

 <b>Marwadi</b> <b>University</b> Marwadi Chandarana Group	<b>Marwadi University</b> <b>Faculty of Engineering and Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject:</b> Design & Analysis of Algorithms (01CT0512)	<b>Aim:</b> Implementing the Searching Algorithms and understanding the time and space complexities	
<b>Experiment No:</b> 2	<b>Date:</b> 03/08/2025	<b>Enrollment No:</b> 92301733049

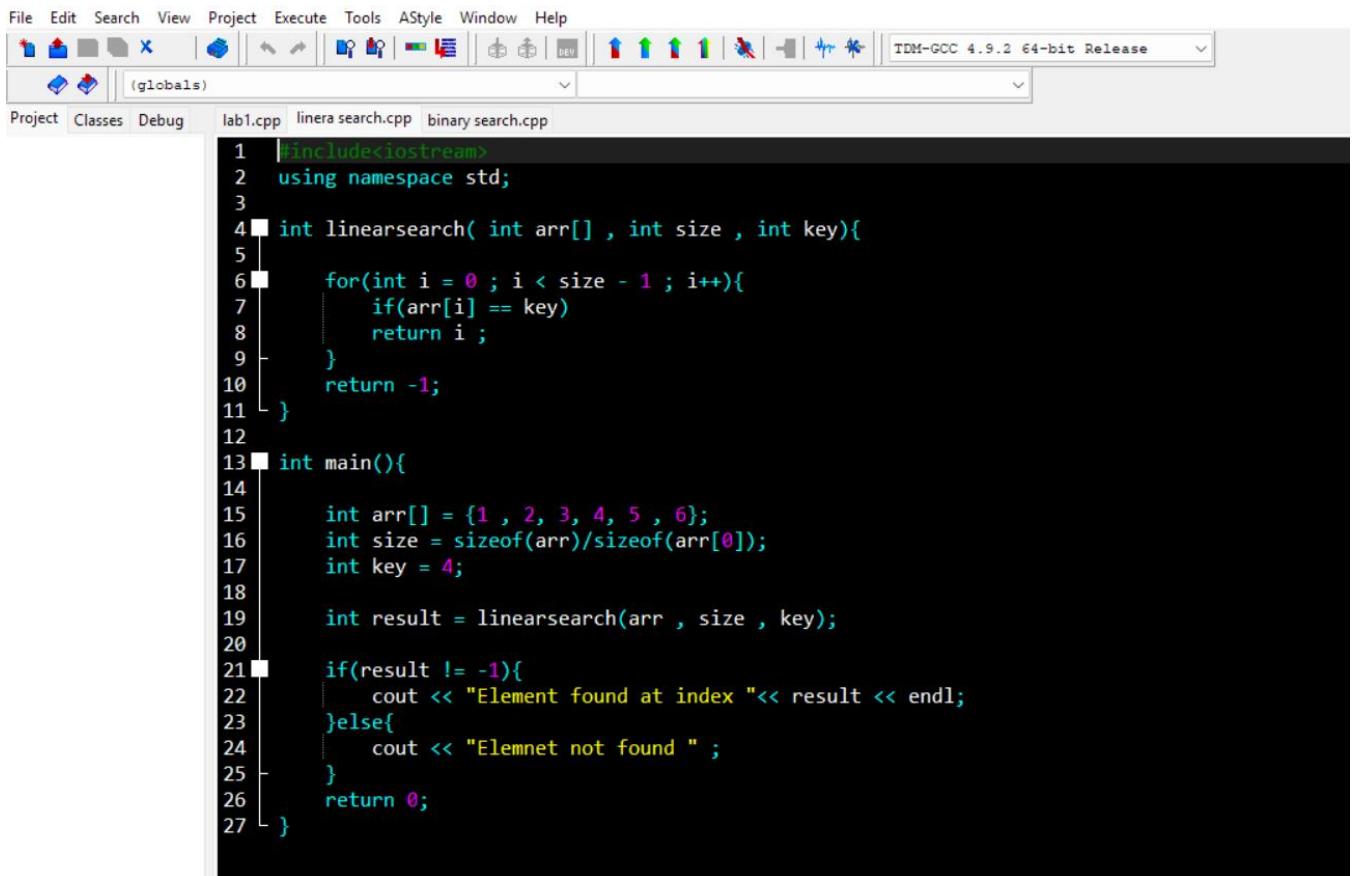
## 1. Linear Search

### Theory

In Linear Search, we traverse the array from the beginning and compare each element with the target value. If a match is found, we return the index; otherwise, continue until the end of the array.

