Unit2 Quicksort

Quicksort Algorithm

- Quicksort is a sorting algorithm based on the divide and conquer approach
- Where An array is divided into subarrays by selecting a pivot element (element selected from the array).
- While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.

- The left and right subarrays are also divided using the same approach. This process continues until each subarray contains a single element.
- At this point, elements are already sorted. Finally, elements are combined to form a sorted array.

The quick sort algorithm works as follows:

- 1. Select an element pivot from the array elements.
- 2. Rearrange the elements in the array in such a way that all elements that are less than the pivot appear before the pivot and all elements greater than the pivot element come after it (equal values can go either way). After such a partitioning, the pivot is placed in its final position.
- This is called the partition operation.

- 3. Recursively sort the two sub-arrays thus obtained. (One with sub-list of values smaller than that of the pivot element and the other having higher value elements.

Steps

- Quick Sort is a highly efficient divide-and-conquer sorting algorithm. It works by selecting a "pivot" element from the array and partitioning the other elements into two sub-arrays:
- those less than the pivot and those greater than the pivot.
- The process is then recursively applied to the sub-arrays.

- 1. Choose a Pivot
- Select an element from the array as the pivot. This can be the first element, last element, middle element, or chosen randomly.

- 2. Partition the Array
- Rearrange the array so that:
- All elements smaller than the pivot are placed to its left.
- All elements larger than the pivot are placed to its right.
- The pivot is now in its correct sorted position.

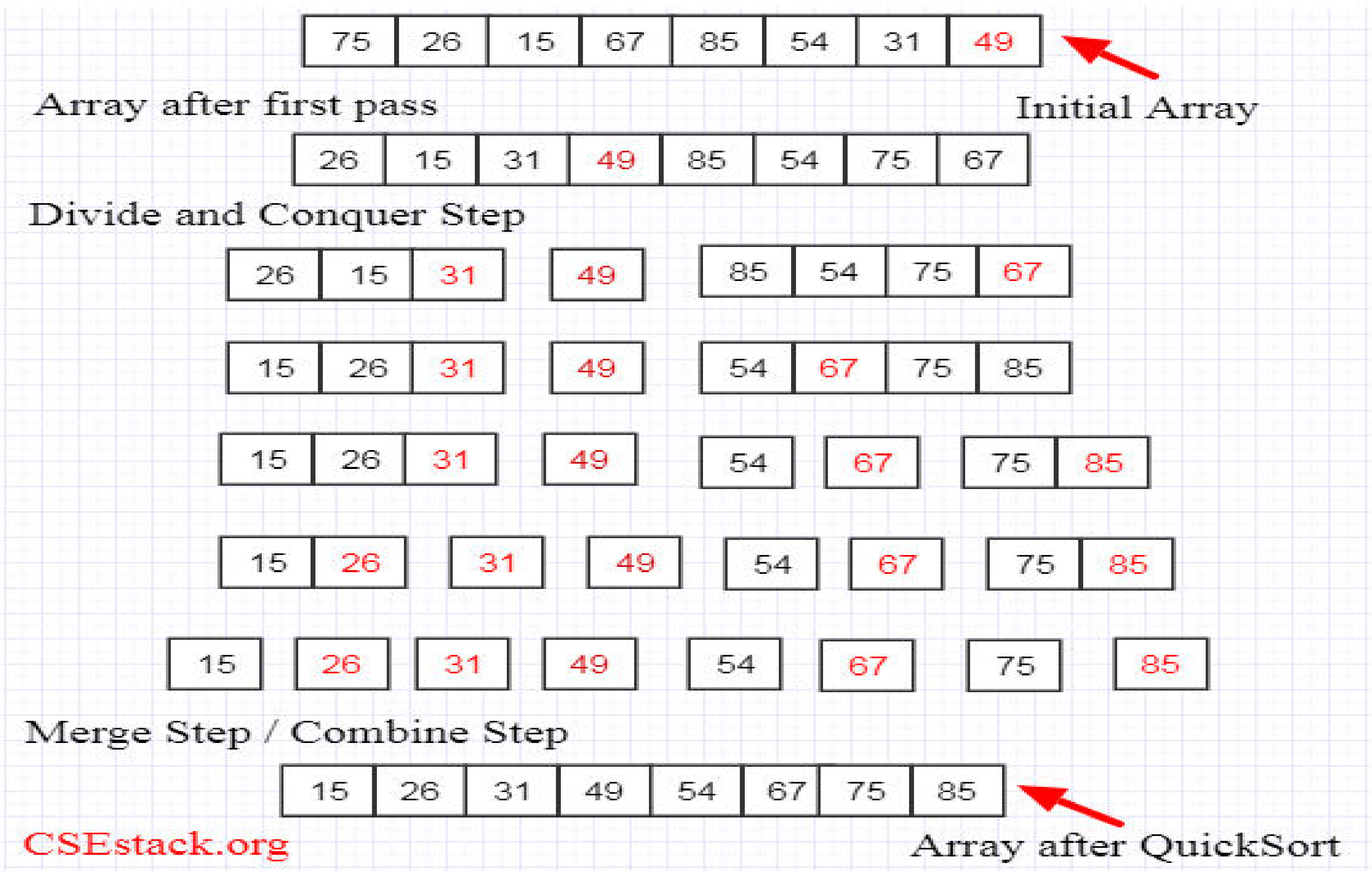
- 3. Recursively Apply Quick Sort
- Recursively apply the same process to the sub-arrays on the left and right of the pivot.
- Base Case
- The recursion stops when the sub-array has one or zero elements, as it is already sorted.

