Sorting and Order Statistics: Problems and Solutions

Order Statistics:

Q. Given a set of n numbers, you need to find the i largest in sorted order using a comparison-based algorithm. What is the best complexity you can achieve?

Solution:

Use an order-statistic algorithm to find the ith largest number, partition around that number, and sort the i largest numbers.

The running time of finding and partitioning around the ith largest number is $O(n)$, and the running time of sorting the i largest numbers is $O(i \lg i)$. Therefore, the total running time is $O(n + i \lg i)$.

Q. Describe an $O(n)$-time algorithm that, given a set S of n distinct numbers and a positive integer $k \leq n$, determines the k numbers in S that are closest to the median of S.

Solution:

Find the median in $O(n)$; create a new array, each element is the absolute value of the original value subtract the median; find the kth smallest number in $O(n)$, then the desired values are the elements whose absolute difference with the median is less than or equal to the kth smallest number in the new array.

Linear Time Sorting

Q. You are given an array of n dates in the $dd - mm - yyyy$ format. Propose a linear-time algorithm to sort the array in the usual increasing order (chronological order).

Q. Suppose that n points are chosen uniformly inside a circle of radius r (that is, the probability of choosing a point in any region R of area a inside the circle is $a/(\pi r^2)$). Give an algorithm that sorts the n given points with respect to their distances from the center of the circle in expected linear time.

Hint: The first one is just counting sort multiple times. The second problem will use bucket sort. