CM1602 : Data Structures and Algorithms for AI

6. Searching algorithms and Sorting algorithms

Lecture 6 - Part 1 | R. Sivaraman

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

MODULE CONTENT

Lecture	Topic
Lecture 01	Introduction to Fundamentals of Algorithms
Lecture 02	Analysis of Algorithms
Lecture 03	Array and Linked Lists
Lecture 04	Stack
Lecture 05	Queue
Lecture 06	Searching algorithms and Sorting algorithms
Lecture 07	Trees
Lecture 08	Maps, Sets, and Lists
Lecture 09	Graph algorithms

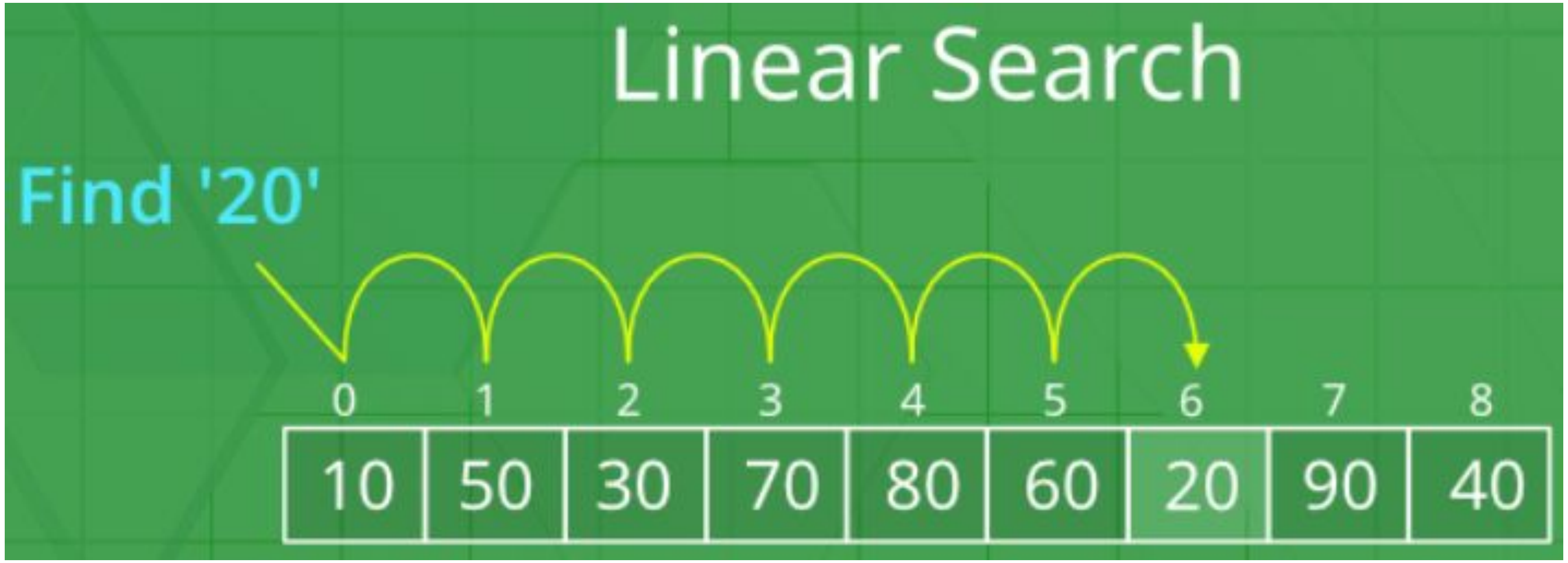
Learning Outcomes

- LO1 : Describe the fundamental concepts of algorithms and data structures.
- LO4 : Adapt and extend algorithms to real-world problems and address implementation requirements.
- On completion of this lecture, students are expected to be able to:
  - Describe and Implement and analyze Linear Search, Binary Search, and Insertion Sort
  - Analyze the performance of Searching and Sorting algorithms

