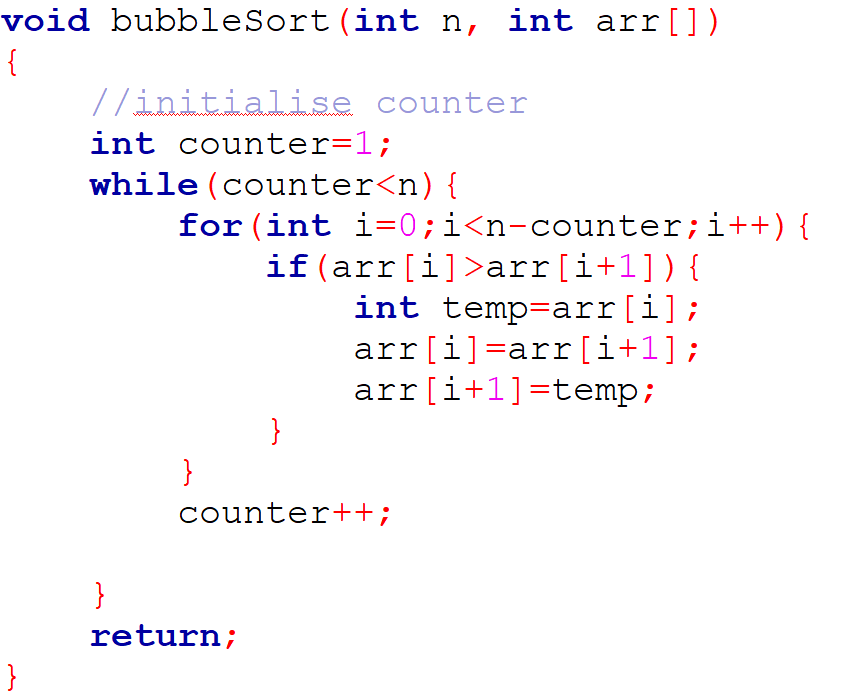
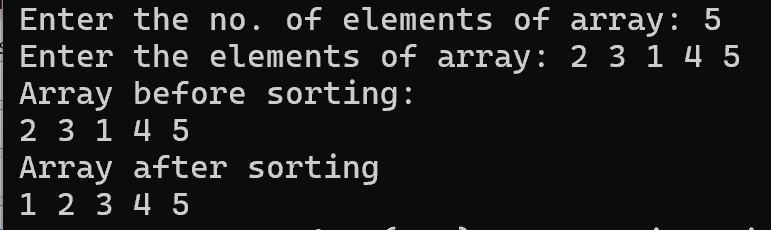
**Sorting Algorithm in CPP:**

### Bubble Sort:

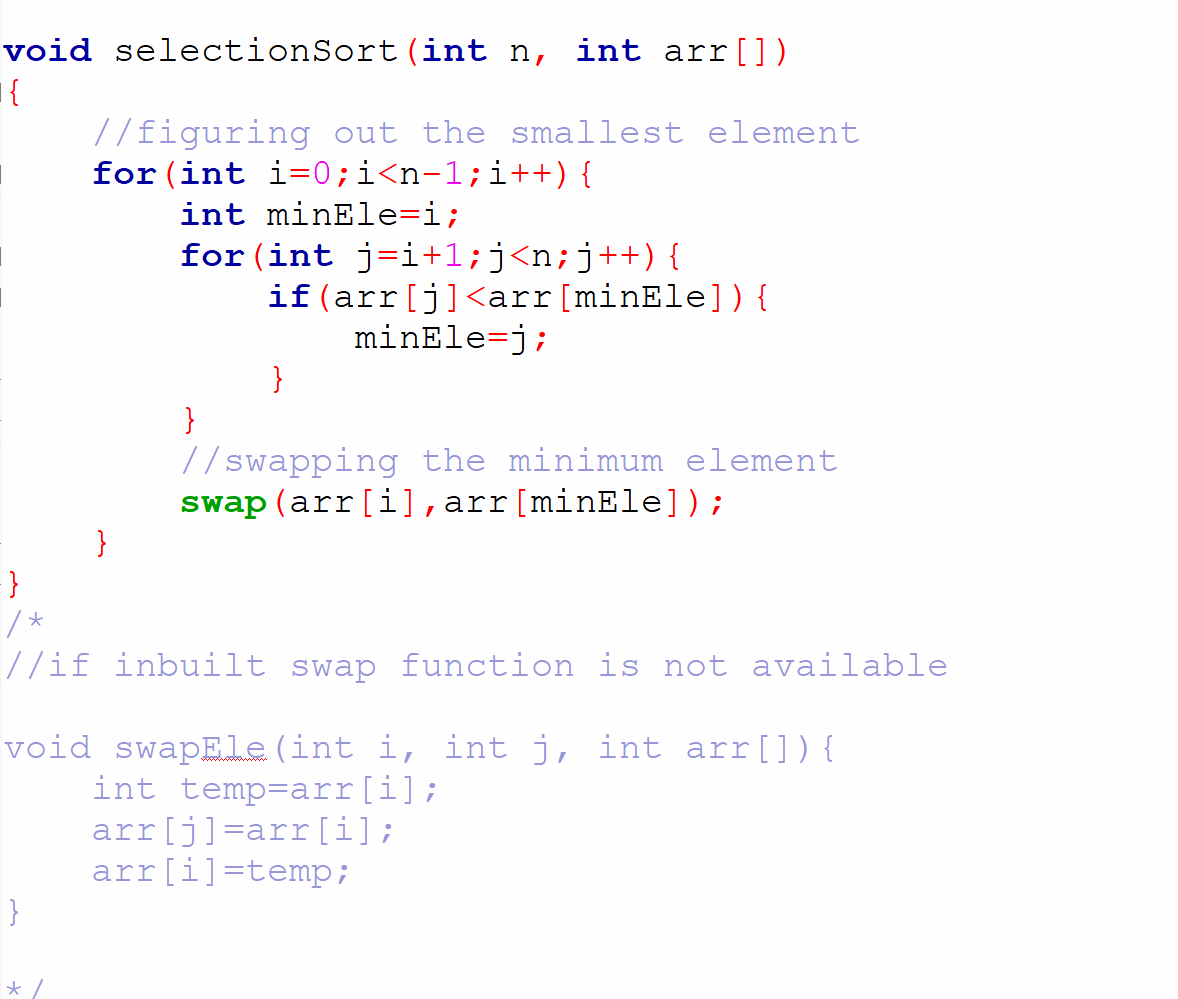
* It’s the simplest way of sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order.
* It is particularly used for sorting small datasets.
* Time complexity: O(n2 ) //not favorable

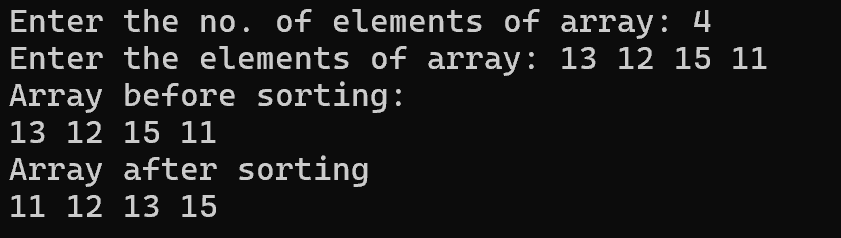




### Selection Sort:

* It’s a comparison-based sorting algorithm.
* It sorts an array by repeatedly selecting the smallest (or largest) element from the unsorted portion and swapping it with the 1st unsorted element.
* This process continues until the entire array is sorted.
* Time complexity: O(n2)



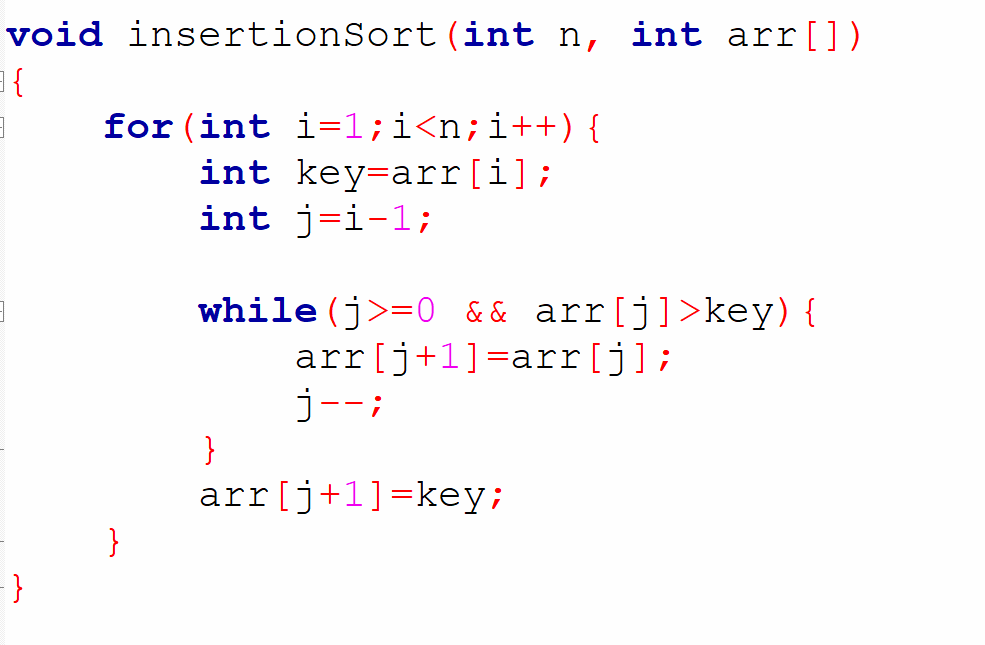


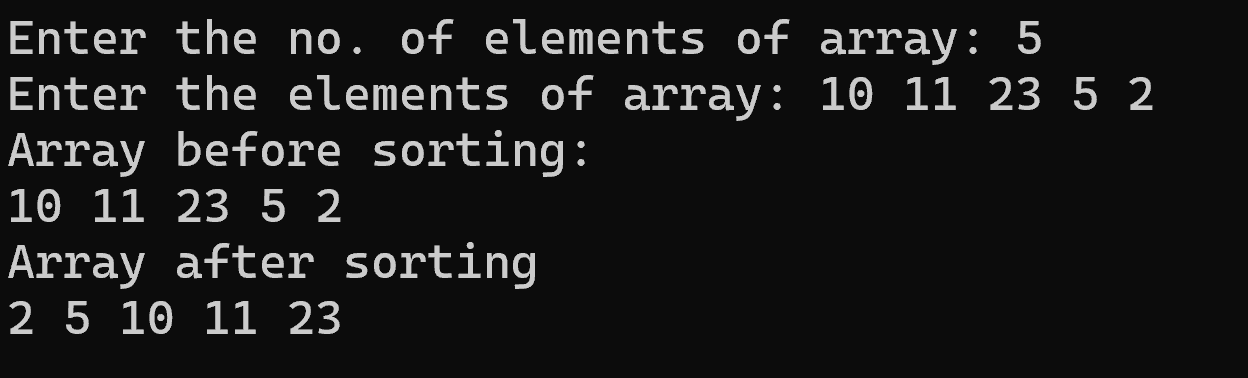
### Insertion Sort:

* It’s a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct portion in a sorted portion of the array.
* It’s like sorting playing cards in your hands.
* You split the cards into 2 groups:

1. the sorted cards
2. the unsorted cards.

* You pick a card from unsorted group and put it in the right place in the sorted group.
* Time Complexity: O(n2) *//worst case and average case*
* O(n)  *// best case time complexity if array is already sorted.*





