of numbers within the specified range. Incorporate error-handling mechanisms for invalid inputs, ensuring a smooth user experience. Provide an overview of the algorithmic and structural decisions you would make in the initial stages of designing and implementing this interactive guessing game.

Input Format:

- The program expects two integers as input, separated by space: rangeStart and rangeEnd.
- The range [rangeStart, rangeEnd] defines the set of numbers among which the middle number is chosen as the target.

Output Format:

- The program outputs messages guiding the user to guess the middle number within the specified range.
- After each guess, the program provides feedback (higher, lower, or correct) and prompts the user to try again if the guess is incorrect.
- Upon correctly guessing the middle number, the program congratulates the user and displays the number of attempts made.

Note:

- Guess number must me enter Numeric input otherwise its show "Invalid input. Thank you.".
- Guess number must enter the within limitation otherwise its show "Incorrect guess. Try again!".
- Entered guess number is below the target number to print "The correct number is higher.".
- Entered guess number is higher then target number to print "The correct number is lower.".
- Entered guess number is exactly target number to print " Congratulations! You guessed the correct number in 'number of attempt ' attempts.

Title for Question 5: Interactive Number Guessing Game

Solution:

```
#include <iostream>
#include <vector>
#include <limits>
#include <cstdlib>
#include <ctime>
using namespace std;
```

```
int binarySearch(const vector<int>& arr, int target) {
    int low = 0;
    int high = arr.size() - 1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1; // Target not found
}
int main() {
    srand(time(0));
    int rangeStart, rangeEnd;
 //   cout << "Enter the range of numbers (start end): ";
    cin >> rangeStart >> rangeEnd;
    if (rangeEnd <= rangeStart) {
        cout << "Invalid input. End range must be greater than start rang
        return 1;
    }
    int middleNumber = (rangeStart + rangeEnd) / 2;
    vector<int> numbers;
    for (int i = rangeStart; i <= rangeEnd; ++i) {
        numbers.push_back(i);
    }
    int targetNumber = middleNumber;
    int attempts = 0;
    int guess;
 //   cout << "Guess the middle number between " << rangeStart << " and "
    do {
        if (!(cin >> guess)) {
            cout << "Invalid input. Please enter a valid integer.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }
        attempts++;
        int result = binarySearch(numbers, guess);
        if (result == -1) {
            cout << "Incorrect guess. Try again!\n";
        } else if (numbers[result] < targetNumber) {
            cout << "The correct number is higher.\n";
        } else if (numbers[result] > targetNumber) {
            cout << "The correct number is lower.\n";
        } else {
            cout << "Congratulations! You guessed the correct number in "
        }
    } while (guess != targetNumber);
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs

1	1 20 15 10	The correct number is lower. Congratulations! You guessed the correct number in 2 attempts.
2	2 5 3	Congratulations! You guessed the correct number in 1 attempts.
3	1 10 6 7 5	The correct number is lower. The correct number is lower. Congratulations! You guessed the correct number in 3 attempts.
4	15 30 23 22	The correct number is lower. Congratulations! You guessed the correct number in 2 attempts.
5	10 50 20 40 30	The correct number is higher. The correct number is lower. Congratulations! You guessed the correct number in 3 attempts.
6	2 7 8 4	Incorrect guess. Try again! Congratulations! You guessed the correct number in 2 attempts.

White List:

Black List:

Question 6:

You've been assigned a task to implement a binary search algorithm to search for a target number within a sorted array. Your implementation should allow users to input the size of the array, its elements, and the target number. Additionally, it should provide feedback on whether the target number is found within the array or not.

Input Format:

- The program prompts the user to enter the size of the array: "Enter the size of the array: "
- The second line instructs the user to input the array elements in ascending order separated by spaces: "Enter the elements of the sorted array: "
- The third line asks the user to input the target number to search for: "Enter the target number to search for: "

Output Format:

- If the target number is found in the array, the program outputs: "The search number is found in the array."
- If the target number is not found, the output is: "The search number does not exist in the array."

Title for Question 6: Recursive Binary Search in Sorted Array

Solution:

```
#include <iostream>

int binarySearch(int arr[], int n, int target, int low, int high) {
```

```
    int mid, result = 0;

    if (low <= high) {
        mid = (low + high) / 2;


        if (target == arr[mid]) {
            result = 1;
        } else if (target < arr[mid]) {
            return binarySearch(arr, n, target, low, mid - 1);
        } else {
            return binarySearch(arr, n, target, mid + 1, high);
        }
    }


    return result;
}


int main() {
    int arr[10], n, target, low, high;
    std::cin >> n;
    for (int i = 0; i < n; i++) {
        std::cin >> arr[i];
    }
    std::cin >> target;
    low = 0, high = n - 1;
    int result = binarySearch(arr, n, target, low, high);
    if (result == 0)
        std::cout << "The search number does not exist in the array.\n\n"
    else
        std::cout << "The search number is found in the array.\n\n";
    return 0;
}
```

TestCases:

S.No	Inputs	Outputs
1	5 1 2 3 4 5 3	The search number is found in the array.
2	7 10 20 30 40 50 60 70 25	The search number does not exist in the array.
3	3 -5 0 5 0	The search number is found in the array.
4	4 12 34 56 78 90	The search number does not exist in the array.
5	6 8 16 24 32 40 48 32	The search number is found in the array.
6	2 5 10 8	The search number does not exist in the array.

White List:

Black List:
