Data Structures and Algorithms

Lecture 24: Quick sort – Recursive implementation

Quick Sort

This algorithm is quick or fast in terms of speed and this is one of the reasons why it is so famous. This algorithm is based on the divide and conquer approach of programming.

The divide and conquer approach involves dividing a task into small atomic sub-tasks and then solving those subtasks. The results of those subtasks are then combined and evaluated to get the final result.

Quick Sort

- In the quick sort algorithm, we select a pivot element and we position the pivot in such a manner that all the elements smaller than the pivot element are to its left and all the elements greater than the pivot are to its right.
- The elements to the left and right of the pivot form two separate arrays. Now the left and right subarrays are sorted using this same approach. This process continues until each subarray consists of a single element.
- When this situation occurs, the array is now sorted and we can merge the elements to get the required array.
- Two important characteristics of the quick sort algorithm are that it is an inplace algorithm and is not stable.

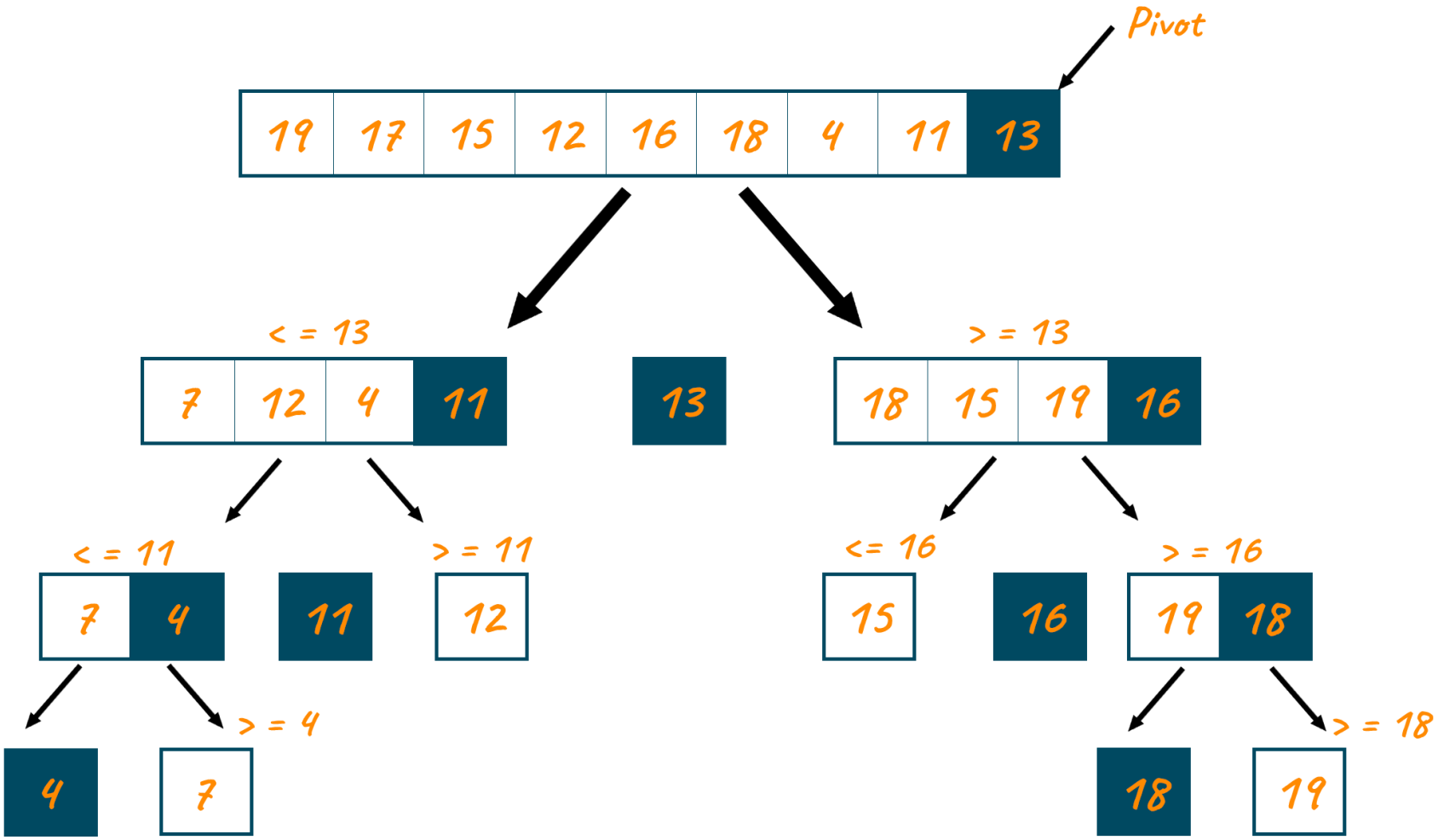
Quick Sort

How to select the pivot element?

There are 4 common ways of selecting a pivot. One can use any one of the following methods:

- Pick the first element
- Pick the last element
- Pick a random element
- Pick the median element

Example of Quick Sort



Algorithm for Quick Sort

The quick algorithm consists of two different algorithms, one for partitioning the array according to the pivot and one for actual sorting. Let us look at both the algorithms one by one. Let us first discuss the algorithm of partitioning the array.

Step 1: Take two elements low and high

Step 2: Make low store the starting index of the array

Step 3: Make high store the last index of the array

Step 4: Take a variable j and initialize it to low

Step 5: Take pivot as the last element is the array

Step 6: Traverse through the entire array using the variable i and compare each element with pivot

Step 7: If arr[i] < pivot then increment j and swap element as position j and i

Algorithm for Quick Sort

Step 8: Increment i by 1

Step 9: When the loop terminates, return j – 1

The value returned by the partitioning algorithm gives us the correct position or index of the pivot in the array. What is meant by correct position is that all elements preceding the pivot are smaller than the pivot and all the elements after the pivot are greater than it. Let us now look at the sorting algorithm.

Step 1: Take pivot as the last element in the array

Step 2: Find the correct index of the pivot using the partition algorithm

Step 3: Recursively call Quicksort on the left subarray

Step 4: Recursively call Quicksort on the right subarray

Algorithm for Quick Sort

```
void quickSort(int arr[], int low, int high){
        if(low < high){
        int pivot = arr[high];
        int pos = partition(arr, low, high, pivot);

        quickSort(arr, low, pos-1);
        quickSort(arr, pos+1, high);
        }
}
```

Time complexity and space complexity

The running time complexity of quicksort for the best case and the average case is O(N log N). Whereas the time complexity is for the worst case is O(N2). Coming to the space complexity, since the quick sort algorithm doesn't require any additional space other than that to store the original array, therefore, the space complexity of the quick sort algorithm is O(N). N is the size of the input array.

Applications of Quick Sort

The various fields where quicksort is used are:

- Commercial computing
- Numerical computations
- Information searching

Advantages of quicksort

- The average-case time complexity to sort an array of n elements is O(n lg n).
- Generally, it runs very fast. It is even faster than merge sort.
- No extra storage is required

Disadvantages of quicksort

- Its running time can be different for different array contents.
- The worst-case quick sort takes place when the array is already sorted.
- It is not stable.