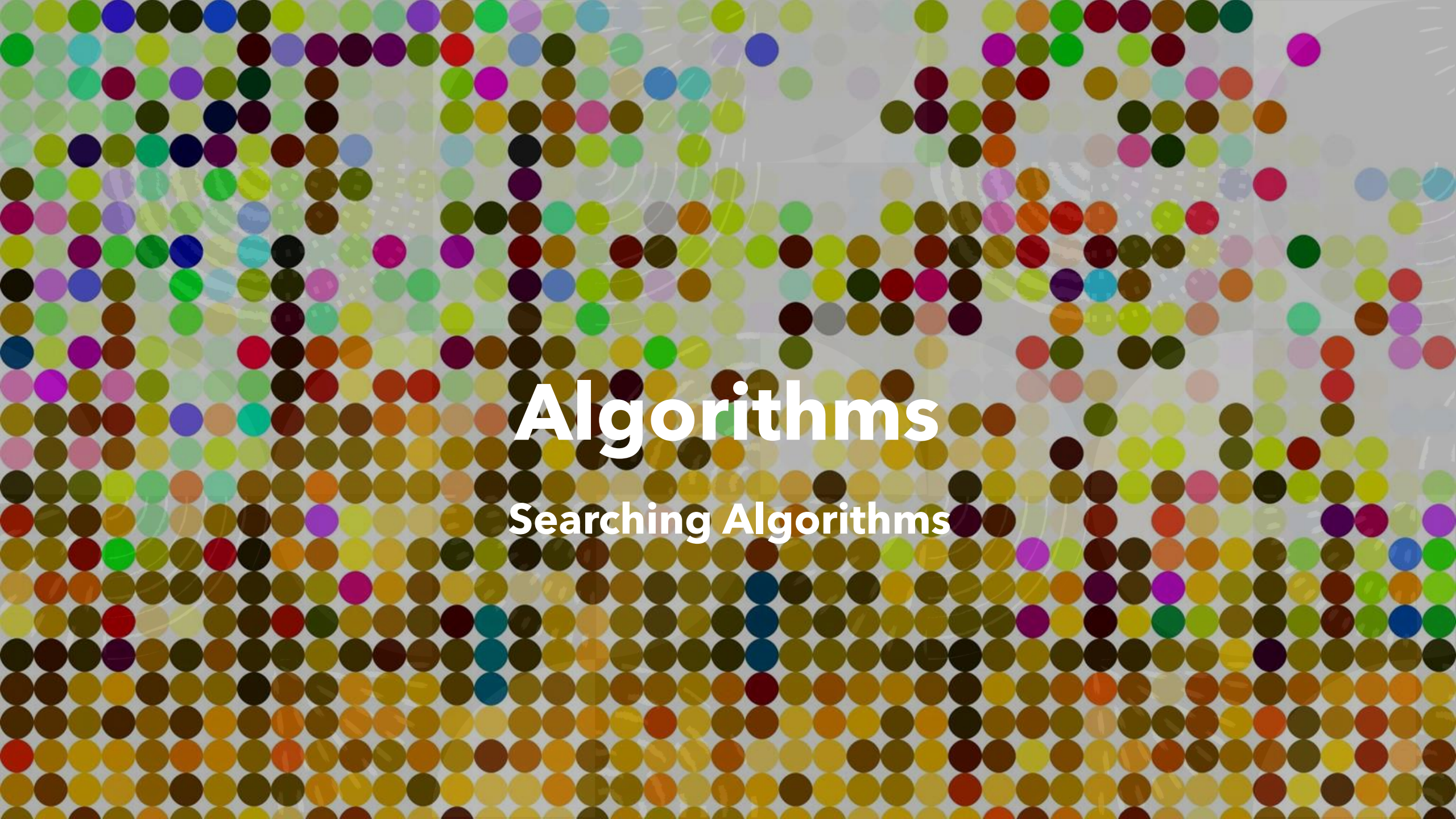

Algorithms
Searching Algorithms

What is Searching algorithm?

Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored.

Based on the type of search operation, these algorithms are generally classified into **two categories**:

Searching algorithms types

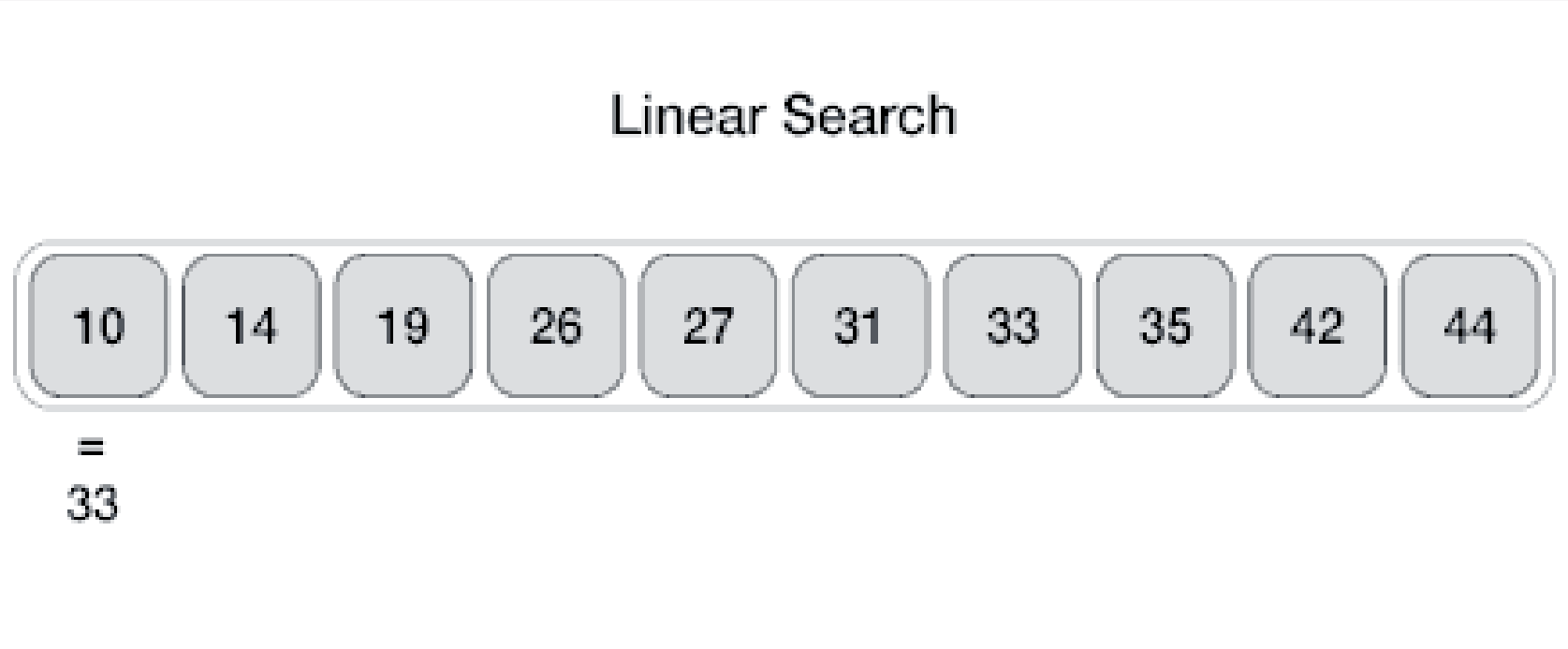
- **Sequential Search:** In this, **the list or array is traversed sequentially**, and every element is checked. For example: Linear Search.

- **Interval Search:** These algorithms are specifically designed for **searching in sorted data-structures**. These type of searching algorithms are much more efficient than **Linear Search** as they repeatedly target the center of the search structure and divide the search space in half. For Example: **Binary Search**.

Linear Search

Linear search is a sequential searching algorithm where we **start from one end and check every element of the list until the desired element is found**.

It is the simplest searching algorithm.

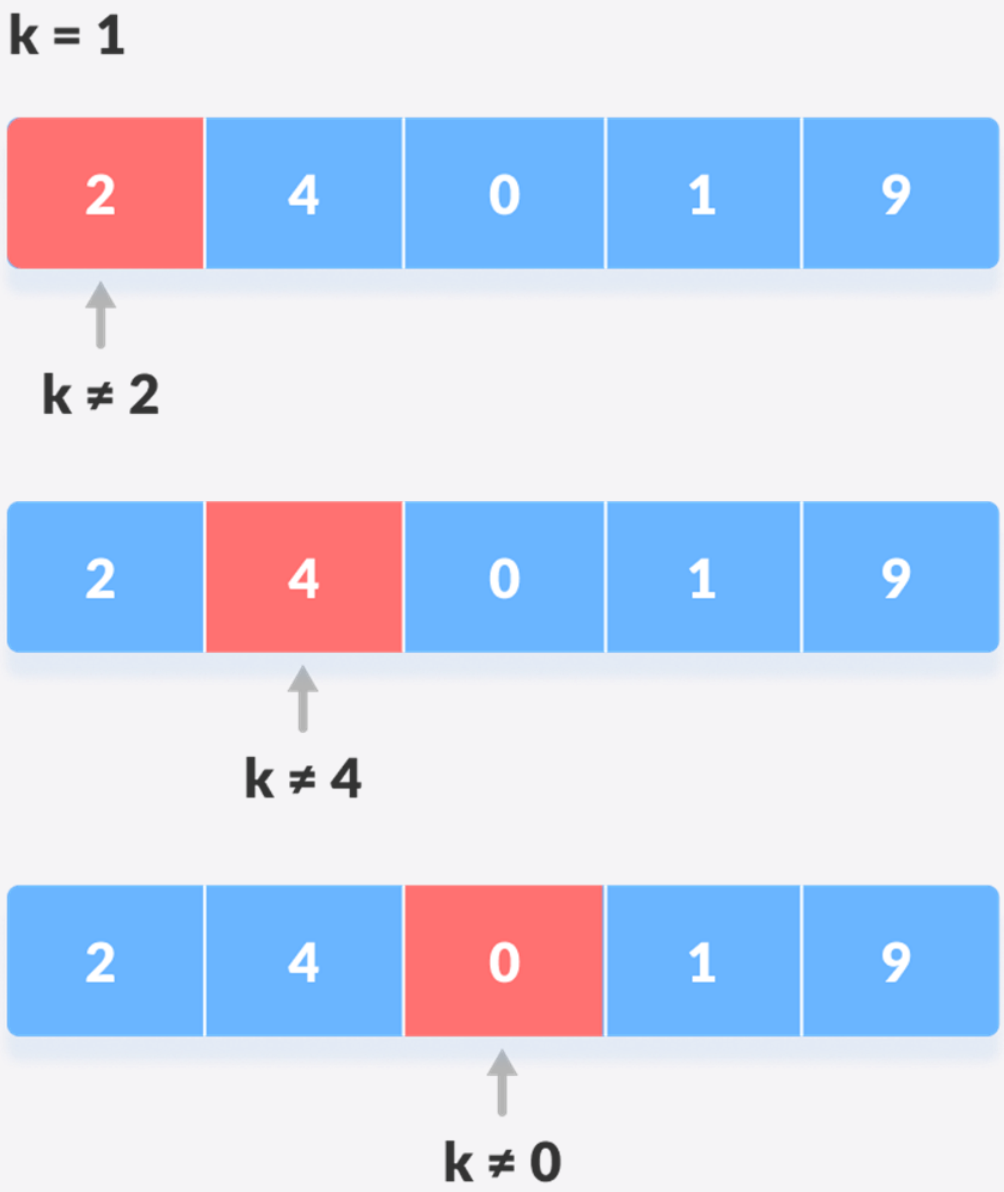


How Linear Search Works?