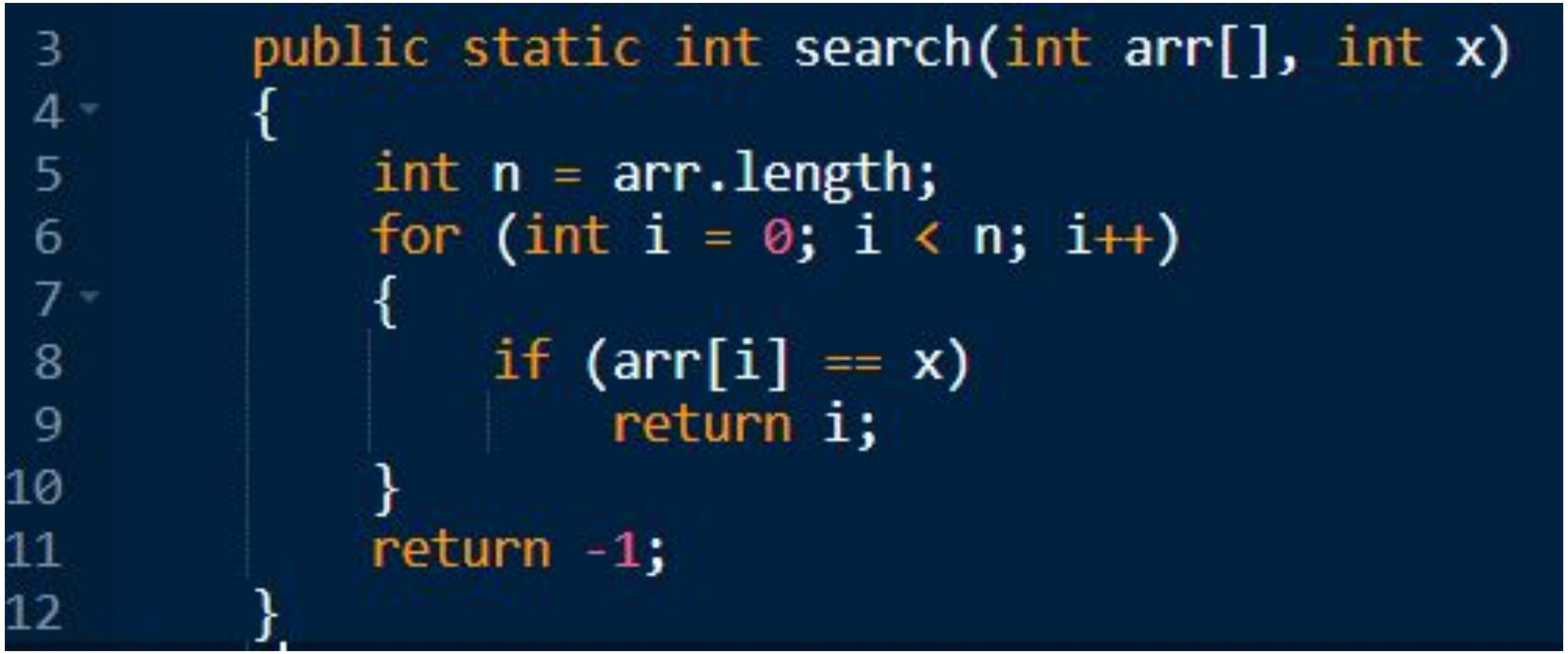
Linear Search

- Complexity - O(N).
- Linear search is a very simple search algorithm.
- Sequential search is made over all items one by one.
- Every item is checked and if a match is found then that particular item is returned.
- Otherwise the search continues till the end of the data collection.