### Programming Language: C++

#### Code:



```

File Edit Search View Project Execute Tools AStyle Window Help
File Project Class Debug lab1.cpp linear search.cpp binary search.cpp TDM-GCC 4.9.2 64-bit Release
(globals) Project Classes Debug lab1.cpp linear search.cpp binary search.cpp TDM-GCC 4.9.2 64-bit Release
1 #include<iostream>
2 using namespace std;
3
4 int linearsearch( int arr[] , int size , int key){
5
6   for(int i = 0 ; i < size - 1 ; i++){
7     if(arr[i] == key)
8       return i ;
9   }
10  return -1;
11 }
12
13 int main(){
14
15   int arr[] = {1 , 2, 3, 4, 5 , 6};
16   int size = sizeof(arr)/sizeof(arr[0]);
17   int key = 4;
18
19   int result = linearsearch(arr , size , key);
20
21   if(result != -1){
22     cout << "Element found at index " << result << endl;
23   }else{
24     cout << "Element not found " ;
25   }
26   return 0;
27 }

```

#### Output:

 <b>Marwadi</b> University Marwadi Chandarana Group	<b>Marwadi University</b> <b>Faculty of Engineering and Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject:</b> Design & Analysis of Algorithms (01CT0512)	<b>Aim:</b> Implementing the Searching Algorithms and understanding the time and space complexities	
<b>Experiment No:</b> 2	<b>Date:</b> 03/08/2025	<b>Enrollment No:</b> 92301733049

```
Element found at index 3
```

```
-----
Process exited after 0.3427 seconds with return value 0
Press any key to continue . . . |
```

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

**Justification:**

Linear search operates directly on the input array without using any extra data structures, so it uses constant auxiliary space.

**Time complexity:**

– **Best case time complexity:** O(1)

**Justification:**

If the element to be found is at the first position, we find it in the first comparison.

– **Worst case time complexity:** O(n)