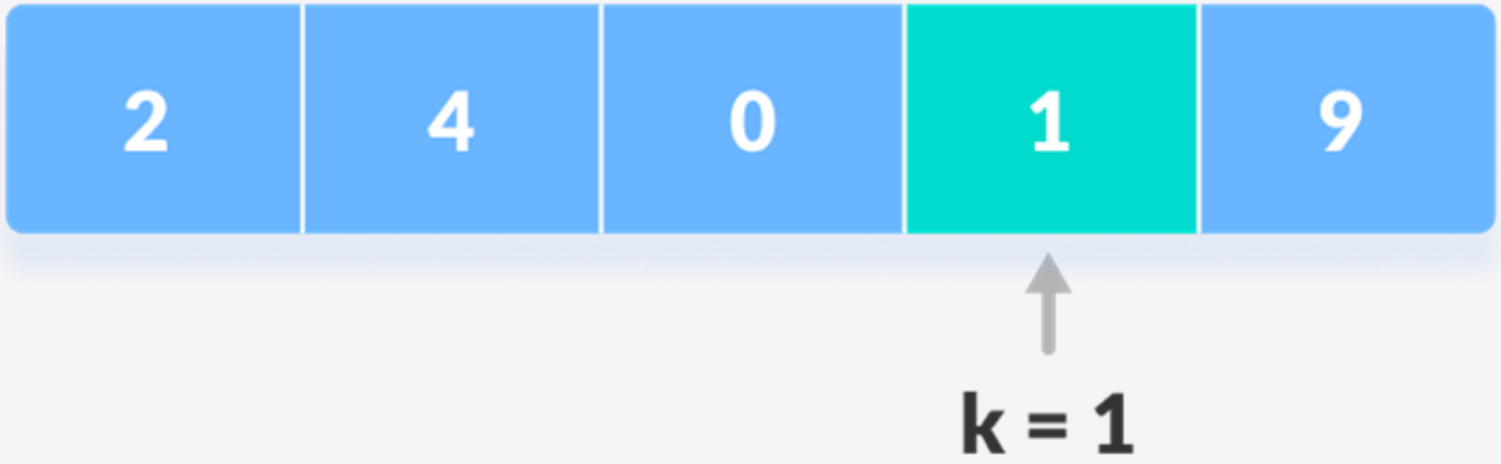
The following steps are followed to search for an element **k = 1** in the list below.

Supposing that our array is an array of integers and unsorted.

1. Start from the first element, compare **k = 1** with each element **x**.



2. If **x == k**, return the **index**.



3. Else, return **not found**.

Linear Search Algorithm

```
LinearSearch(array, key)
  for each item in the array
    if item == value
      return its index
```

Linear Search Complexities

Time complexity

Space complexity: O(1)

Worst case: O(n)

(the element does not exist or is in the last position)

Best case: Ω(1)

(the item is in first position)

C++ Example

Program body:

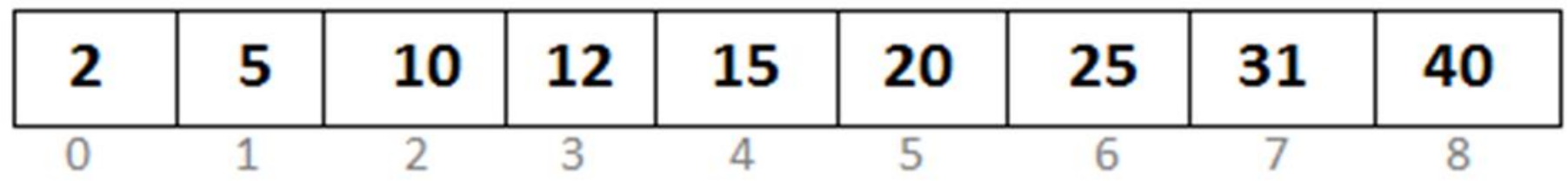
```cpp
#include <iostream>
using namespace std;

int search(int array[], int n, int x)
{
    // Going through array sequencially
    for (int i = 0; i < n; i++)
        if (array[i] == x)
            return i;

    return -1;
}

int main(void)
{
    system("cls");

    int array[] = {5, 3, 7, 8, 4, 9, 6};
    int n = sizeof(array) / sizeof(array[0]);

    int x = 8;

    int result = search(array, n, x);

    if (result == -1)
        printf("Not found");
    else
        printf("Element is found at index %d", result);

    return 0;
}
```

Result:

```
Element is found at index 3
```

Linear Search in sorted Arrays

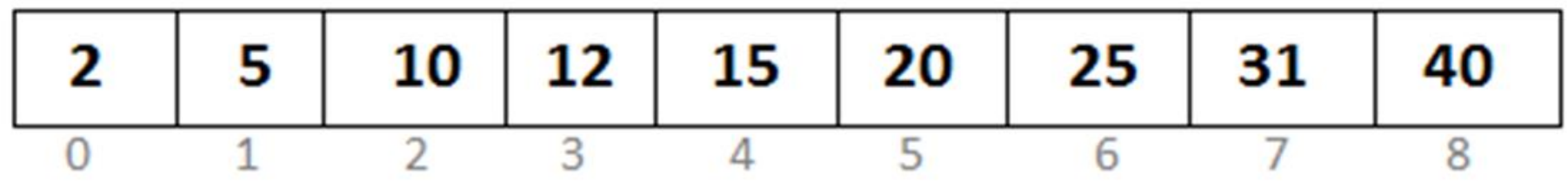
Case 1. Search for the element **20**.



On which index does it stop?

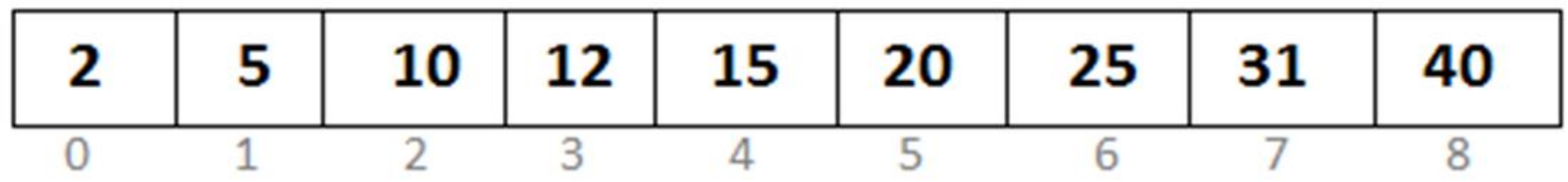
Linear Search in sorted Arrays

Case 1. Search for the element **20**.



On which index does it stop?

Case 2. Search for the element **26**.



On which index does it stop?

Linear Search Complexities in sorted arrays

Time complexity

Worst case: n

(If the item is in last position)

Average case: n/2

(On average if the item does not exist)

Best case: 1

(the item is in first position)

Space complexity: O(1)

C++ Example

sorted arrays

Program body:

```cpp
#include <iostream>
using namespace std;

int sort_search(int array[], int n, int x)
{
    // Going through array sequencially
    for (int i = 0; i < n; i++)
    {
        if (array[i] > x)
        {
            cout << "Stop: " << i << endl;
            break;
        }
        if (array[i] == x)
            return i;
    }
    return -1;
}

int main(void)
{
    system("cls");

    int array[] = {3, 4, 5, 7, 8, 9};
    int n = sizeof(array) / sizeof(array[0]);

    int x = 6;

    int result = sort_search(array, n, x);

    if (result == -1)
        printf("Not found");
    else
        printf("Element is found at index %d", result);

    return 0;
}
```

Result:

```
Stop: 3
Not found
```

Binary Search

Binary Search is a searching algorithm for finding an element's position in a **sorted array**.

Binary search compares the target value to the **middle element** of the array. If they are not equal, the **half in which the target cannot lie is eliminated** and the search **continues on the remaining half**, and repeating this until the target value is found.

Note. Binary search can be implemented **only on a sorted list** of items. If the elements are not sorted already, we need to sort them first.

Binary Search Working

Binary Search Algorithm can be implemented in two ways which are discussed below.

1. **Iterative Method**
2. **Recursive Method**

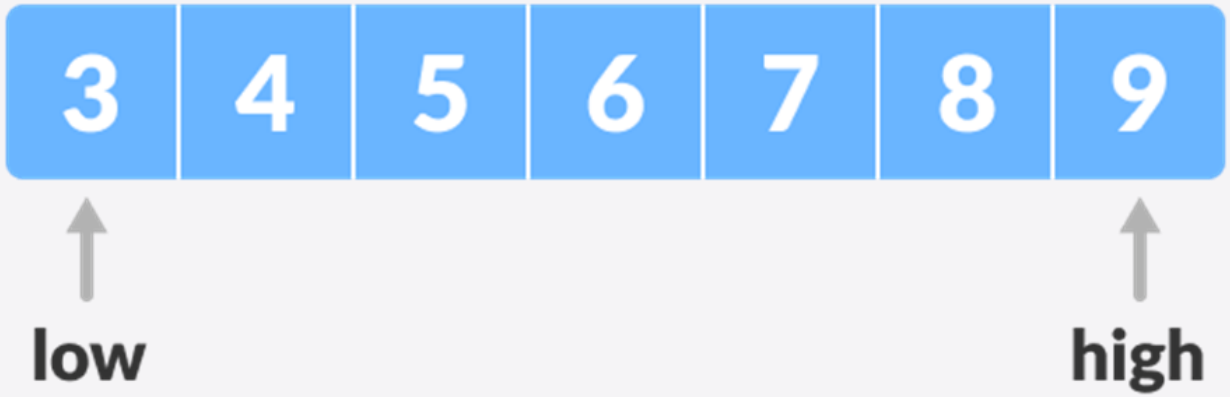
The recursive method follows the divide and conquer approach.

How Linear Search Works?

1.The array in which searching is to be performed is:

Let **x = 4** be the element to be searched.

2. Set **two pointers** **low** and **high** at the lowest and the highest positions respectively.

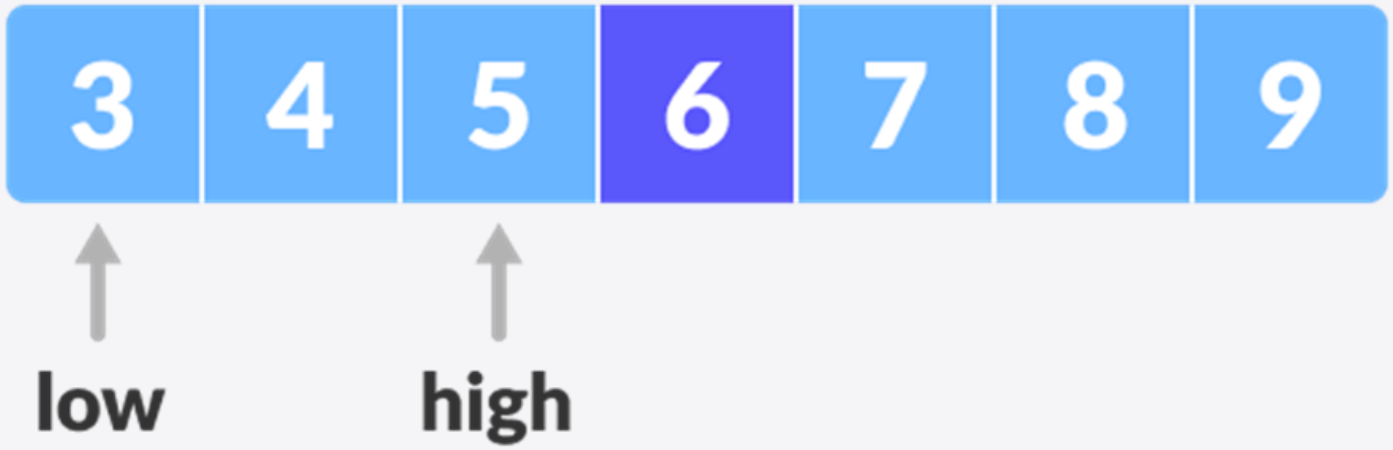


3. Find the middle element **mid** of the array, i.e., **arr[(low + high)/2] = 6**.

4. If **x == mid**, then **return mid**. Else, compare the **element to be searched x** with **mid**.

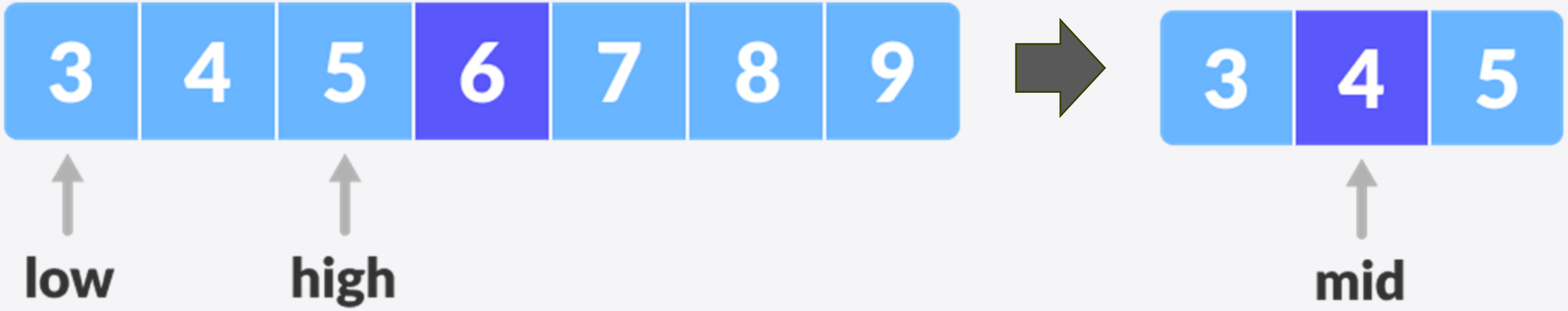
5. If **x > mid**, compare **x** with the **middle element** of the elements on the **mid right array section**. This is done by setting **low** to **low = mid + 1**.

6. Else, compare **x** with the **middle element** of the elements on the **mid left array section**. This is done by setting **high** to **high = mid - 1**.

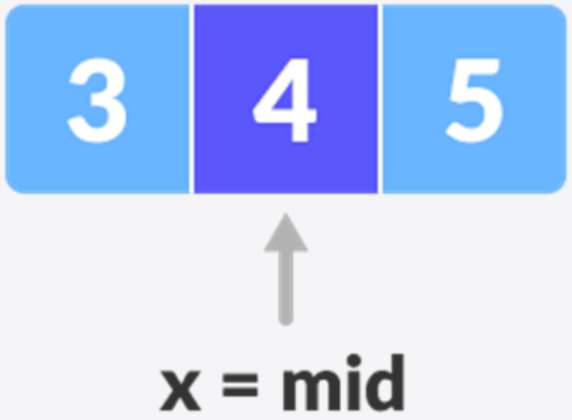


Finding element If **x < mid**

7. Repeat **steps** **3** to **6** until **low** meets **high**.



8. **x = 4** is found.



Binary Search Algorithm

Iteration Method

```
do until the pointers low and high meet each other.
    mid = (low + high)/2
    if (x == arr[mid])
        return mid
    else if (x > arr[mid]) // x is on the right side
        low = mid + 1
    else                          // x is on the left side
        high = mid - 1
```

¿Recursive Method?

Linear Search Complexities

Time complexity

Best case:

Worst / Average case:

Space complexity:

Linear Search Complexities

Time complexity

Best case: Ω(1)

(the item is in middle position)

Worst / Average case:

Space complexity:

Linear Search Complexities

Time complexity

Best case: Ω(1)

(the item is in middle position)

Worst / Average case: O(log n) ¿?