Linear Search



Linear Search

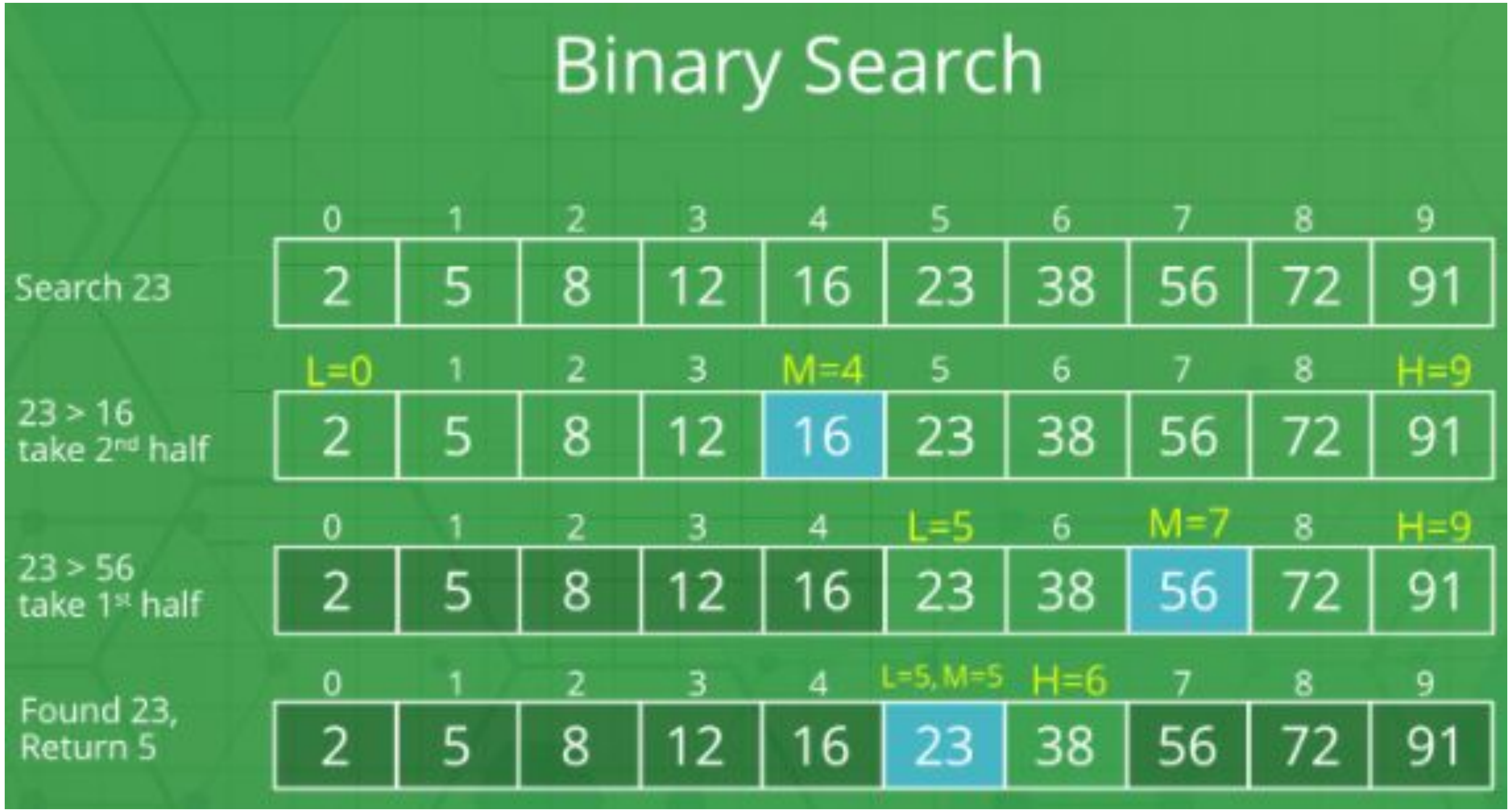


```
public static int search(int arr[], int x)
{
    int n = arr.length;
    for (int i = 0; i < n; i++)
    {
        if (arr[i] == x)
            return i;
    }
    return -1;
}
```

Binary Search

- Complexity – O(log N)
  Precondition - Array Must be sorted.
- Binary search looks for a particular item by comparing the middle most item of the collection.
- If a match occurs, then the index of item is returned.
- If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item.
- Otherwise, the item is searched for in the sub-array to the right of the middle item.
- This process continues on the sub-array as well until the size of the subarray reduces to zero.

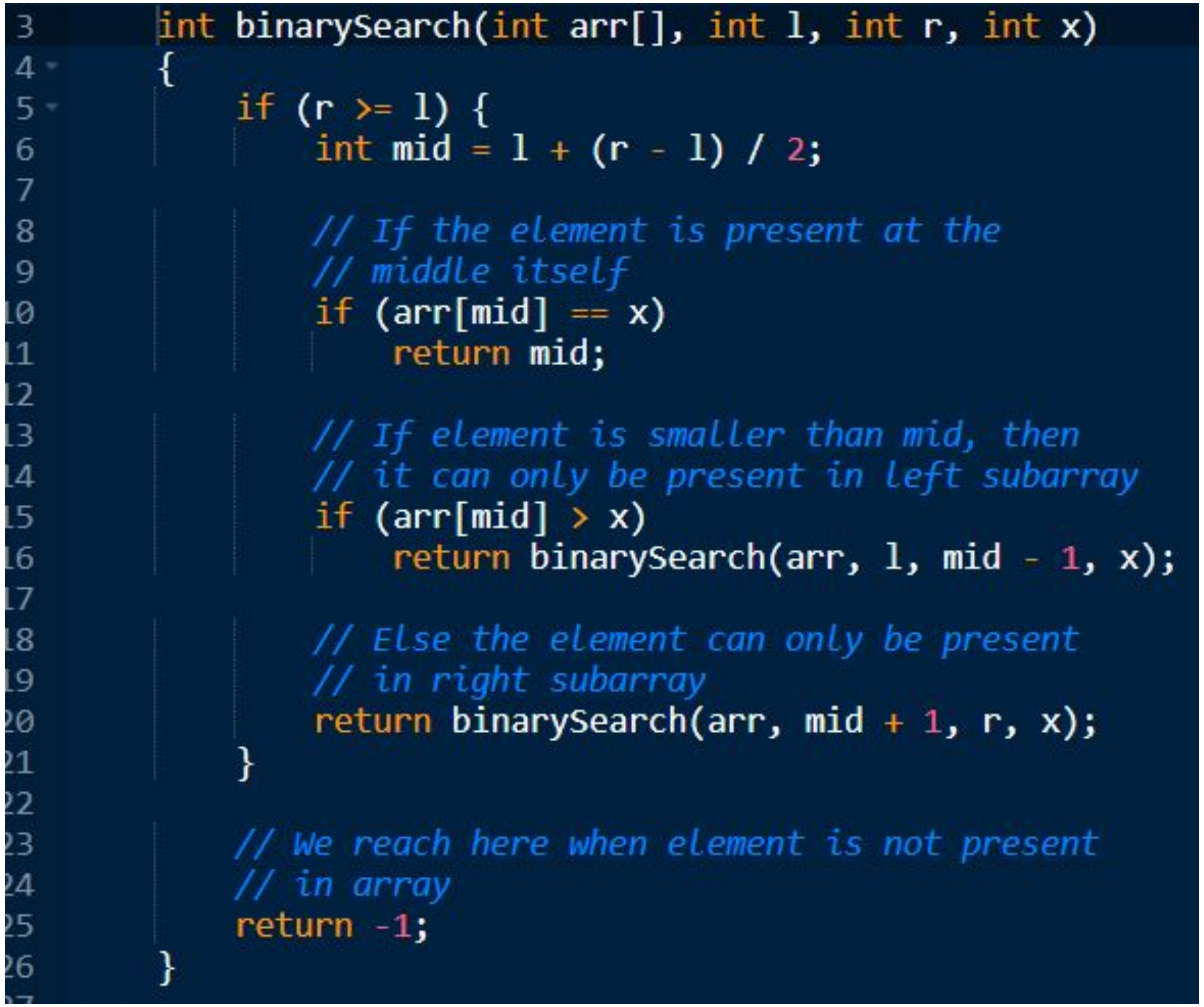
Binary Search



Binary Search

	0	1	2	3	4	5	6	7	8	9
Search 23	2	5	8	12	16	23	38	56	72	91
	L=0	1	2	3	M=4	5	6	7	8	H=9
23 > 16 take 2nd half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5	6	M=7	8	H=9
23 > 56 take 1st half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5, M=5	H=6	7	8	9
Found 23, Return 5	2	5	8	12	16	23	38	56	72	91

