Quicksort Algorithm

Zezhou Jing

Revised September 2, 2016

Algorithmic Paradigm

Divide and conquer by recursively dividing a problem instance into two or more sub-problems of the same or related type, until they are basic and trivial enough. Solve the base problems directly and combine the solutions of all sub-problems to yield a solution to the original problem.

Performance

Let n be the length of the input array to be sorted ($n \geq 1$).

Time complexity:

- $\mathcal{O}(n \log n)$ (in fact $\Theta(n \log n)$) on average;
- $\Omega(n \log n)$ in the best case;
- $\mathcal{O}(n^2)$ in the worst case.

Space complexity:

- $\mathcal{O}(1)$ (in fact $\theta(n \log n)$) on average;
- $\mathcal{O}(1)$ in the best case;
- $\mathcal{O}(\log n)$ in the worst case.

Algorithm

- Selecting a *pivot* element from the given array (or sub-arrays): the first element of the array (naive), randomized pivot selection, the median-of-three pivot selection.
- Partitioning (and "sorting") around the pivot: rearrange the array such that all elements with values less than the pivot are ordered to the left of the pivot and elements with values more than the pivot are ordered to the right of the pivot. The pivot is in its "rightful position" (i.e. its position is finalized). The partitioning operation is usually done in linear $\mathcal{O}(n)$ time without auxiliary space. After the partitioning, the input array is divided into two sub-arrays, namely the array to the left of the pivot and the array to the right of it.
- Recursively applying the selecting and partitioning subroutines to the left and right sub-arrays until reaching the base case, which leaves only 0 or 1 element in the sub-array. An array of length 0 or 1 is trivially sorted.