(the element does not exist or is in the first / last position)

Space complexity:

Linear Search Complexities

Time complexity

Best case: Ω(1)

(the item is in middle position)

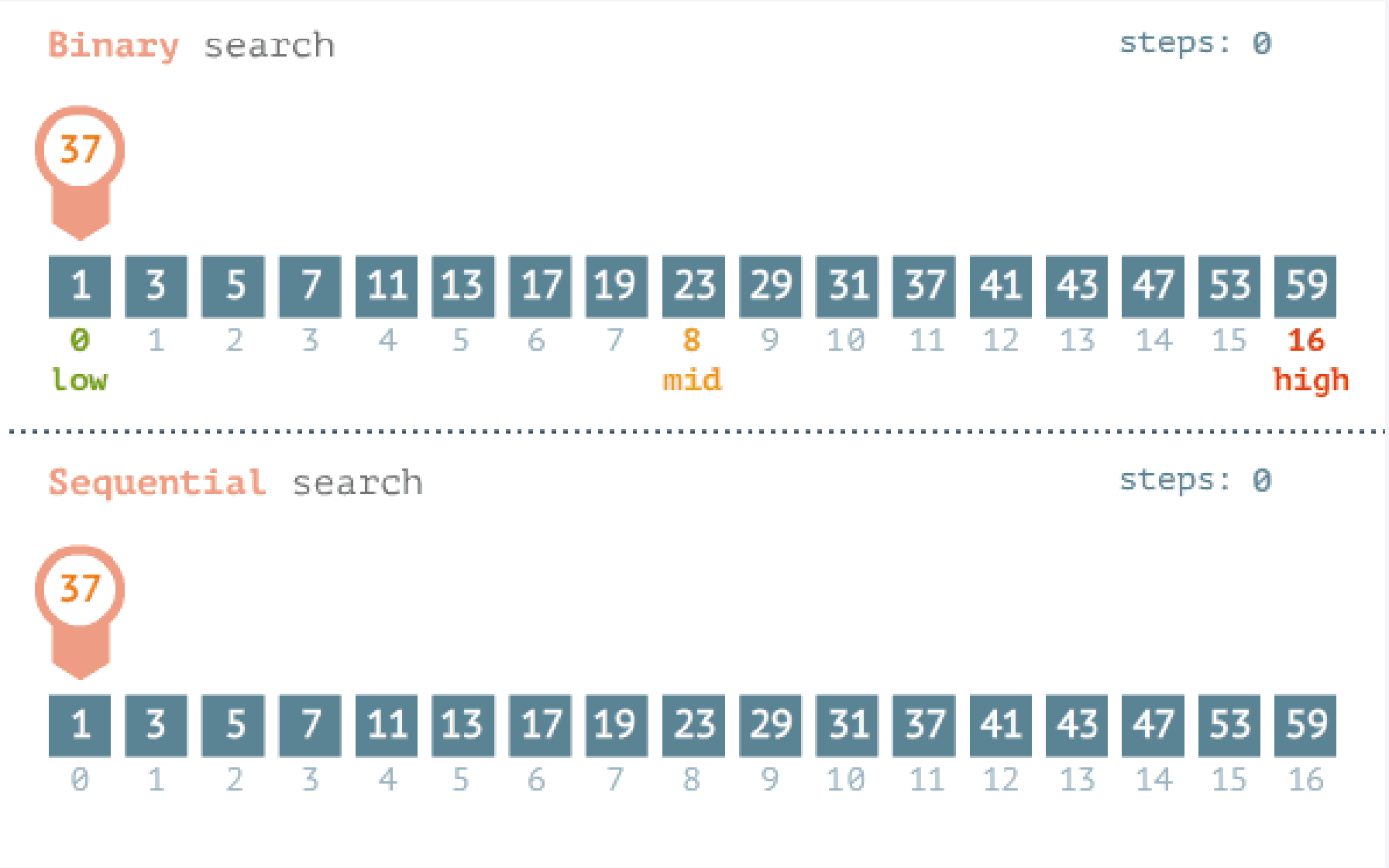
Worst / Average case: O(log n)

(the element does not exist or is in the first / last position)

Space complexity:

The space complexity of the binary search is **O(1)**.

Summary



Binary search is an algorithm for efficiently searching for an item within a **sorted list of items**.