Binary Search

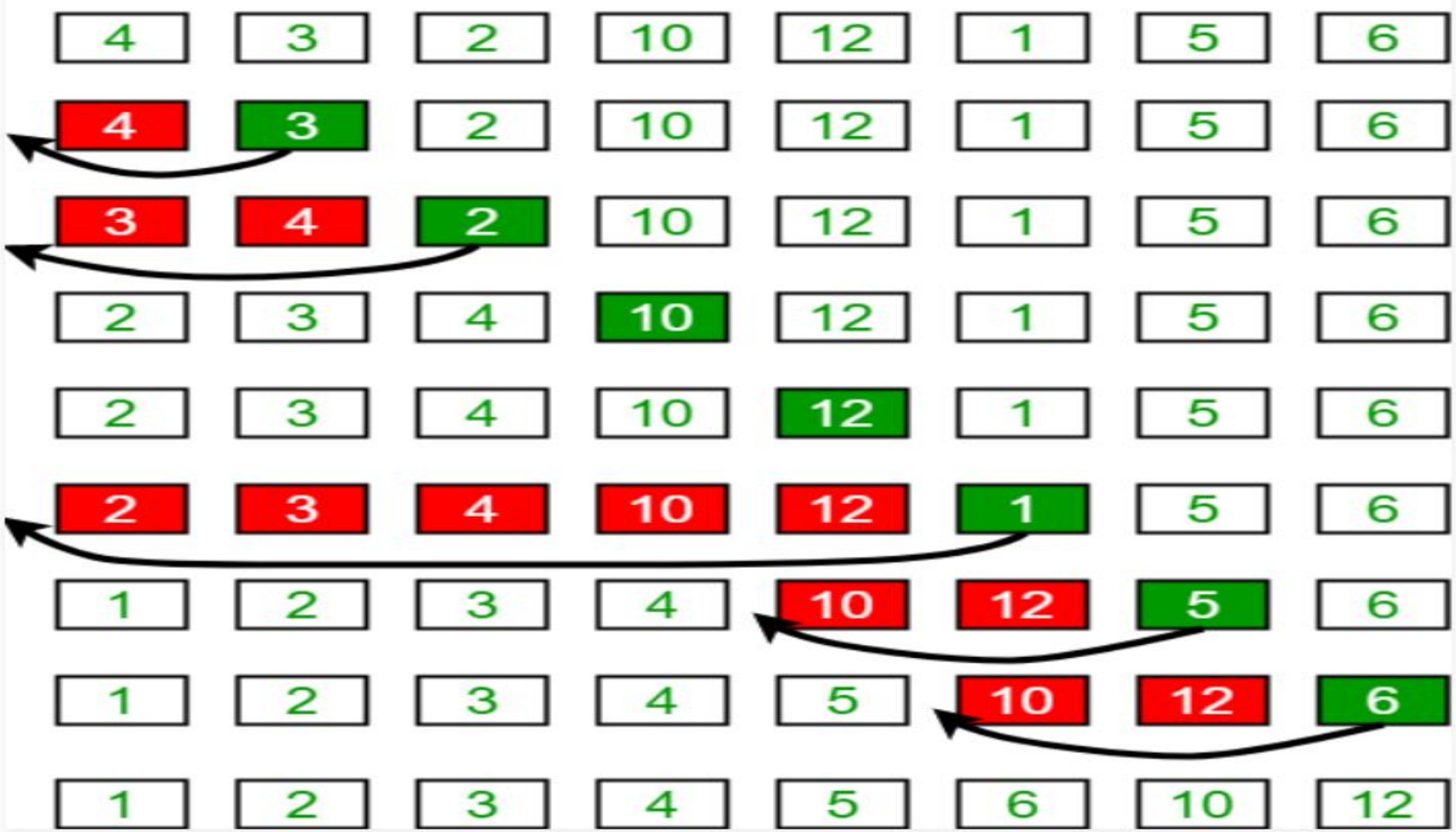


```
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        // 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 subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not present
    // in array
    return -1;
}
```