### Quick Sort:

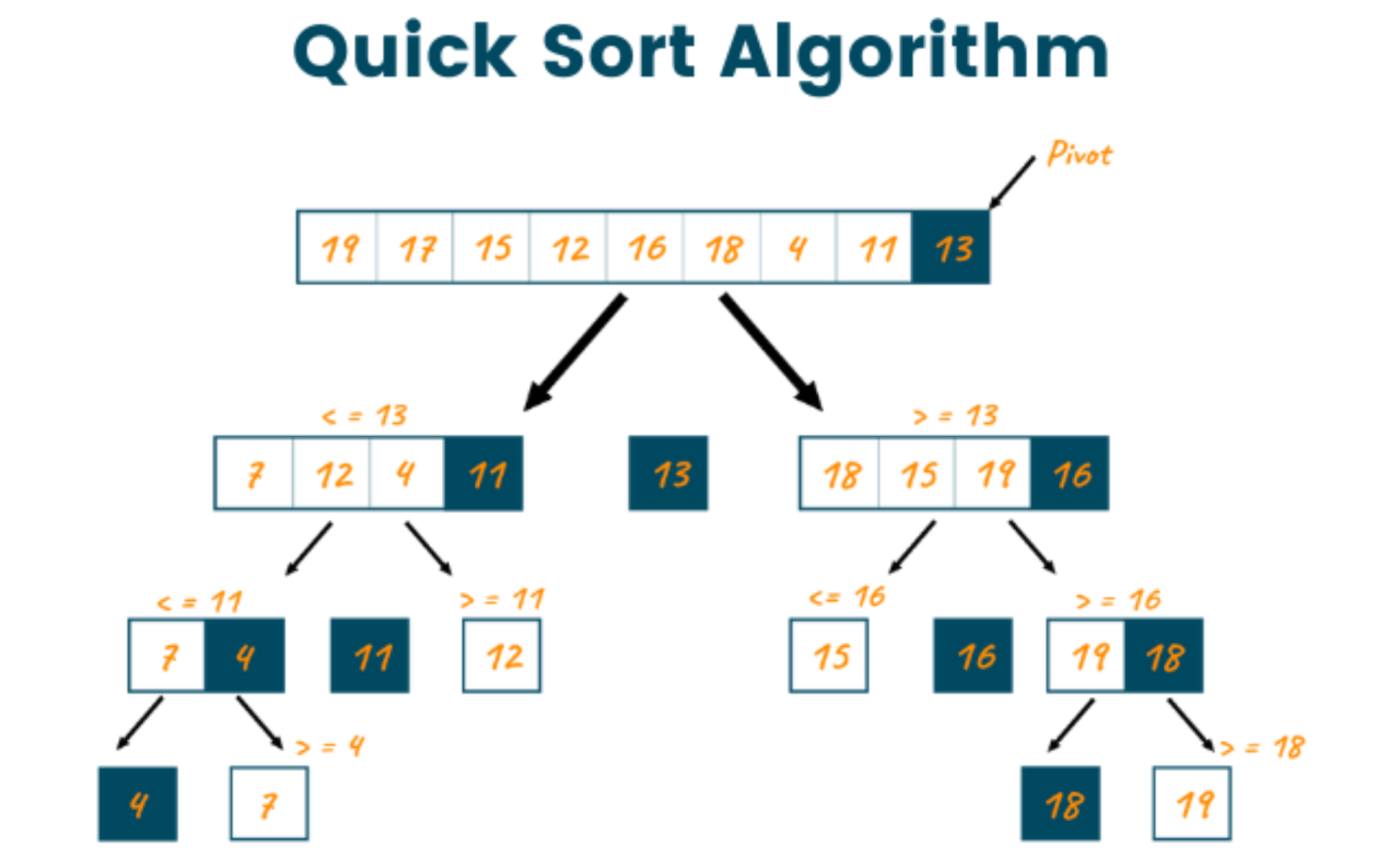
* It’s a comparison-based sorting algorithm that uses the divide & conquer approach to sort an array of elements.
* It is implemented by using 2 functions:

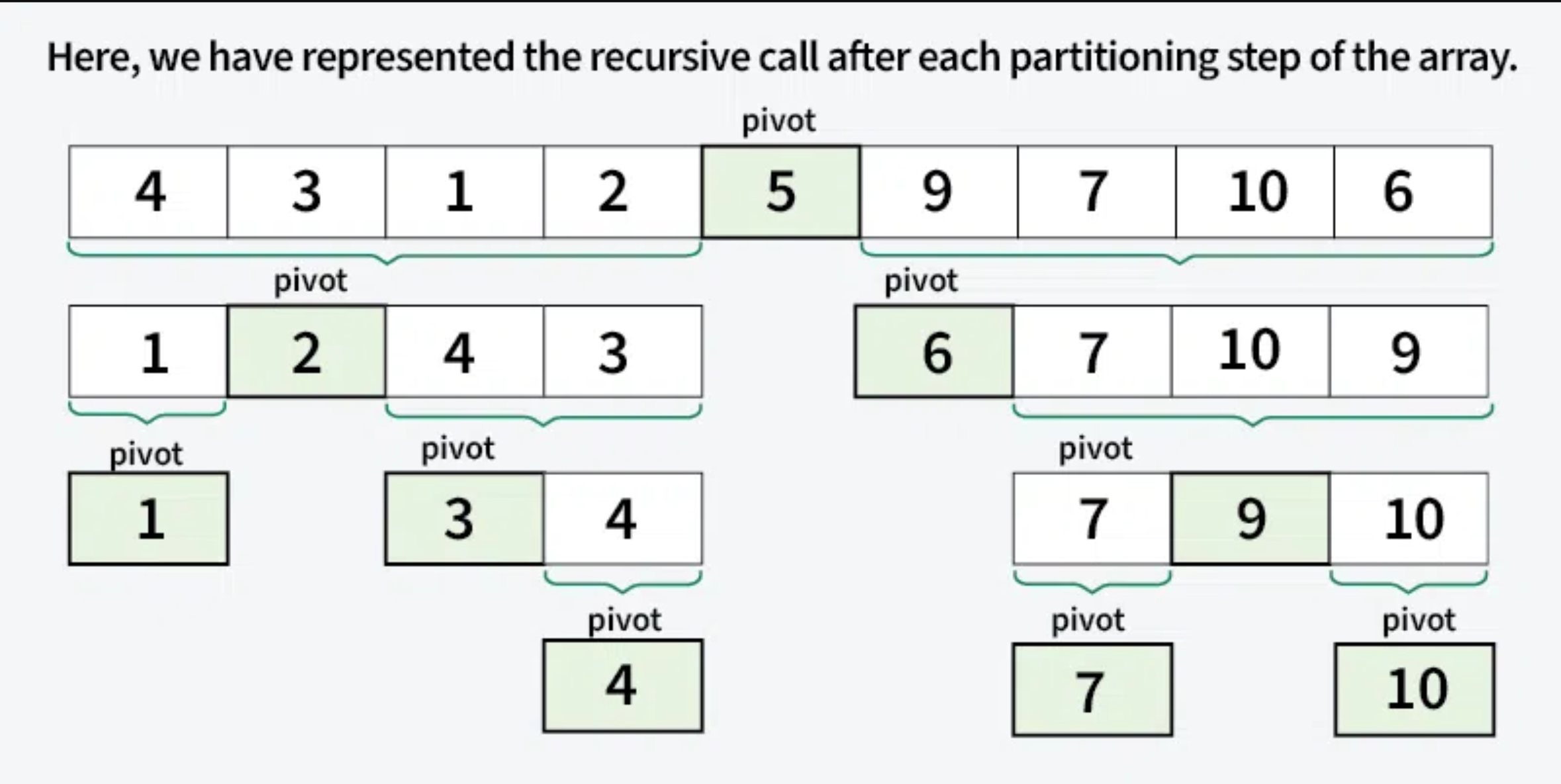
1. partition( ): It is a key process in the quick sort algorithm. It involves selecting a pivot element and rearranging the array so that all elements smaller than the pivot are placed to it’s left, and the elements greater than pivot are placed to its right. The point where the pivot is placed is called the partitioning index and it’s returned to its caller quickSort( ).
2. quickSort( ): It is the main recursive function that implements the divide & conquer strategy. It divides the given array into 2 sub-arrays based on the partitioning index returned by the partition( ). It keeps calling itself for these 2 subarrays until the whole array is sorted.

