Marwadi University Marwadi Chandarana Group NAAC A+	Marwadi University Faculty of Engineering and Technology Department of Information and Communication Technology	
Subject: DAA (01CT0512)	**AIM: Implementing the Searching Algorithms and understanding the time and space complexities.**	
Experiment No: 2	**Date:**	**Enrolment No: 92301733034**

1. Linear Search:

Theory:

- Linear Search scans each element of the array one by one until it finds the target element. It does not require the array to be sorted.

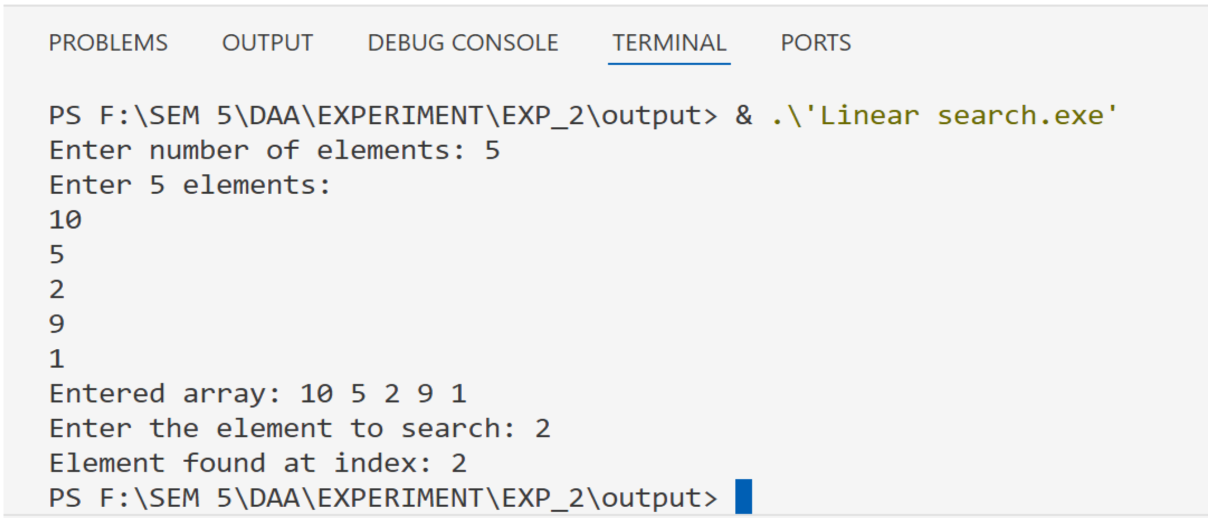
Programming Language: C++

Code:

```
#include <iostream>
using namespace std;
int linearSearch(int arr[], int n, int key) {
    for(int i = 0; i < n; i++) {
        if(arr[i] == key)
            return i;
    }
    return -1;
}
int main() {
    int n, key;
    cout << "Enter number of elements: ";
    cin >> n;

    int arr[n];
    cout << "Enter " << n << " elements:\n";
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << "Entered array: ";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << "\nEnter the element to search: ";
    cin >> key;
    int result = linearSearch(arr, n, key);
    if(result != -1)
        cout << "Element found at index: " << result <<
endl;
    else
        cout << "Element not found." << endl;
    return 0;
}
```

Output:



```
PS F:\SEM 5\DAA\EXPERIMENT\EXP_2\output> & .\'Linear search.exe'
Enter number of elements: 5
Enter 5 elements:
10
5
2
9
1
Entered array: 10 5 2 9 1
Enter the element to search: 2
Element found at index: 2
PS F:\SEM 5\DAA\EXPERIMENT\EXP_2\output>
```

1. **Space complexity: O(1)**

   Justification: Binary Search uses only a few variables (`low`, `high`, `mid`) and does not require any additional space.