**Justification:**

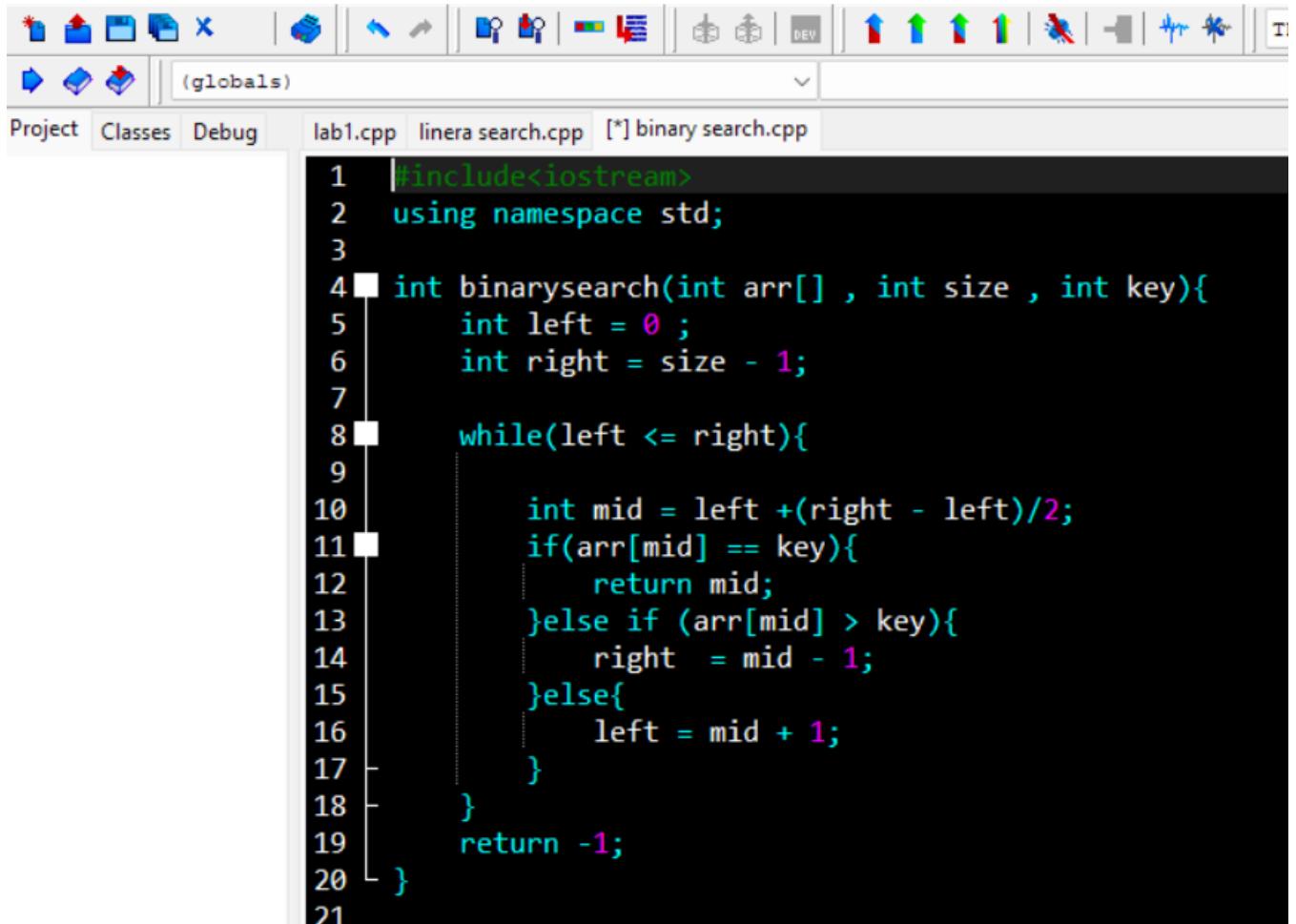
If the element is at the last position or not present in the array, we have to check every element once, leading to linear time.

 <b>Marwadi</b> University Marwadi Chandarana Group	<b>Marwadi University</b> <b>Faculty of Engineering and Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject:</b> Design & Analysis of Algorithms (01CT0512)	<b>Aim:</b> Implementing the Searching Algorithms and understanding the time and space complexities	
<b>Experiment No:</b> 2	<b>Date:</b> 03/08/2025	<b>Enrollment No:</b> 92301733049

## 2. Binary Search

### Theory

**Binary Search** is applied on a **sorted** array. It repeatedly divides the search interval in half. If the value of the search key is less than the item in the middle, it narrows the interval to the lower half. Otherwise, to the upper half.



The screenshot shows a C++ development environment with the following interface elements:

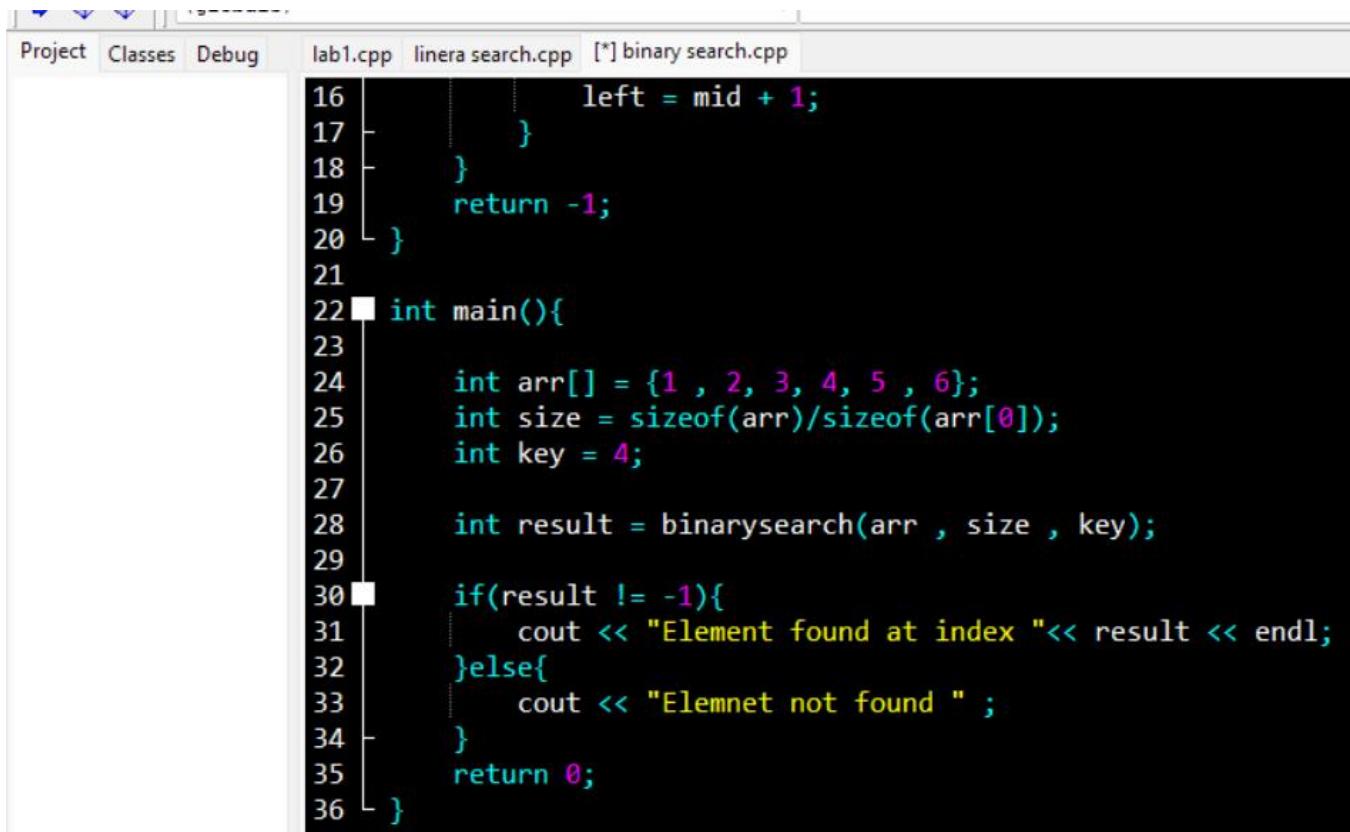
- Toolbar:** Standard icons for file operations (New, Open, Save, Close), project management, and code navigation.
- Global Tools:** A dropdown menu labeled "(globals)" containing various tools and settings.
- Project Bar:** Shows the current project structure with files: lab1.cpp, linear search.cpp, and binary search.cpp.
- Code Editor:** Displays the C++ code for a binary search algorithm. The code uses standard input-output streams and implements a recursive search function.

```

1 #include<iostream>
2 using namespace std;
3
4 int binarysearch(int arr[], int size , int key){
5     int left = 0 ;
6     int right = size - 1;
7
8     while(left <= right){
9
10        int mid = left +(right - left)/2;
11        if(arr[mid] == key){
12            return mid;
13        }else if (arr[mid] > key){
14            right = mid - 1;
15        }else{
16            left = mid + 1;
17        }
18    }
19    return -1;
20}
21

```

 <b>Marwadi</b> University Marwadi Chandarana Group	<b>Marwadi University</b> <b>Faculty of Engineering and Technology</b> <b>Department of Information and Communication Technology</b>
<b>Subject:</b> Design & Analysis of Algorithms (01CT0512)	<b>Aim:</b> Implementing the Searching Algorithms and understanding the time and space complexities
<b>Experiment No:</b> 2	<b>Date:</b> 03/08/2025 <b>Enrollment No:</b> 92301733049