Insertion Sort

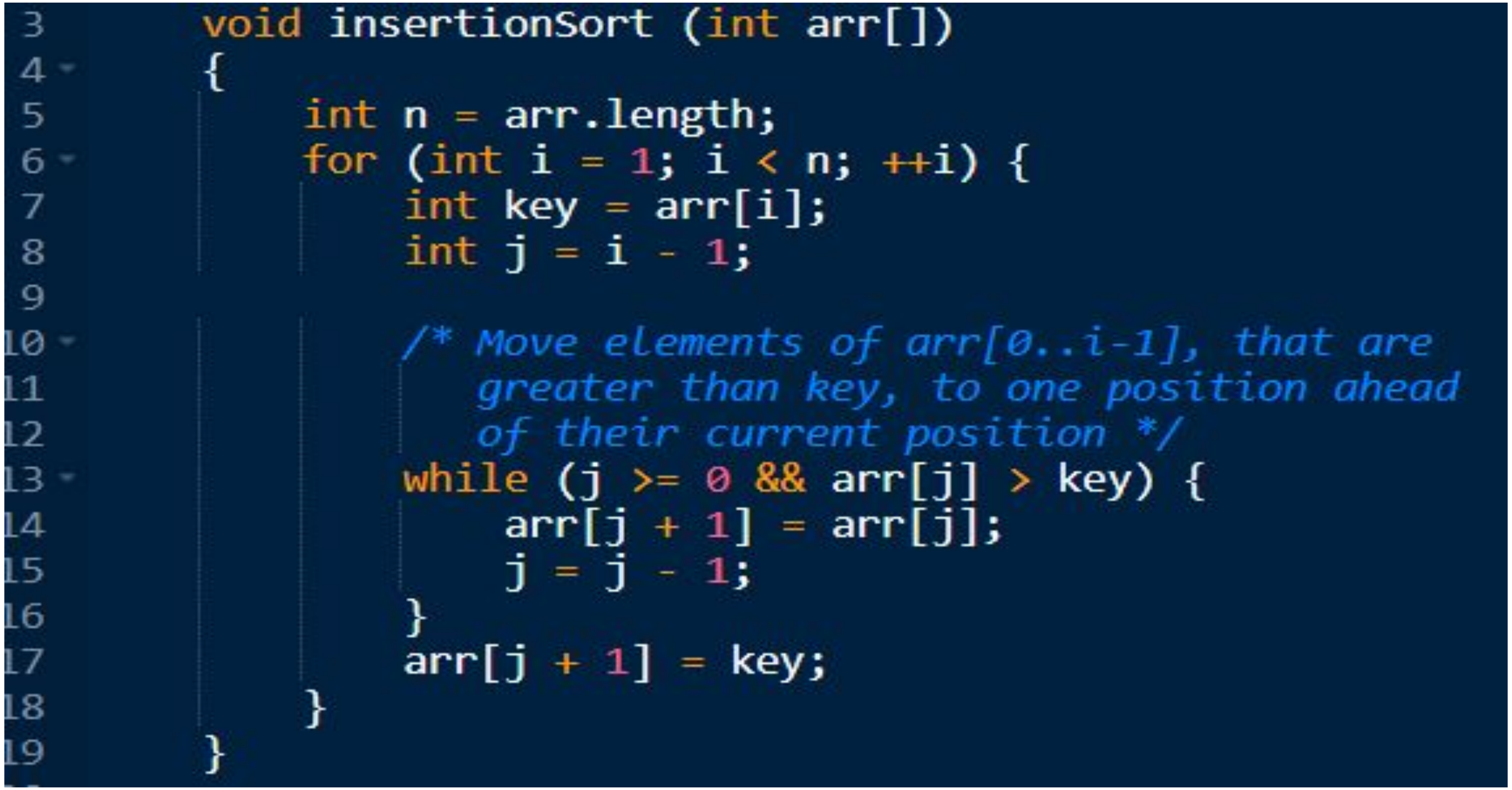
- Complexity – O(log N)
- Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands.
- The array is virtually split into a sorted and an unsorted part.
- Values from the unsorted part are picked and placed at the correct position in the sorted part.

Insertion Sort



4	3	2	10	12	1	5	6
4	3	2	10	12	1	5	6
3	4	2	10	12	1	5	6
2	3	4	10	12	1	5	6
2	3	4	10	12	1	5	6
2	3	4	10	12	1	5	6
1	2	3	4	10	12	5	6
1	2	3	4	5	10	12	6
1	2	3	4	5	6	10	12

Insertion Sort



```java
void insertionSort (int arr[])
{
    int n = arr.length;
    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```