2. **Time complexity:**
   - **Best case time complexity:** O(1)
     Justification: If the target is the middle element, it's found immediately in the first comparison.
   - **Worst case time complexity:** O(log n)
     Justification: The search range is halved each time, so the maximum number of comparisons required is $\log_2(n)$.

Marwadi University Marwadi Chandarana Group NAAC A+	**Marwadi University** **Faculty of Engineering and Technology** **Department of Information and Communication Technology**	
Subject: DAA (01CT0512)	**AIM: Implementing the Searching Algorithms and understanding the time and space complexities.**	
Experiment No: 2	**Date:**	**Enrolment No: 92301733034**

2. Binary Search:

Theory:

- Binary Search is a fast and efficient search algorithm used on **sorted arrays**. It works by repeatedly dividing the array in half and comparing the target value with the middle element. If the target is equal to the middle, it returns the index. If it is less, it continues on the left half; otherwise, it moves to the right half.

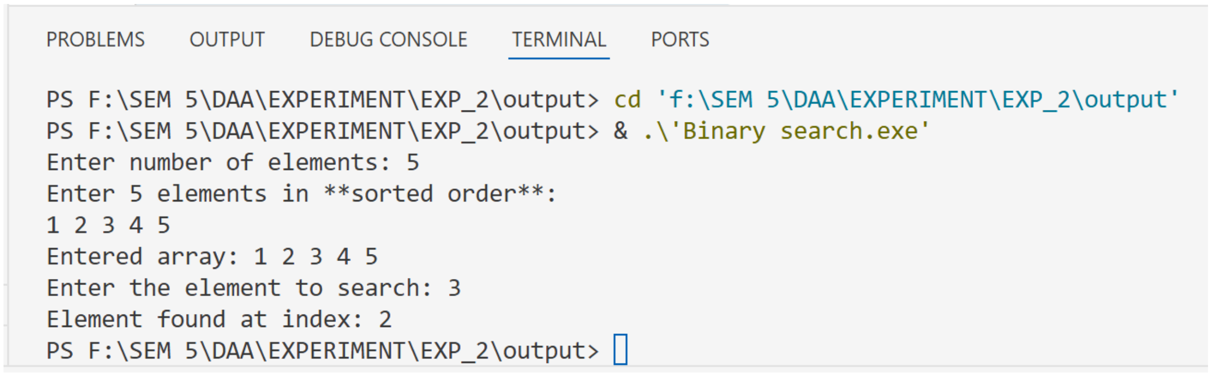
Programming Language: C++

Code:

```
#include <iostream>
using namespace std;
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;
    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 n, key;
    cout << "Enter number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " elements in **sorted
order**:\n";
    for(int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    cout << "Entered array: ";
    for(int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << "\nEnter the element to search: ";
    cin >> key;
    int result = binarySearch(arr, n, key);
    if(result != -1)
        cout << "Element found at index: " << result <<
endl;
    else
        cout << "Element not found." << endl;
    return 0;
}
```

Output:



```
PS F:\SEM 5\DAA\EXPERIMENT\EXP_2\output> cd 'f:\SEM 5\DAA\EXPERIMENT\EXP_2\output'
PS F:\SEM 5\DAA\EXPERIMENT\EXP_2\output> & .\'Binary search.exe'
Enter number of elements: 5
Enter 5 elements in **sorted order**:
1 2 3 4 5
Entered array: 1 2 3 4 5
Enter the element to search: 3
Element found at index: 2
PS F:\SEM 5\DAA\EXPERIMENT\EXP_2\output>
```

3. **Space complexity: O(1)**

   Justification: Only a few variables are used (`low`, `high`, `mid`), so no extra space is required beyond the input array.

4. **Time complexity:**
   - **Best case time complexity:** O(1)
     Justification: If the element is found at the middle in the first comparison.
   - **Worst case time complexity:** O(log n)
     Justification: Each step reduces the array size by half. So, at most $\log_2(n)$ steps are needed.