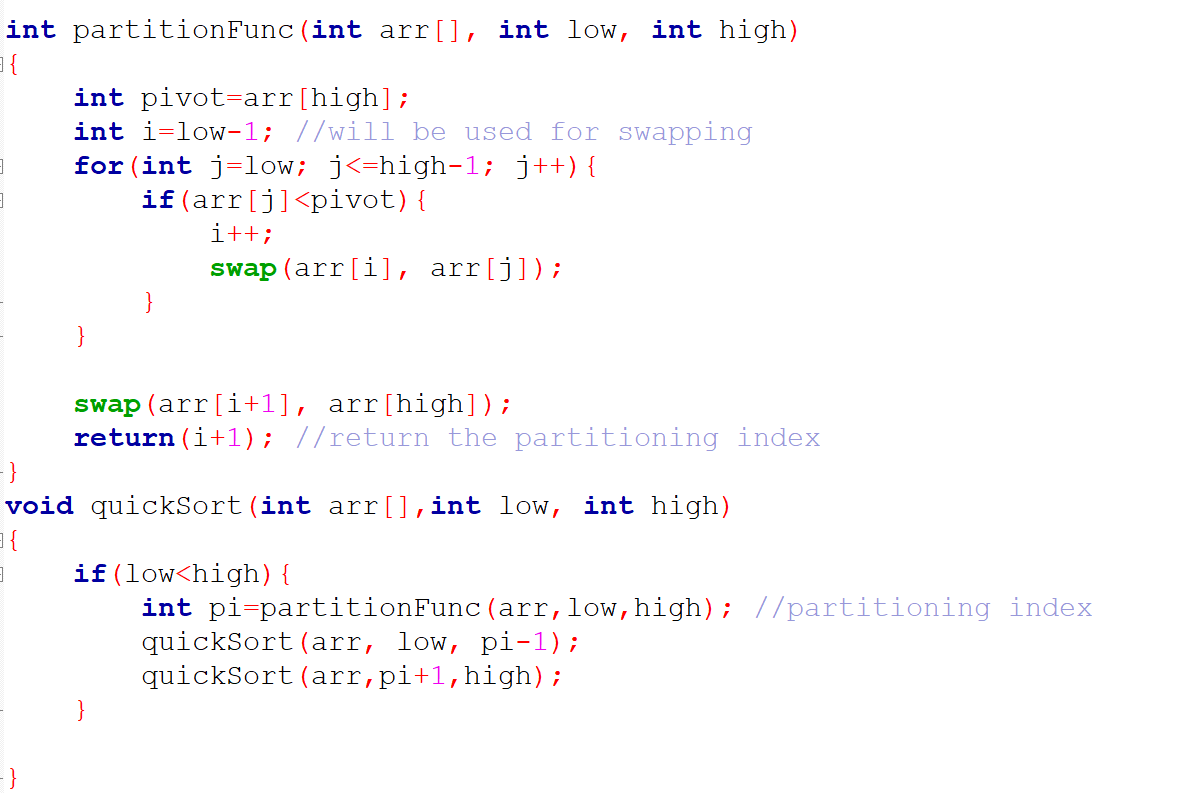
* Time complexity:

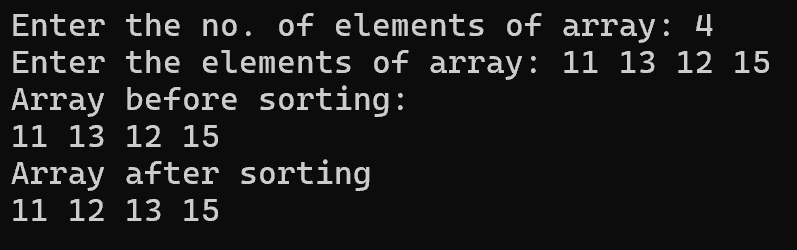
1. Worst : O(n2)
2. Best & average: O(n \* log n)

//when it’s pivot element is the middle element or near middle element









### Merge Sort:

* It’s a sorting algorithm that follows divide and conquer approach.
* It works by recursively dividing the input array into smaller sub-arrays and sorting those sub-arrays, then merging them back together to obtain the sorted array.
* In simple terms, we can say that the process of merge sort is to divide the array into 2 halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.
* Time complexity: O(n log n)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **QUICK SORT** | **BUBBLE SORT** | **MERGE SORT** | **INSERTION SORT** | **SELECTION SORT** |
| **Best time complexity** | O(n log(n)) | O(n) | O(n log(n)) | O(n) | O(n2) |
| **Average time complexity** | O(n log(n)) | O(n2) | O(n log(n)) | O(n2) | O(n2) |
| **Worst time complexity** | O(n2) | O(n2) | O(n log(n)) | O(n2) | O(n2) |
| **Space complexity** | O(log(n)) | O(1) | O(n) | O(1) | O(1) |
| **Stability** | Not stable | Stable | Stable | Stable | Not stable |
| **Adaptability** | Adapt well to partially sorted data | Doesn’t adapt well to partially sorted data | Doesn’t adapt well to partially sorted data | Adapt well to partially sorted data | Doesn’t adapt well to partially sorted data |