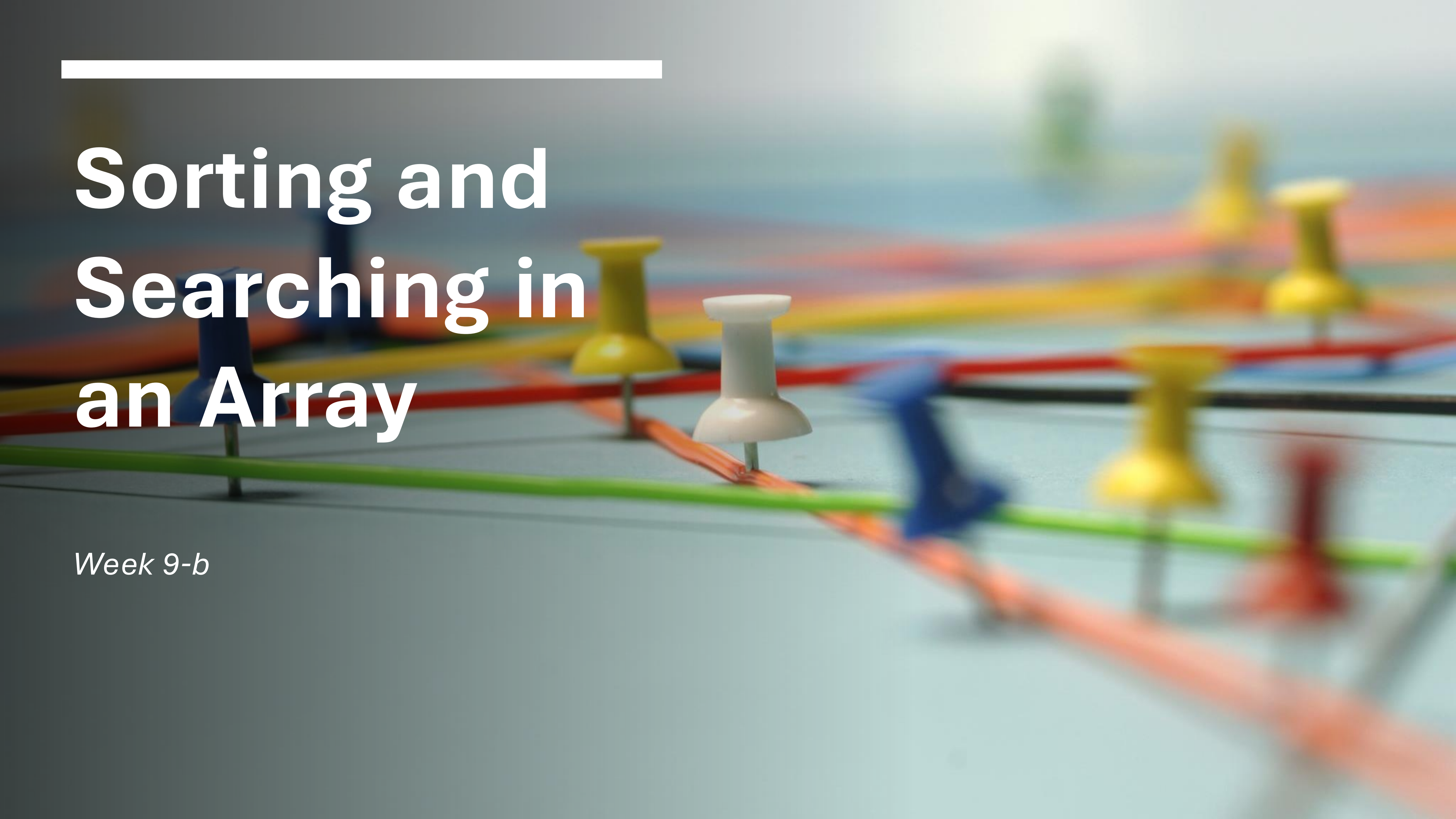
Fundamentals of Computer Programming

CS-110

Course Instructor: Dr. Momina Moetesum



Sorting and Searching in an Array

Week 9-b

Learning Objectives

01

To understand how to sort an array using Bubble Sort

02

To understand how to search for an item in an array using Linear Search

03

To understand how to search for an item in an array using Binary Search

Sorting an Array

For Smaller Arrays (<10,000 Elements)

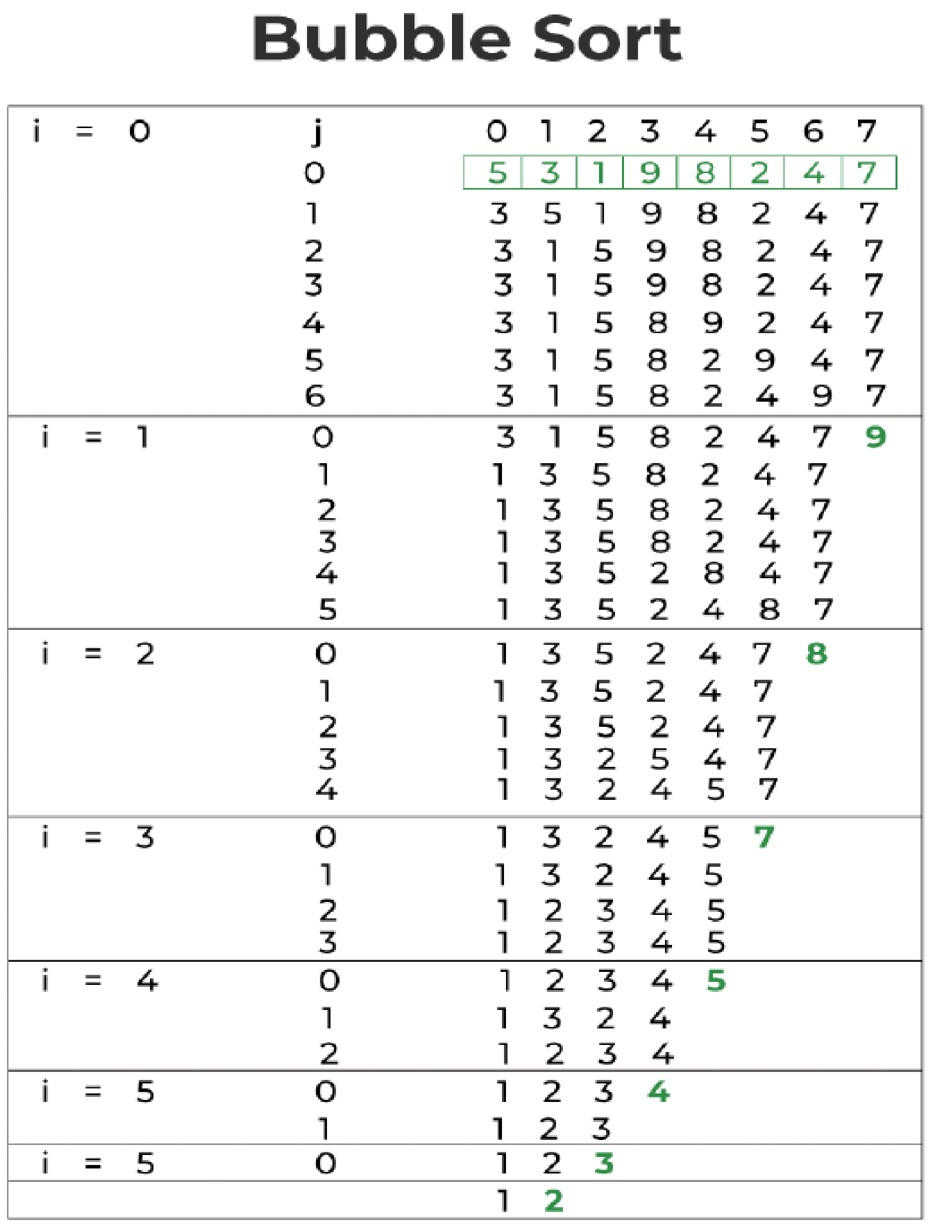
- Bubble Sort
- Selection Sort
- Insertion Sort

For Larger Arrays (>10,000 Elements)

- Quick Sort
- Merge Sort
- Radix Sort
- Heap Sort

Bubble Sort

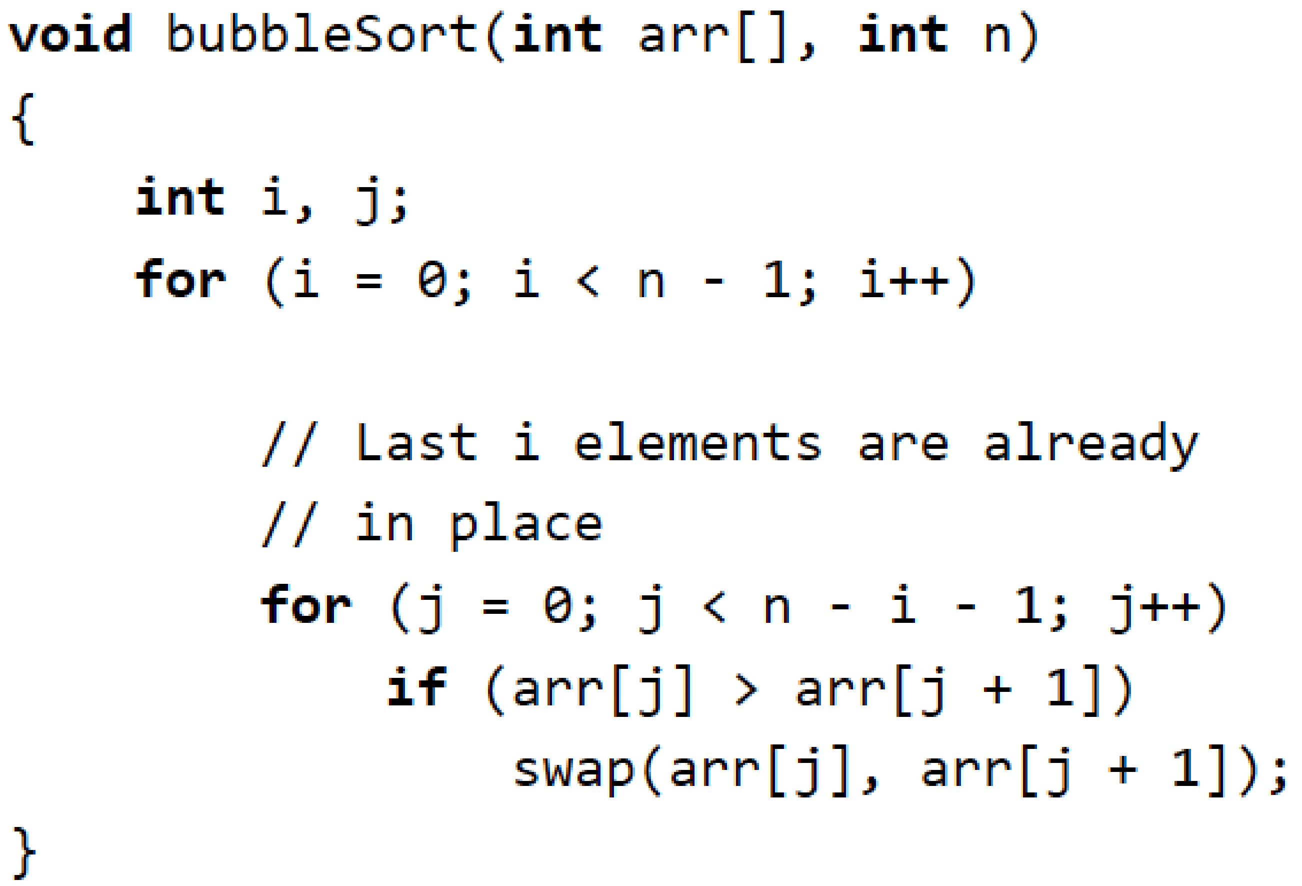
- *Traverse from left and compare adjacent elements and the higher one is placed at right side.*
- *In this way, the largest element is moved to the rightmost end at first.*
- *This process is then continued to find the second largest and place it and so on until the data is sorted.*



Bubble Sort

i = 0	j	0	1	2	3	4	5	6	7
	0	5	3	1	9	8	2	4	7
	1	3	5	1	9	8	2	4	7
	2	3	1	5	9	8	2	4	7
	3	3	1	5	9	8	2	4	7
	4	3	1	5	8	9	2	4	7
	5	3	1	5	8	2	9	4	7
	6	3	1	5	8	2	4	9	7
i = 1	0	3	1	5	8	2	4	7	**9**
	1	1	3	5	8	2	4	7	
	2	1	3	5	8	2	4	7	
	3	1	3	5	8	2	4	7	
	4	1	3	5	2	8	4	7	
	5	1	3	5	2	4	8	7	
i = 2	0	1	3	5	2	4	7	**8**	
	1	1	3	5	2	4	7		
	2	1	3	5	2	4	7		
	3	1	3	2	5	4	7		
	4	1	3	2	4	5	7		
i = 3	0	1	3	2	4	5	**7**		
	1	1	3	2	4	5			
	2	1	2	3	4	5			
	3	1	2	3	4	5			
i = 4	0	1	2	3	4	**5**			
	1	1	3	2	4				
	2	1	2	3	4				
i = 5	0	1	2	3	**4**				
	1	1	2	3					
i = 5	0	1	2	**3**					
		1	**2**						

Naïve Bubble Sort



```
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}
```

Optimized Bubble Sort

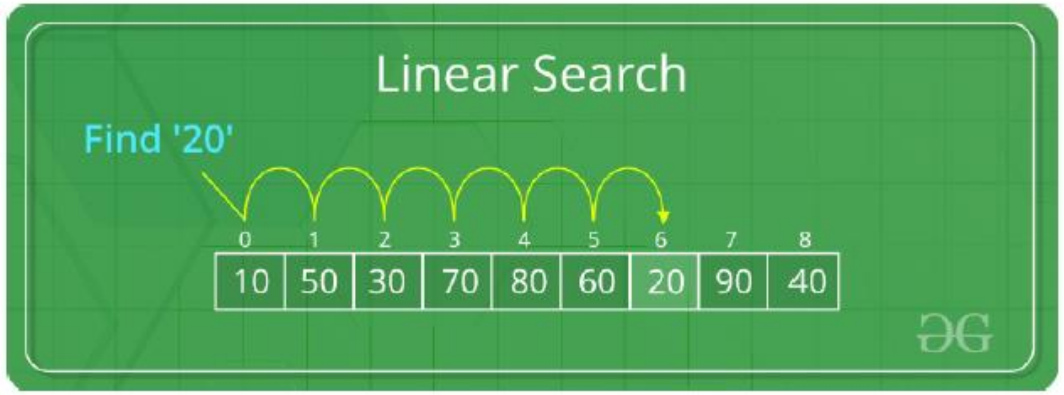
```
void bubbleSort(int arr[], int n) {
    bool isUnsorted;
    do {
        isUnsorted = false;
        for (int i = 0; i < (n - 1); i++) {
            if (arr[i] > arr[i + 1]) {
                isUnsorted = true;
                for (; i < (n - 1); i++) {
                    if (arr[i] > arr[i + 1]) {
                        std::swap(arr[i], arr[i + 1]);
                    }
                }
            }
        }
    } while (isUnsorted);
}
```

Searching in an Array

LINEAR SEARCH

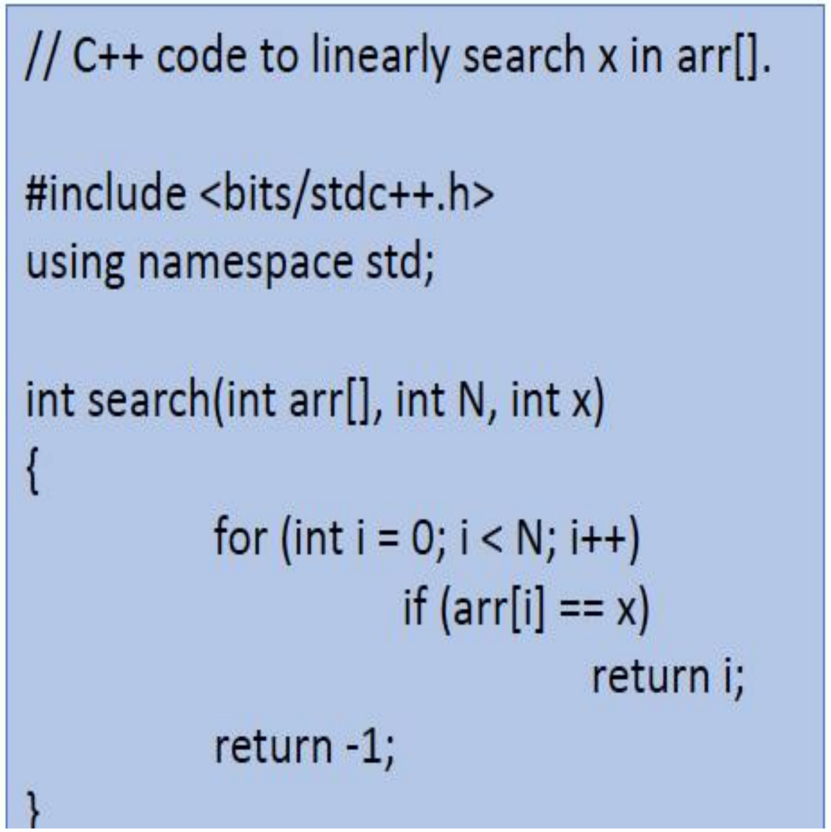
BINARY SEARCH

Linear Search



- Every element is considered as a potential match for the key and checked for the same.
- If any element is found equal to the key, the search is successful and the index of that element is returned.
- If no element is found equal to the key, the search yields “No match found”.

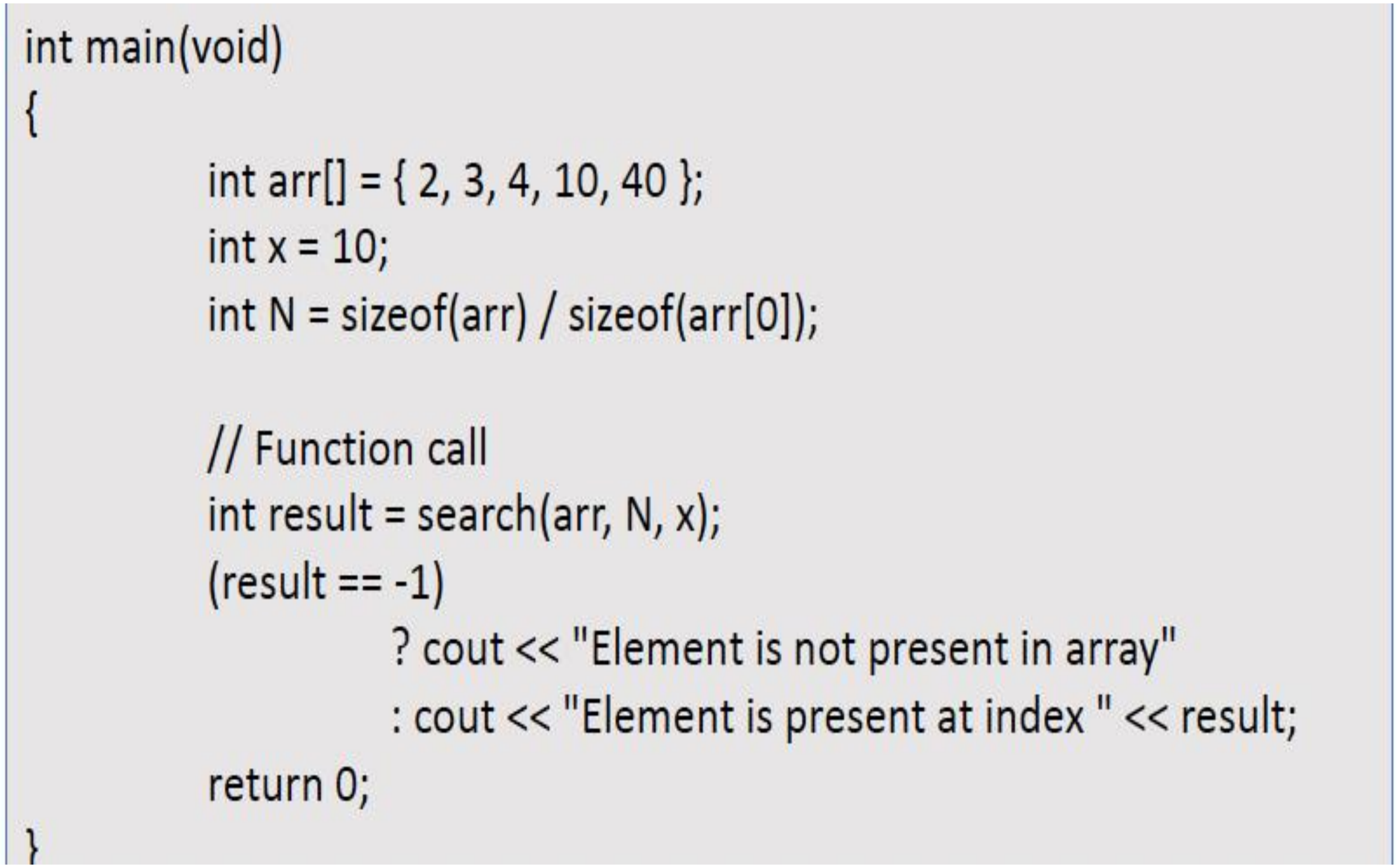
Linear Search



```
// C++ code to linearly search x in arr[].

#include <bits/stdc++.h>
using namespace std;

int search(int arr[], int N, int x)
{
        for (int i = 0; i < N; i++)
                if (arr[i] == x)
                        return i;
        return -1;
}
```

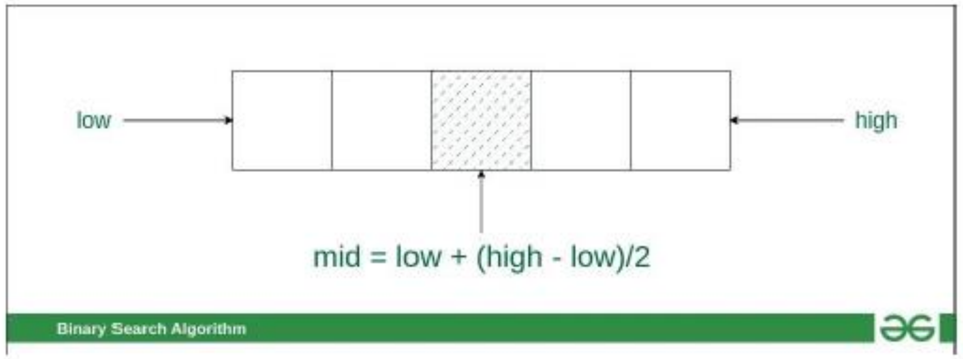


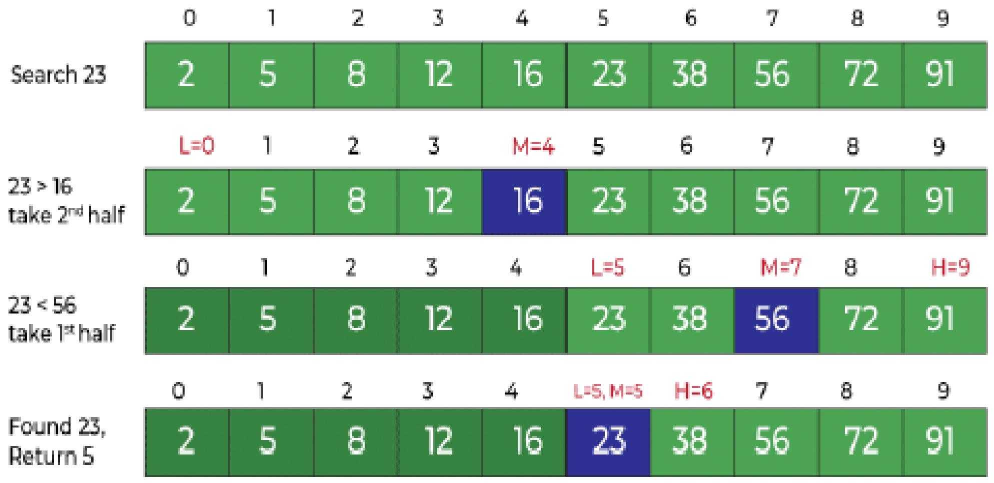
```
int main(void)
{
        int arr[] = { 2, 3, 4, 10, 40 };
        int x = 10;
        int N = sizeof(arr) / sizeof(arr[0]);

        // Function call
        int result = search(arr, N, x);
        (result == -1)
                ? cout << "Element is not present in array"
                : cout << "Element is present at index " << result;
        return 0;
}
```

Binary Search

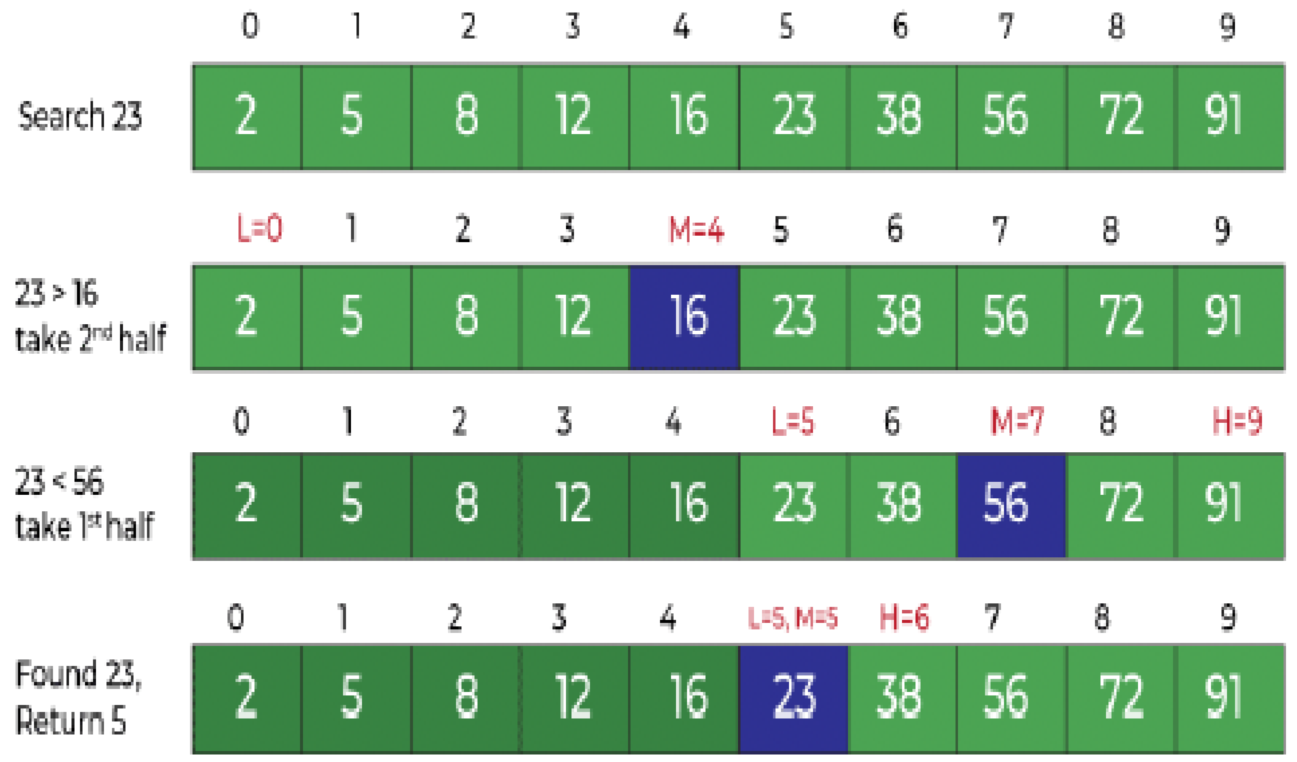
- Compare the middle element of the search space with the key.
- If the key is found at middle element, the process is terminated.
- If the key is not found at middle element, choose which half will be used as the next search space.
    - If the key is smaller than the middle element, then the left side is used for next search.
    - If the key is larger than the middle element, then the right side is used for next search.
- This process is continued until the key is found or the total search space is exhausted.





