# Analysis and Design of Algorithms

# Searching Algorithms

# Table of Contents

# Searching Algorithms

- **Searching Algorithm** is an algorithm made up of a series of instructions that retrieves information stored within some data structure, or calculated in the search space of a problem domain.

- There are many searching algorithms, such as:
  - Linear Search, Binary Search, Jump Search, Interpolation Search, Exponential Search, Ternary Search

# Linear Search

# Linear Search

- **Linear Search** is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.

# Linear Search

## ❑ Algorithm:

- **Step1**: Start from the leftmost element of array and one by one compare x with each element of array.
- Step2: If x matches with an element, return the index.
- Step3: If x doesn't match with any of elements, return -1.

# Linear Search

- Assume the following Array:
- Search for 9

| 8 | 12 | 5 | 9 | 2 |
|---|---|---|---|---|

# Linear Search

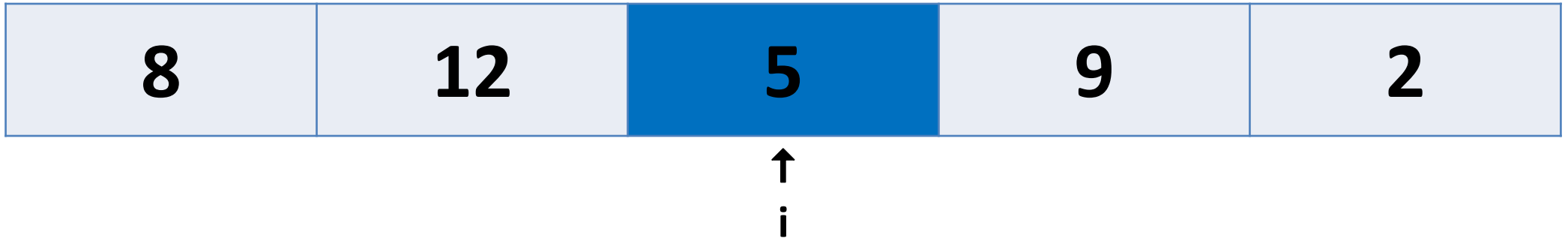
- Compare
- X= 9

# Linear Search

- Compare
- X= 9

# Linear Search

- Compare
- X= 9



# Linear Search

- Compare
- X= 9

# Linear Search

- Found at index = 3

# Linear Search

- **Time Complexity:** O(n)

- Example of worst case: search for the last element

| 4 | 6 | 8 | 9 | 1 |
|---|---|---|---|---|

# Binary Search

# Binary Search

- **Binary Search** is the most popular Search algorithm. It is efficient and also one of the most commonly used techniques that is used to solve problems. Binary search use sorted array by repeatedly dividing the search interval in half.

# Binary Search

- Algorithm:

- Step1: Compare x with the middle element.
- Step2: If x matches with middle element, we return the mid index.
- Step3: Else If x is greater than the mid element, search on right half.
- Step4: Else If x is smaller than the mid element. search on left half.

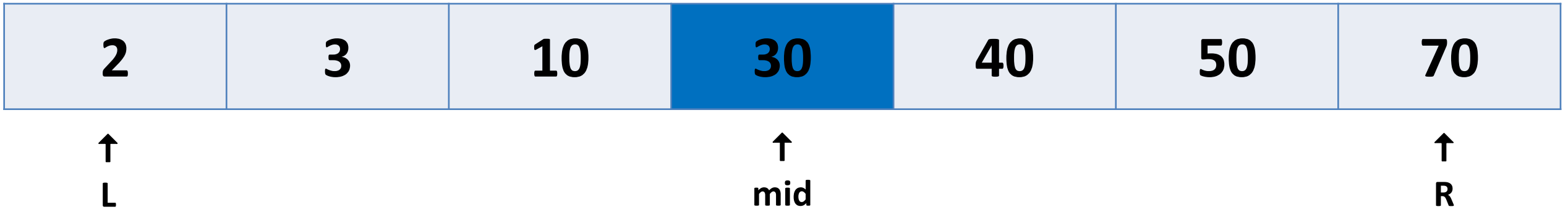
# Binary Search

- Assume the following Array:
- Search for 40

| 2 | 3 | 10 | 30 | 40 | 50 | 70 |
|---|---|---|---|---|---|---|

# Binary Search

- Compare
- X = **40**

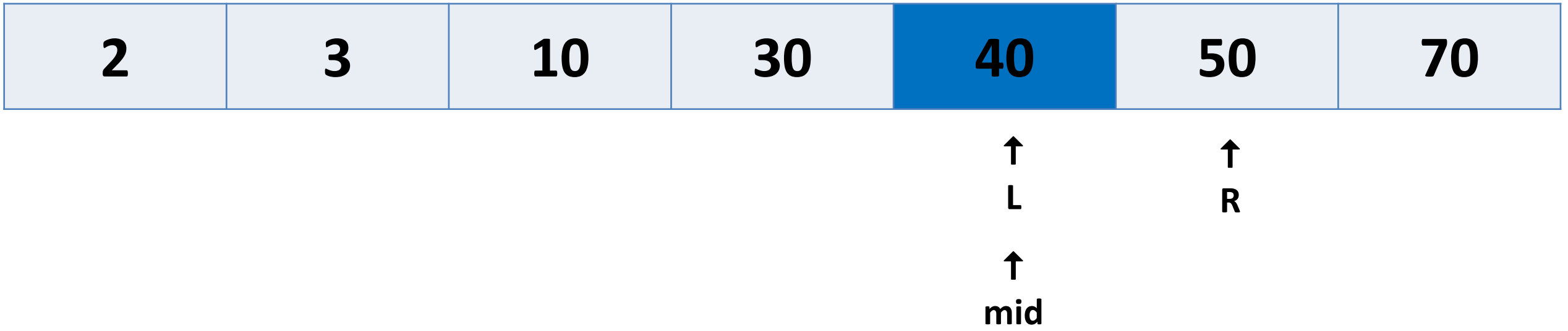


# Binary Search

- Compare
- X = **40**

# Binary Search

- Compare
- X = **40**



# Binary Search

- x=40 , found at index = 4

| 2 | 3 | 10 | 30 | 40 | 50 | 70 |
|---|---|---|---|---|---|---|

# Binary Search

- **Time Complexity:** $O(\log_2 n)$

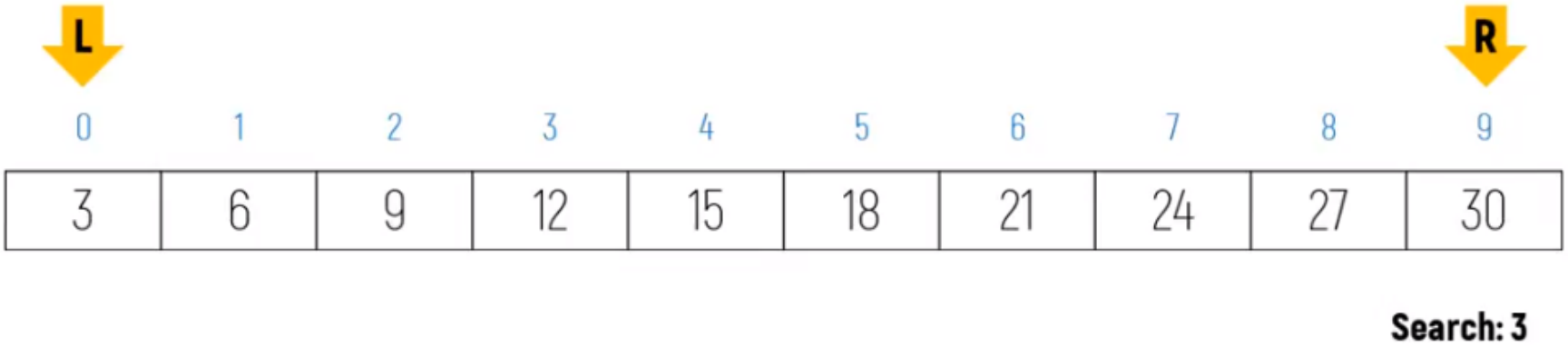
# Another Example

A sorted array .

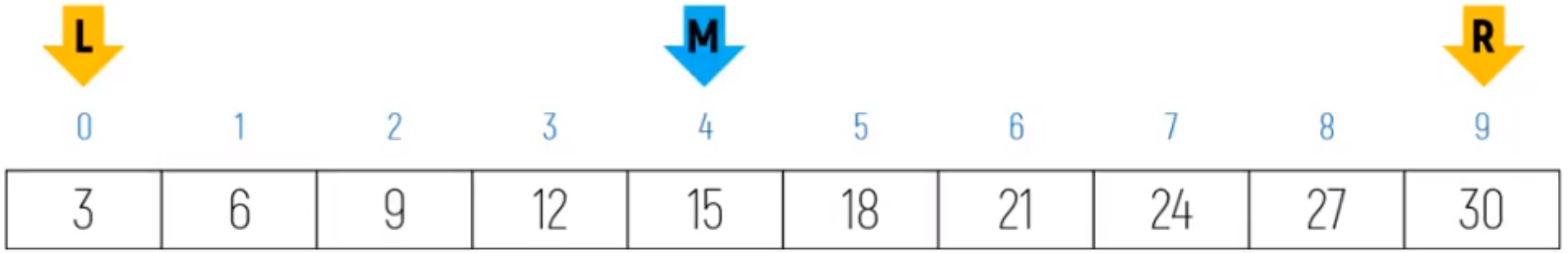
First element index denoted as L.

Last element index denoted as R.

We have to search : 3



For this procedure we need to know the mid point of this array.




**Search: 3**

**Mid-Pont** = (L+R) ÷ 2

= (0+9) ÷ 2

= 9 ÷ 2

= 4.5

= 4

Mid point value is not the searching value. Mid point value = Array[4] = 15

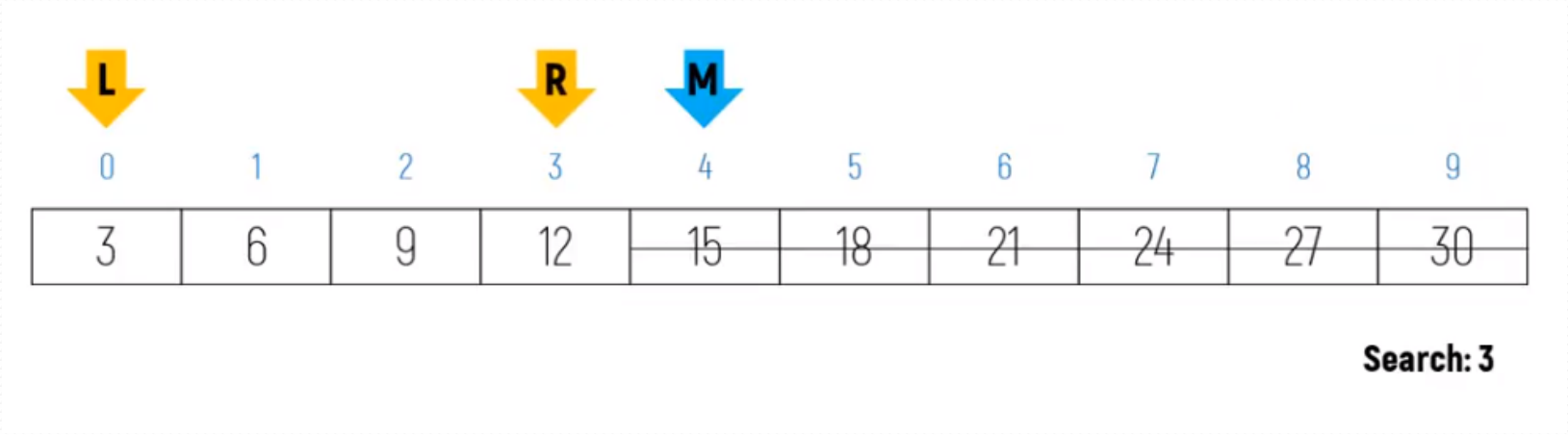
Searching value < Mid point value

So, R = mid point – 1

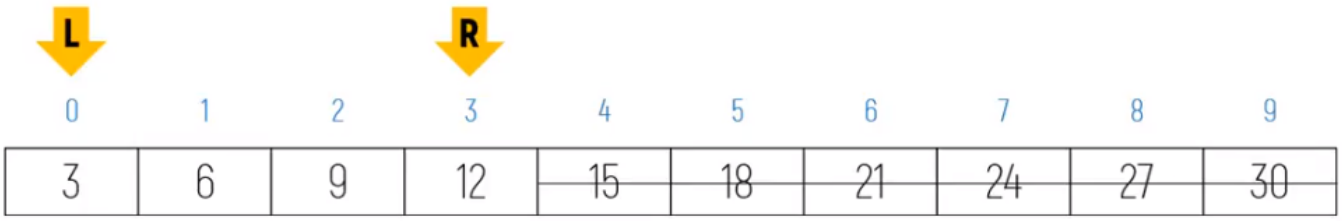
= 4-1

= 3

Ignore all the values of the right side of new R



New array index from 0 to 3.




**Search: 3**

**Mid-Pont** = (L+R) ÷ 2
= (0+3) ÷ 2
= 3 ÷ 2
= 1.5
= 1

Mid point value is not the Searching value .

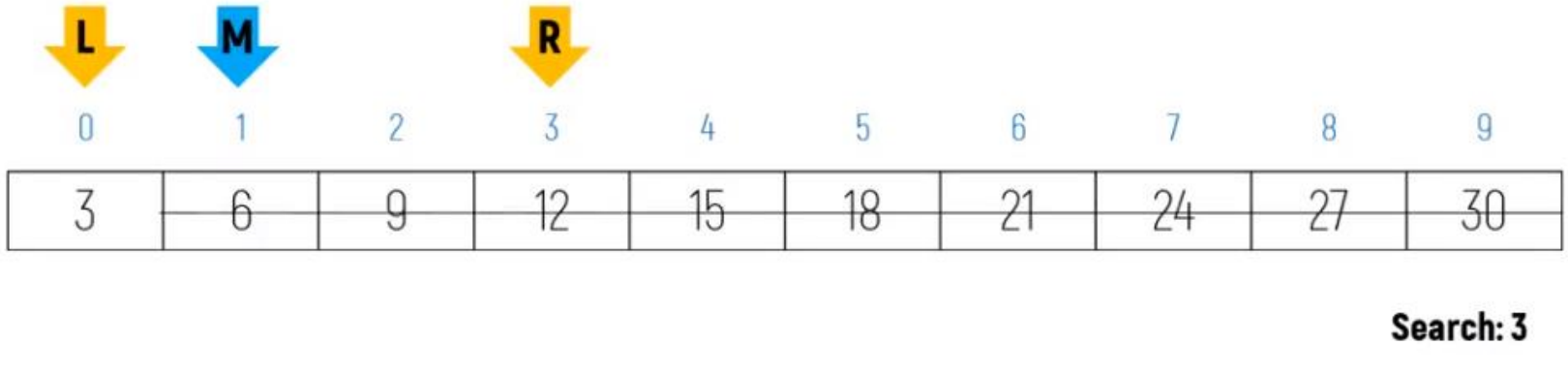
Searching value < Mid point value

So, R = Mid point – 1

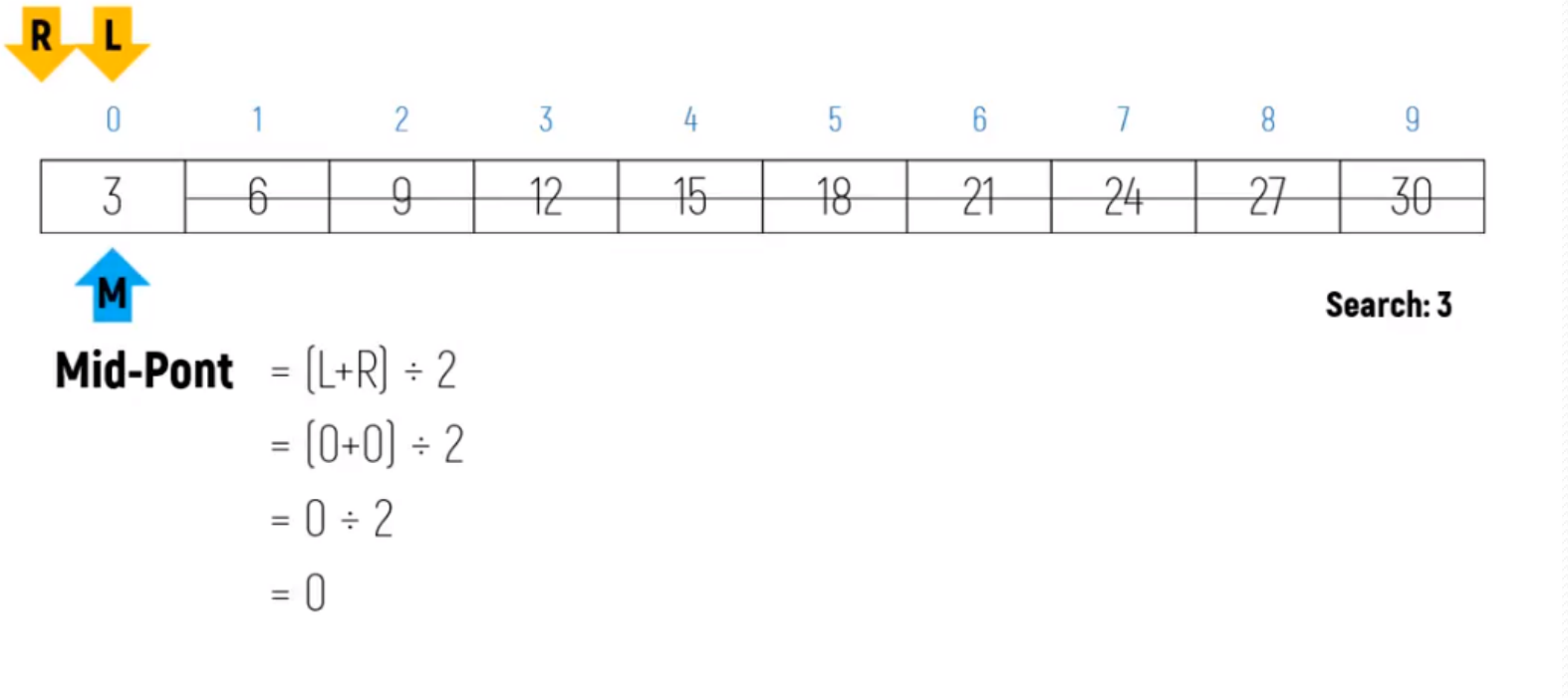
= 1-1

= 0

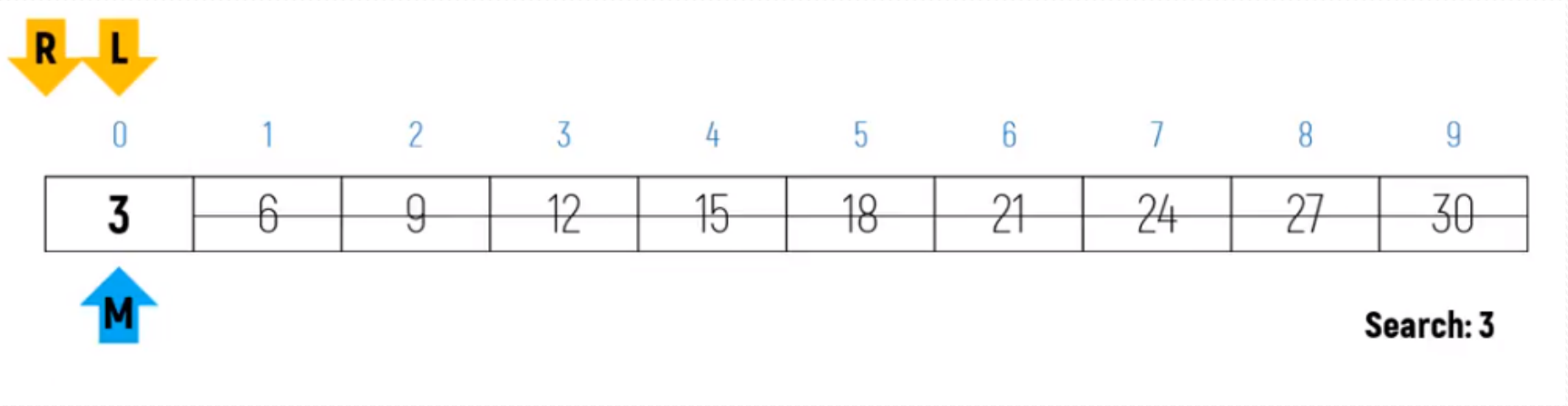
Ignore all the values of the right side of new R.



New array index is 0.



Mid point value is the searching value . So we have found the value in 3 times of searching.



# Time complexity of Binary search:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |

If there are n elements, then the maximum searching time is: $\mathbf{Log_2\ n}$

We always must assume that we have $\mathbf{2^n}$

There are 10 elements but we assume that, we have 16($\mathbf{2^4}$) elements.

So, the maximum required search is: $\mathbf{Log_2\ 16} = \mathbf{Log_2\ 2^4} = 4$

Time Complexity of the Binary Search is: $\mathbf{O(log_2\ n)}$

# Code

```
int BinarySearch(int arr[], int l, int r, int n) {
    while(l<=r) {
        int mid = (l+r)/2;
        if(n==arr[mid]) {
            return mid;
        } else if(n<arr[mid]) {
            r = mid-1;
        } else if(n>arr[mid]) {
            l = mid+1;
        }
    }
    return -1;
}
```

# Advantage and Disadvantage

**Advantage** :

Binary Search is an optimal searching algorithm using which we can search desired element very efficiently.

**Disadvantages** :

This algorithm requires the list to be sorted. Then only this method is applicable.

# THANKS FOR YOUR TIME