Quicksort - Cprogramming.com

by Jakub Bomba (axon)

When deciding on the best sorting algorithm we often look at its worst-case running time, and base our decision solely on that factor. That is why beginning programmers often overlook quicksort as a viable option because of its T(n^2) worst-case running time, which could be made exponentially unlikely with a little effort. In fact, quicksort is the currently fastest known sorting algorithm and is often the best practical choice for sorting, as its average expected running time is O(n log(n)).

Quicksort, like mergesort, is a divide-and-conquer recursive algorithm. The basic divide-and-conquer process for sorting a subarray S[p..r] is summarized in the following three easy steps:

Divide: Partition S[p..r] into two subarrays S[p..q-1] and S[q+1..r] such that each element of S[p..q-1] is less than or equal to S[q], which is, in turn, less than or equal to each element of S[q+1..r]. Compute the index q as part of this partitioning procedure

Conquer: Sort the two subarrays S[p...q-1] and S[q+1..r] by recursive calls to quicksort.

Combine: Since the subarrays are sorted in place, no work is needed to combing them: the entire array S is now sorted.

Before a further discussion and analysis of quicksort a presentation of its implementation procedure below:

```
QUICKSORT(S, P, r)
1 If p < r
2       then q <- PARTITION(S, p, r)
3             QUICKSORT(S, p, q-1)
4             QUICKSORT(S, q+1, r)
```

note: to sort the whole array S, the initial parameters would be: QUICKSORT(S, 1, length[A])

```
PARTITION(S, p, r)
1 x <- S[r]
2 i <- p-1
```

```
for j <- p to r-1
      do if S[j] <= x
            then i <- i+1
                  swap S[i] <-> S[j]
swap S[i+1] <-> S[r]
return i+1
```

Quicksort's running time depends on the result of the partitioning routine - whether it's balanced or unbalanced. This is determined by the **pivot** element used for partitioning. If the result of the partition is unbalanced, quicksort can run as slowly as insertion sort; if it's balanced, the algorithm runs asymptotically as fast as merge sort. That is why picking the "best" pivot is a crucial design decision.

> ***The Wrong Way:*** the popular way of choosing the pivot is to use the first element; this is acceptable only if the input is random, but if the input is presorted, or in the reverse order, then the first elements provides a bad, unbalanced, partition. All the elements go either into S[p...q-1] or S[q+1..r]. If the input is presorted and as the first element is chosen consistently throughout the recursive calls, quicksort has taken quadratic time to do nothing at all.
>
> ***The Safe Way:*** the safe way to choose a pivot is to simply pick one randomly; it is unlikely that a random pivot would consistently provide us with a bad partition throughout the course of the sort.
>
> ***Median-of-Three Way:*** best case partitioning would occur if PARTITION produces two subproblems of almost equal size - one of size [n/2] and the other of size [n/2]-1. In order to achieve this partition, the pivot would have to be the median of the entire input; unfortunately this is hard to calculate and would consume much of the time, slowing down the algorithm considerably. A decent estimate can be obtained by choosing three elements randomly and using the median of these three as the pivot.

Short Example of a Quicksort Routine (Pivots chosen "randomly")

```
Input: [13 81 92 65 43 31 57 26 75 0]
Pivot: 65
Partition:  [13 0 26 43 31 57]  65  [ 92 75 81]
Pivot: 31  81
Partition: [13 0 26]  31  [43 57]  65  [75]  81  [92]
Pivot: 13
Partition: [0]  13  [26]  31  [43 57]  65  [75]  81  [92]
Combine: [0 13 26]  31 [43 57]  65  [75 81 92]
Combine: [0 13 26 31 43 57]  65  [75 81 92]
Combine: [0 13 26 31 43 57 65 75 81 92]
```

Summary

Quicksort is a relatively simple sorting algorithm using the divide-and-conquer recursive procedure. It is the quickest comparison-based sorting algorithm in practice with an average running time of O(n log(n)). Crucial to quicksort's speed is a balanced partition decided by a well chosen pivot. Quicksort has the advantage of sorting in place, and it works well even in virtual memory environments.

Previous: Merge Sort