Binary Search Algorithm

- *Start*
- *Take input array and Target*
- *Initialise start = 0 and end = (array size -1)*
- *Intialize mid variable*
- *mid = (start+end)/2*
- *if array[mid] == target then return mid*
- *if array[mid] < target then start = mid+1*
- *if array[mid] > target then end = mid-1*
- *if start<=end then goto step 5*
- *return -1 as Not element found*
- *Exit*



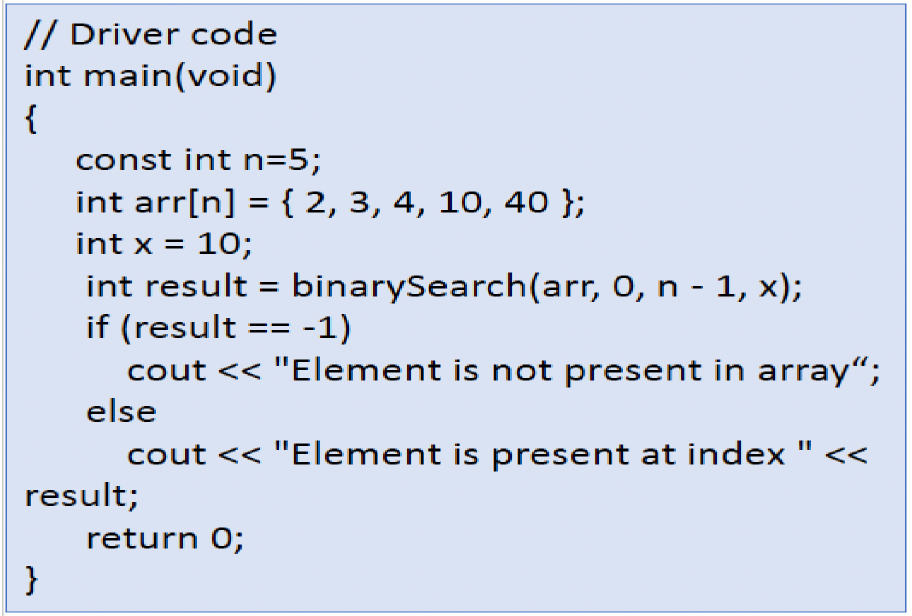
Iterative Solution

```
// Driver code
int main(void)
{
    const int n=5;
    int arr[n] = { 2, 3, 4, 10, 40 };
    int x = 10;
     int result = binarySearch(arr, 0, n - 1, x);
     if (result == -1)
         cout << "Element is not present in array“;
     else
         cout << "Element is present at index " << result;
     return 0;
}
```