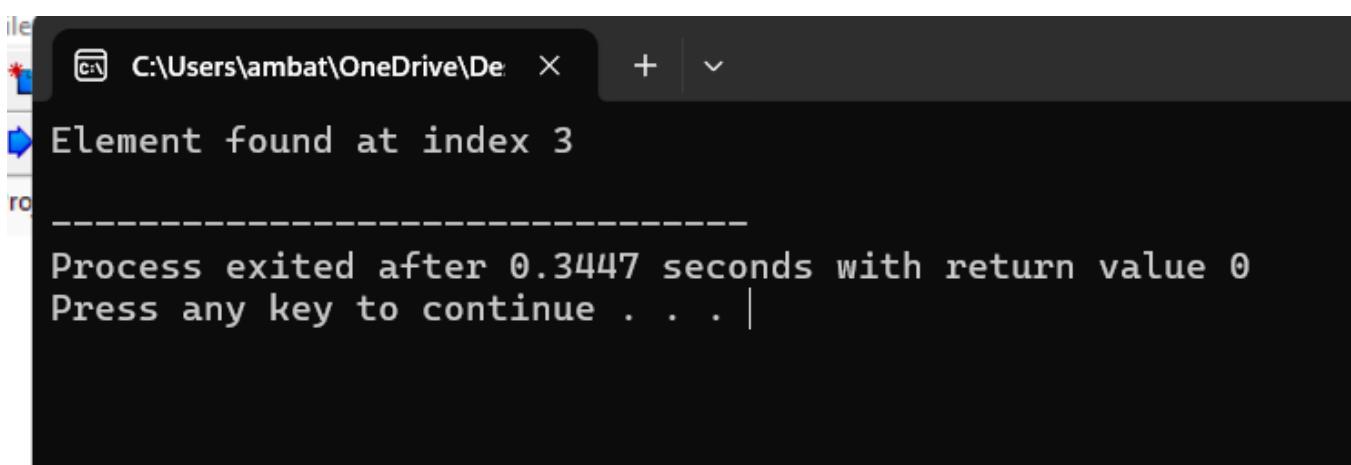


```

Project Classes Debug lab1.cpp linear search.cpp [*] binary search.cpp
16     left = mid + 1;
17 }
18 }
19 return -1;
20 }
21
22 int main(){
23
24     int arr[] = {1 , 2, 3, 4, 5 , 6};
25     int size = sizeof(arr)/sizeof(arr[0]);
26     int key = 4;
27
28     int result = binarysearch(arr , size , key);
29
30 if(result != -1){
31     cout << "Element found at index " << result << endl;
32 }else{
33     cout << "Element not found " ;
34 }
35 return 0;
36 }

```

OUTPUT :-



```

C:\Users\ambat\OneDrive\Desktop\lab1> Element found at index 3
-----
Process exited after 0.3447 seconds with return value 0
Press any key to continue . . .

```

 <b>Marwadi University</b> Marwadi Chandarana Group	<b>Marwadi University</b> <b>Faculty of Engineering and Technology</b> <b>Department of Information and Communication Technology</b>	
<b>Subject:</b> Design & Analysis of Algorithms (01CT0512)	<b>Aim:</b> Implementing the Searching Algorithms and understanding the time and space complexities	
<b>Experiment No:</b> 2	<b>Date:</b> 03/08/2025	<b>Enrollment No:</b> 92301733049

**Space complexity:**

$O(1)$  (for iterative version)

**Justification:**

Binary search only uses a few variables (like low, high, mid) for the search and does not require any extra space beyond that.

**Time complexity:**

– Best case time complexity:  $O(1)$

**Justification:**

If the target element is at the middle index during the first check, it's found immediately.

– Worst case time complexity:  $O(\log n)$

**Justification:**

Each step of the binary search cuts the problem size in half. In the worst case, we keep dividing until one element is left, resulting in  $\log_2(n)$  comparisons.