- Time Complexity
- Best/Average Case: O(n log n) (when the pivot divides the array evenly).
- Worst Case: (n^2) (when the pivot is the smallest or largest element, leading to unbalanced partitions).
- Example

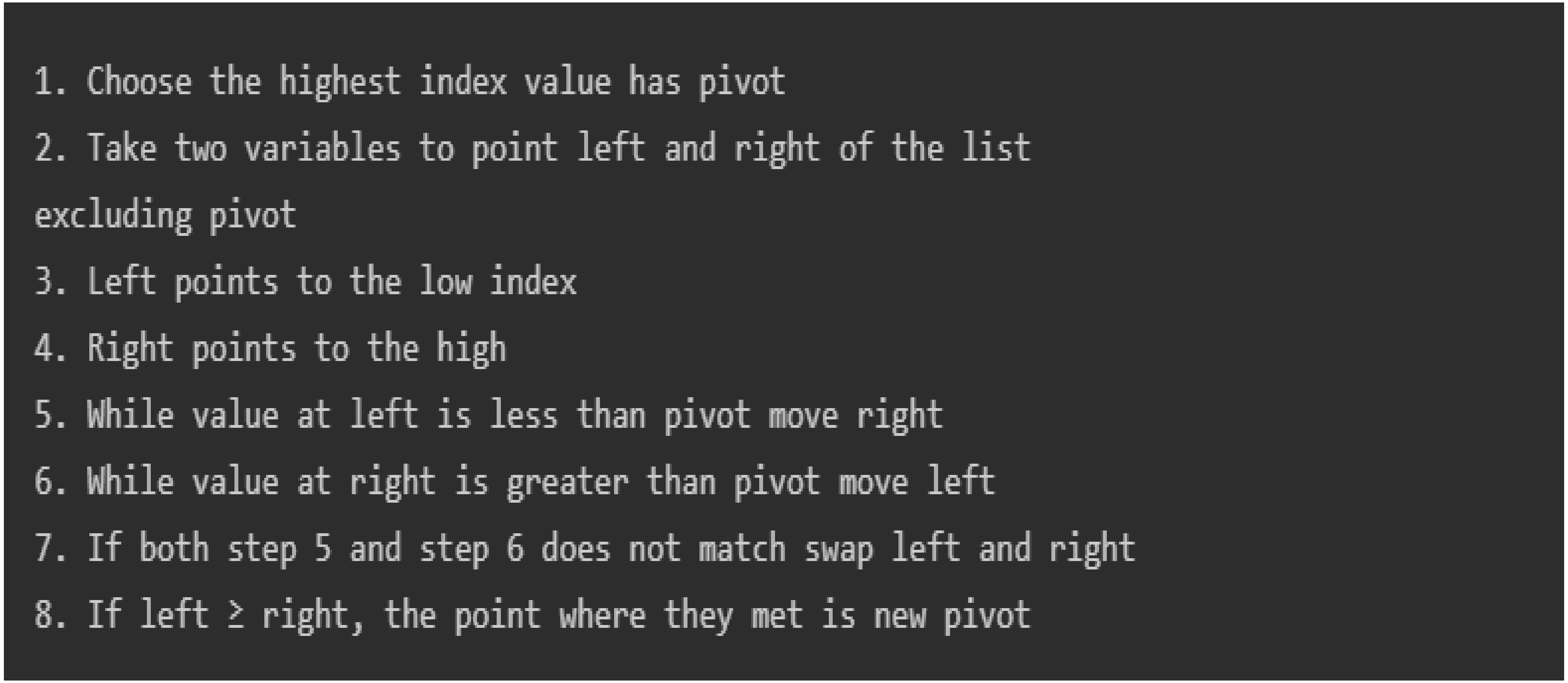
- For the array [8, 3, 1, 7, 0, 10, 2]:
- Choose pivot (e.g., 7).
- Partition: [3, 1, 0, 2] | 7 | [8, 10].
- Recursively sort [3, 1, 0, 2] and [8, 10].
- Combine results: [0, 1, 2, 3, 7, 8, 10].

- Quick Sort is popular due to its efficiency and in-place sorting (no extra memory required).
- However, it’s sensitive to pivot selection, which can affect performance.

Now, using pictorial representation, let's try to sort an array that has initial values as X= {75, 26, 15, 67, 54, 31, 49}.



Quick Sort Pivot Algorithm



```
1. Choose the highest index value has pivot
2. Take two variables to point left and right of the list
excluding pivot
3. Left points to the low index
4. Right points to the high
5. While value at left is less than pivot move right
6. While value at right is greater than pivot move left
7. If both step 5 and step 6 does not match swap left and right
8. If left ≥ right, the point where they met is new pivot
```

Ref link

- https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_algorithm.htm

- Practically, the efficiency of quick sort depends on the element which is chosen as the pivot.
- Its worst-case efficiency is given as $O(n^2)$. The worst case occurs when the array is already sorted
- (either in ascending or descending order) and the left-most element is chosen as the pivot.

Quicksort complexity

Now, let's see the time complexity of quicksort in best case, average case, and in worst case. We will also see the space complexity of quicksort.

1. Time Complexity

Case	Time Complexity
Best Case	O(n*logn)
Average Case	O(n*logn)
Worst Case	O(n^2)

- **Best Case Complexity -** In Quicksort, the best-case occurs when the pivot element is the middle element or near to the middle element. The best-case time complexity of quicksort is **O(n*logn)**.
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of quicksort is **O(n*logn)**.
- **Worst Case Complexity -** In quick sort, worst case occurs when the pivot element is either greatest or smallest element. Suppose, if the pivot element is always the last element of the array, the worst case would occur when the given array is sorted already in ascending or descending order. The worst-case time complexity of quicksort is **O(n^2)**.