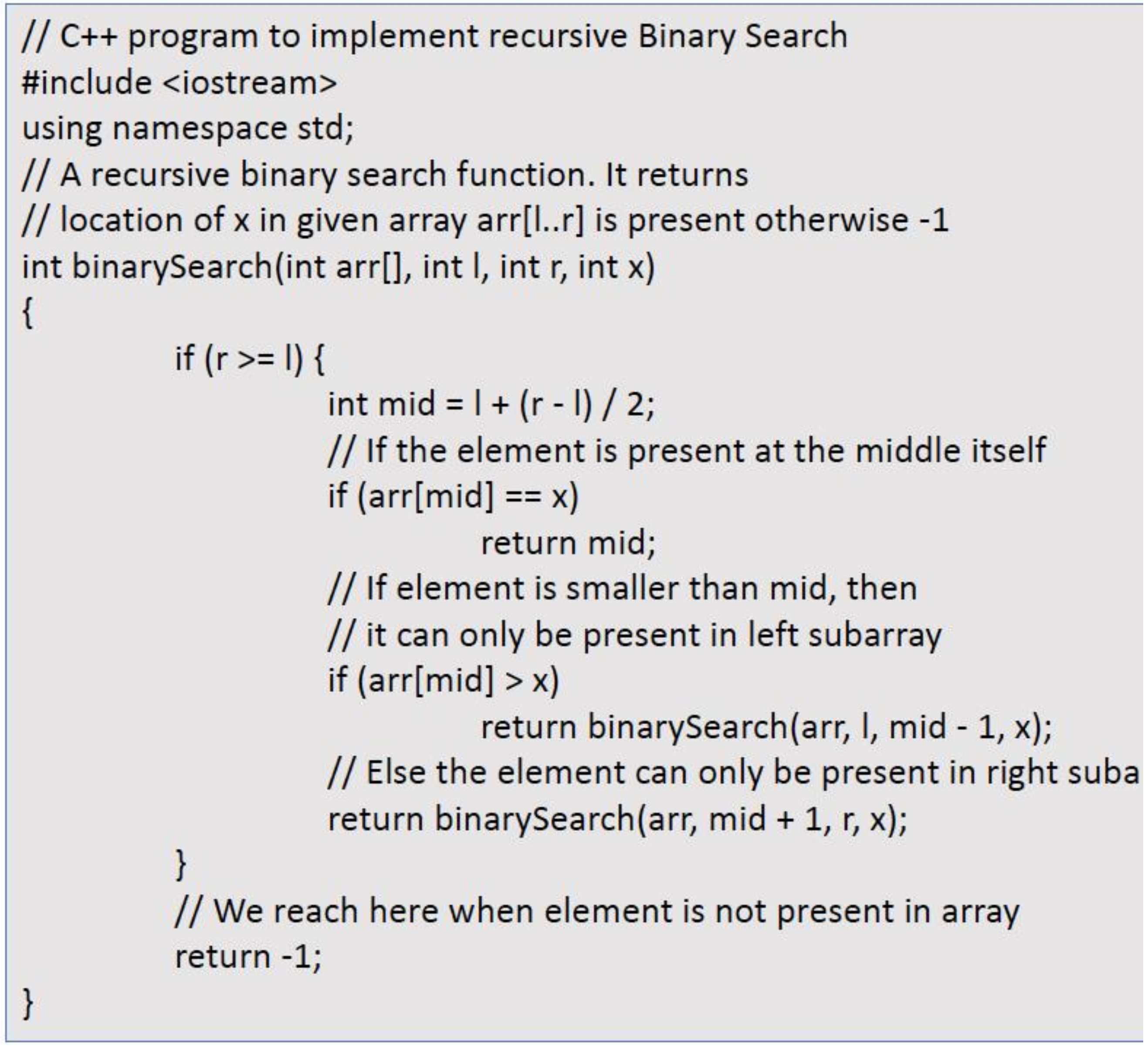
```
// C++ program to implement iterative Binary Search
#include <iostream>
using namespace std;

// An iterative binary search function.
int binarySearch(int arr[], int l, int r, int x)
{
        while (l <= r) {
                int m = l + (r - l) / 2;
                // Check if x is present at mid
                if (arr[m] == x)
                        return m;
                // If x greater, ignore left half
                if (arr[m] < x)
                        l = m + 1;
                // If x is smaller, ignore right half
                else
                        r = m - 1;
        }
        // If we reach here, then element was not present
        return -1;
}
```

Recursive Solution



```
// Driver code
int main(void)
{
    const int n=5;
    int arr[n] = { 2, 3, 4, 10, 40 };
    int x = 10;
     int result = binarySearch(arr, 0, n - 1, x);
     if (result == -1)
         cout << "Element is not present in array“;
     else
         cout << "Element is present at index " <<
result;
     return 0;
}
```



```
// C++ program to implement recursive Binary Search
#include <iostream>
using namespace std;
// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
        if (r >= l) {
                int mid = l + (r - l) / 2;
                // If the element is present at the middle itself
                if (arr[mid] == x)
                        return mid;
                // If element is smaller than mid, then
                // it can only be present in left subarray
                if (arr[mid] > x)
                        return binarySearch(arr, l, mid - 1, x);
                // Else the element can only be present in right suba
                return binarySearch(arr, mid + 1, r, x);
        }
        // We reach here when element is not present in array
        return -1;
}
```

Acknowledgment

- Content of these slides are taken from:
  - https://www.geeksforgeeks.org/
  - https://www.tutorialspoint.com/
  - https://www.programiz.com/
  - https://www.w3schools.com/