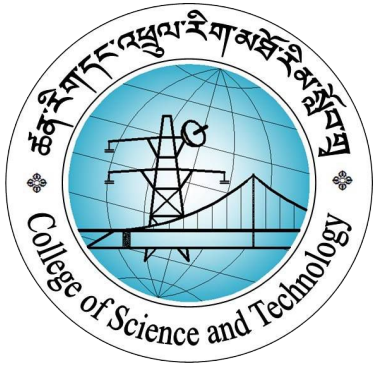



Royal University of Bhutan

# Unit IV: Introduction to Computational Problems & Algorithms

Programming Methodology (CSF101)

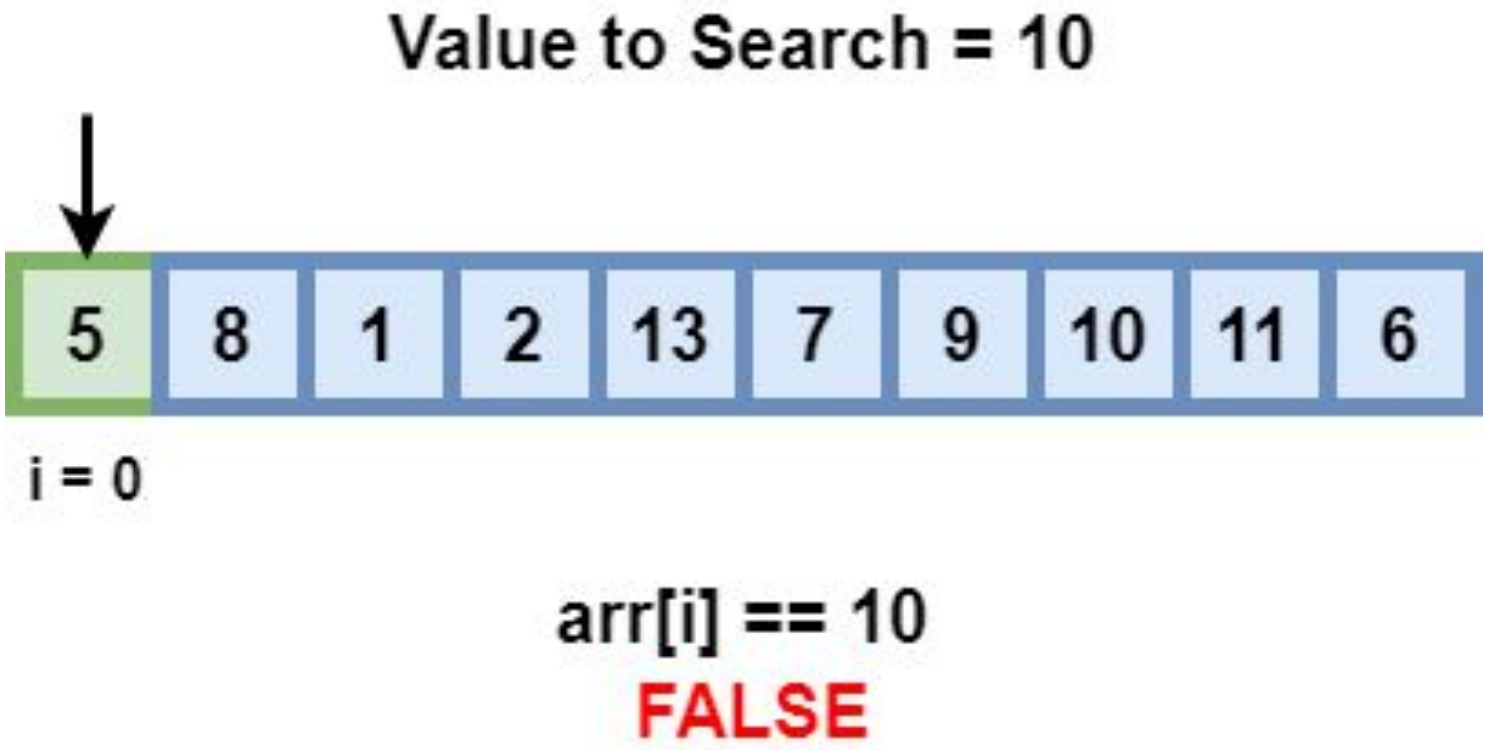
# Outline

- Searching Algorithms
- Arrays & Hashing; Contains Duplicate, Valid Anagram, Two Sums
- Two Pointers; Valid Palindrome, Three Sums

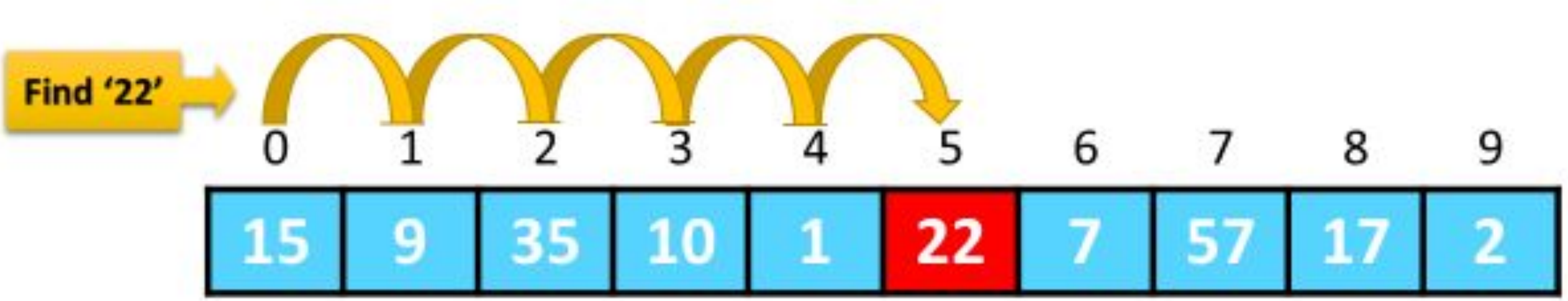
# Searching Algorithms

# Two types of searching algorithms

# Linear Search



# Example



# Linear Search implementation in python code

```
def linearSearch(array, n, x):

    for i in range(0, n):
        if (array[i] == x):
            return i
        print(f"i: {i}, array[i]: {array[i]}")
    return -1

print(linearSearch([15, 9, 35, 10, 1, 22, 7, 57, 17, 2], 10, 22))
```

```
i: 0, array[i]: 15
i: 1, array[i]: 9
i: 2, array[i]: 35
i: 3, array[i]: 10
i: 4, array[i]: 1
5
```

# Complexity Analysis of Linear Search

**Time Complexity:**

Best Case: O(1)

Worst Case: O(n)

Average Case: O(n)

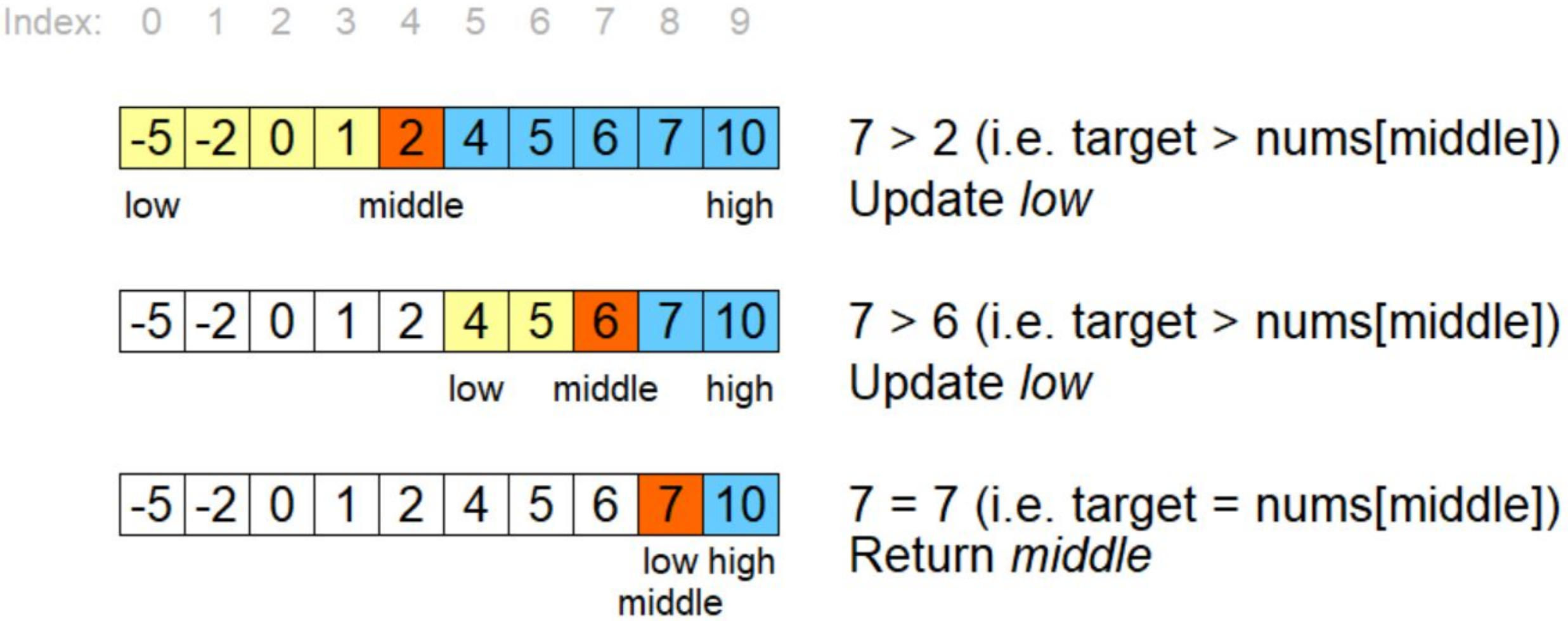
**Space Complexity:** O(1)

# Binary Search

Search for 47

| 0 | 4 | 7 | 10 | 14 | 23 | 45 | 47 | 53 |
|---|---|---|---|---|---|---|---|---|

# Example


Index: 0 1 2 3 4 5 6 7 8 9
-5 -2 0 1 2 4 5 6 7 10
low middle high
7 > 2 (i.e. target > nums[middle])
Update low
-5 -2 0 1 2 4 5 6 7 10
low middle high
7 > 6 (i.e. target > nums[middle])
Update low
-5 -2 0 1 2 4 5 6 7 10
low high
middle
7 = 7 (i.e. target = nums[middle])
Return middle

# Implementation in python code

```
def binarySearch(array, x, low, high):
    # Repeat until the pointers low and high meet each other
    while low <= high:
        mid = low + (high - low)//2  # Calculate the middle index

        if array[mid] == x:  # If the middle element is the target value, return its
        index
            print(f"The value {x} is at index:")
            return mid

        elif array[mid] < x:  # If the middle element is less than the target value,
        search the right half
            low = mid + 1
        else:  # If the middle element is greater than the target value, search the
        left half
            high = mid - 1

    return -1  # Return -1 if the target value is not found in the array

# Example usage:
print(binarySearch([-5, -2, 0, 1, 2, 4, 5, 6, 7, 10], 7, 0, 9))
```

```
The value 7 is at index:
8
```

# Complexity Analysis of Binary Search

**Time Complexity:**

Best Case : O(1) #first mid value

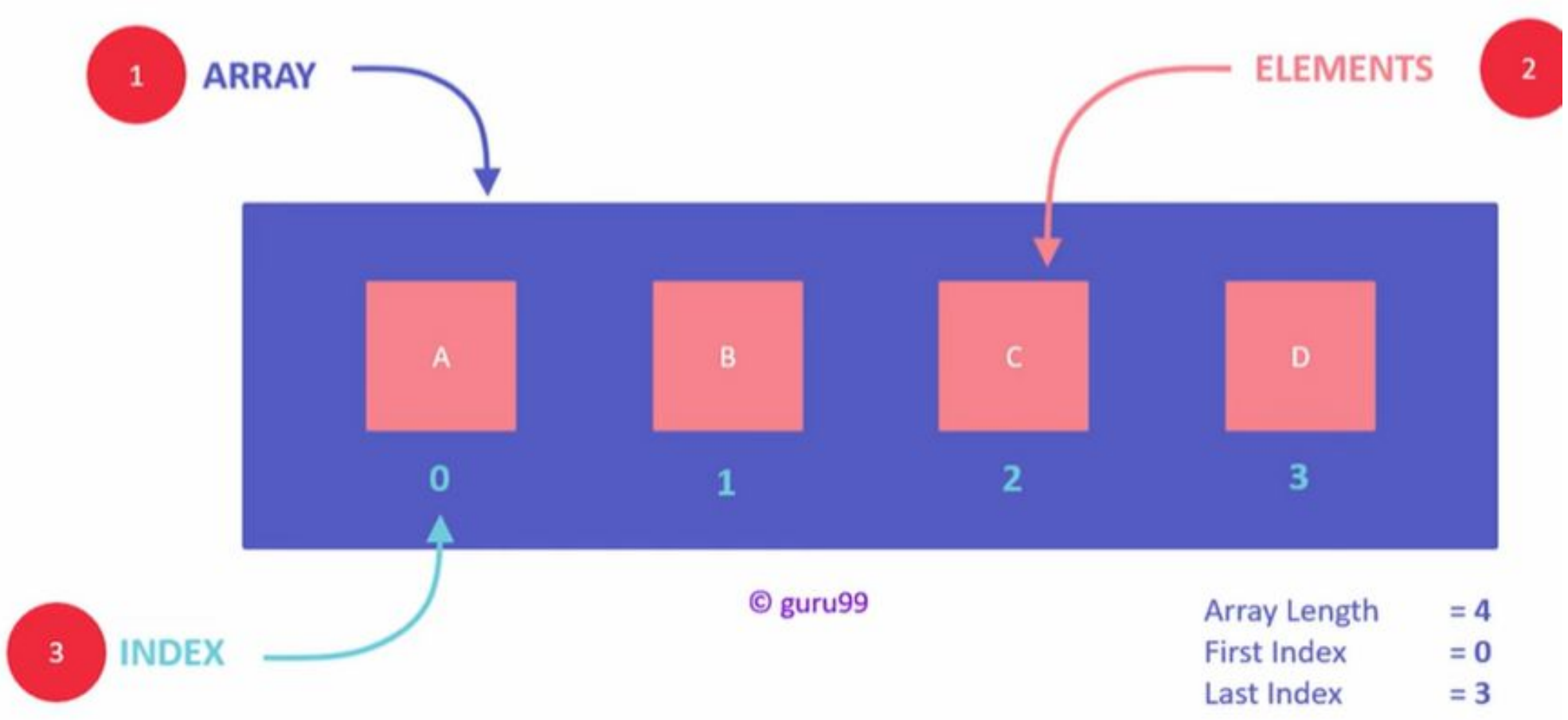
Worst Case : O(logn) #extreme values or no values at all

Average Case : O(logn) #uniform distribution of values

**Space Complexity:**

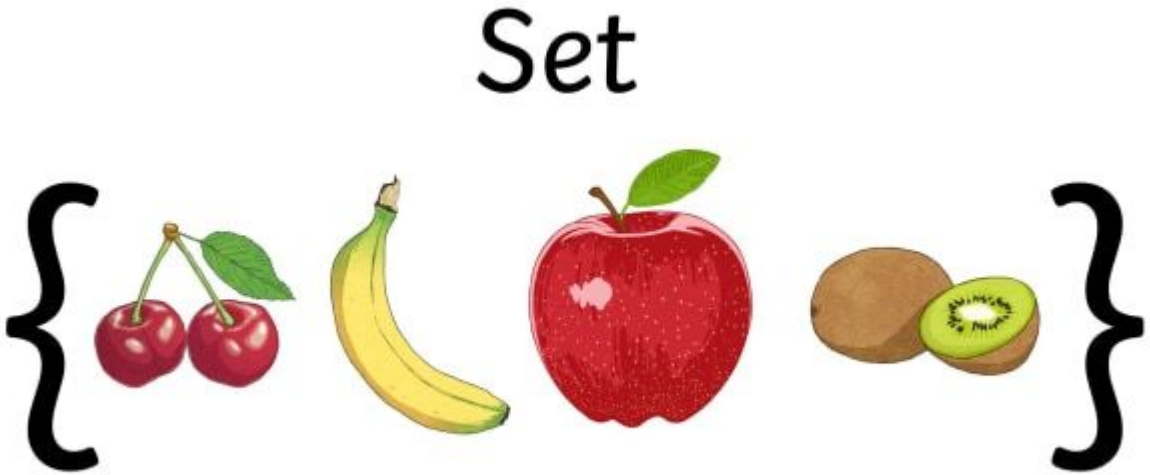
O(1)

# Brief Recapitulation



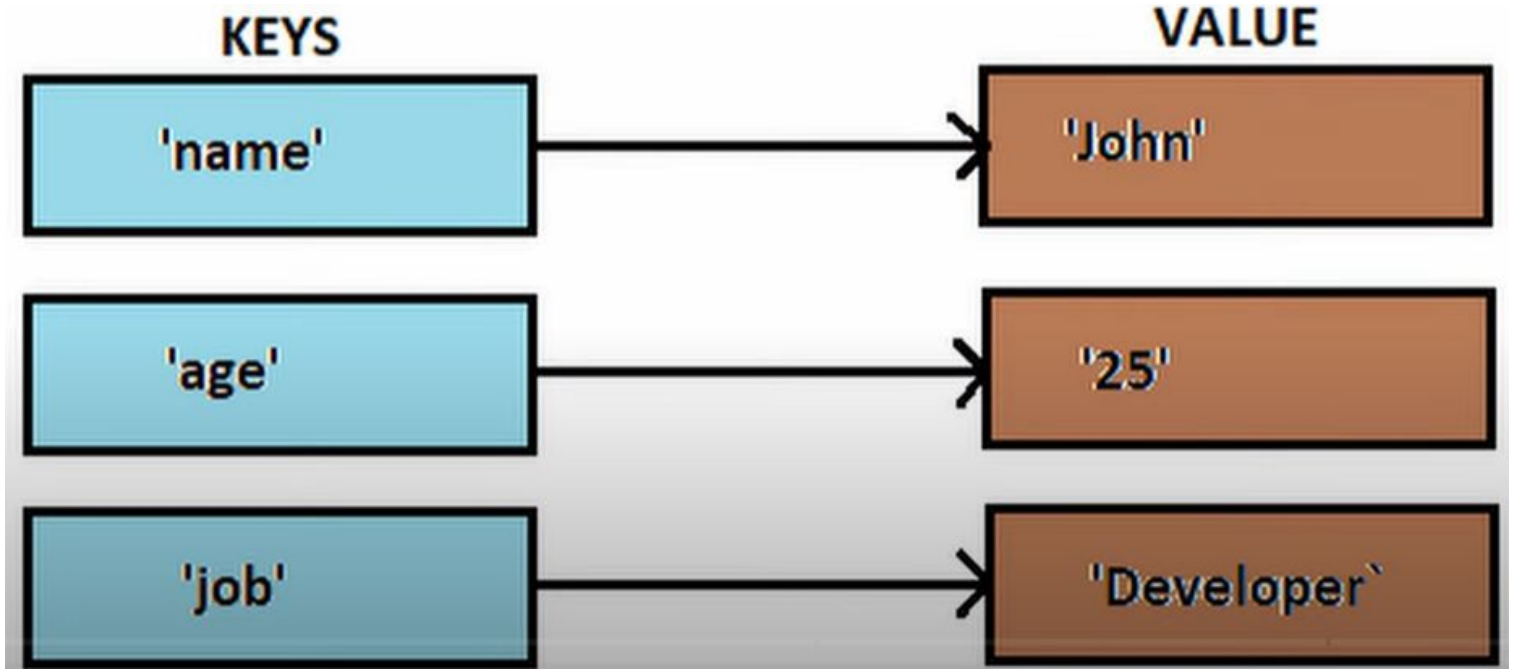
## Conti…

Set is AKA hashsets



# Cont…

Dictionary is AKA hashmaps.



# Cont…

# Leetcode Problems on Arrays and Hashings

1. Contains-duplicate
2. Two-sum
3. valid-anagram

# LeetCode problems on Two Pointers

1. Valid-palindrome
2. Three-sums

# Reference

GfG (2024) Searching algorithms, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/searching-algorithms/?ref=lbp

Jana, S. (2022) Time and space complexity of Binary Search, Scaler Topics. Available at: https://www.scaler.com/topics/time-complexity-of-binary-search/

GeeksforGeeks (2024) Time and space complexity of linear search algorithm, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/time-and-space-complexity-of-linear-search-algorithm/


Thanks!!