DSME

Quicksort

Data Structures Made Easy

DUBLIN CITY UNIVERSITY

Contents

1. *Definition* 3
2. *Implementation* 3
3. *Example* 3
4. *Functions* 4
5. *Pseudocode* 4
6. *Complexity* 5
7. *Advantages of Quicksort* 5
8. *Disadvantages of Quicksort* 5
9. *References* 5

1. ***Definition***

Quicksort is a sorting algorithm that was created by C. A. R. Hoare. It is a sorting algorithm that operates by recursively partitioning elements, which are to be sorted into two separate sets. Quicksort is based on the 'divide and conquer' principal.

2. ***Implementation***

Quicksort works as follows:

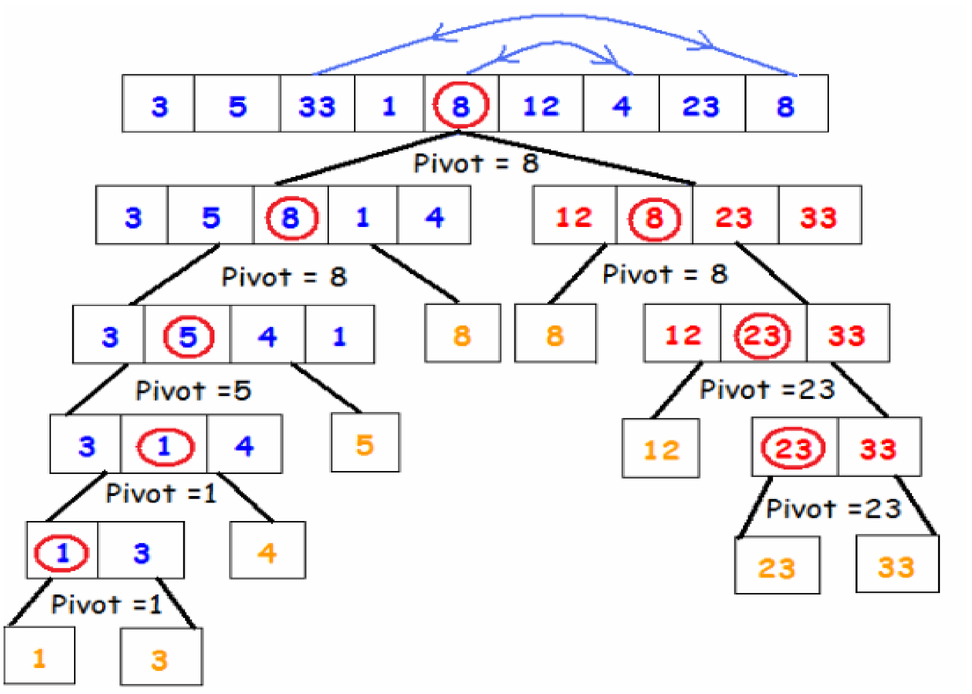
1. Select an element of the array to be the pivot.

2. Divide the remaining elements into two partitions.
    - ➢ Elements that are less than the pivot are to be placed in the first partition.
    - ➢ Elements that are greater than the pivot are to be placed in the second partition.

3. Sort both partitions using recursion.

4. Combine the following in order:
    - ➢ The first (sorted) partition.
    - ➢ The pivot.
    - ➢ The second (sorted) partition.

3. ***Example***



4. ***Functions***

The implementation of Quicksort is based on the following functions

- **quick ()**
  This function selects an element from the array, which is called the pivot. The remaining elements are compared from left to right until a greater element is located and the values are swapped. The same comparison technique is repeated from the right to left with the pivot. When a smaller element is located, the values are swapped.

  This process is repeated until all elements that are less than the pivot are positioned before the pivot and all the elements that are greater than the pivot are positioned after the pivot. Partitioning takes place and the function recursively sorts the sub-array's of lesser and greater elements.

5. ***Pseudocode***

```
function quick ( array arr, low, high ){

        lo = low
        hi = high

        if lo is greater than hi
                return

        pivot = midpoint of arr

        while lo is less than hi
                while arr[lo] is less than pivot
                        increment lo

                while arr[hi] is greater than pivot
                        decrement hi


        if hi is less than lo
                swap ( hi, lo )

        quick ( arr, low, lo )
        quick ( arr, if lo equals low then increment lo else lo, high )
}
```

6. ***Complexity***

The optimal time complexity of Quicksort is O(n). This is achieved by dividing the array into two sets that are nearly identical. If the array contains n elements, then the first process will need O(n). Sorting the remaining sub-arrays will result in the average time complexity of O(n Log n).

The worst case time complexity is O(n^2). This arises if the sorting algorithm selects only one element in each iteration.

7. ***Advantages of Quicksort***

Quicksort is one of the fastest algorithms to be used and is believed to be the best sorting algorithm. This is due to the fact that it is capable of sorting a large list of items in a satisfactory amount of time.

Quicksort does not require additional memory as it is processes in-place (sorting within the array). Developers regarded this as a huge beneficiary, in terms of memory storage.

8. ***Disadvantages of Quicksort***

Quicksort's worst-case performance contributes mainly to its disadvantages. Quicksort's worst-case performance resembles the average performances of other sorts, such as selection sort, insertion sort and bubble sort.

9. ***References***

Vogel L. 2012. *Quicksort in Java* [Online]. Available from: http://www.vogella.com/articles/JavaAlgorithmsQuicksort/article.html

Jitomirsky E. 2013. *The Quicksort Algorithm* [Online]. Available from: http://www.mycstutorials.com/articles/sorting/quicksort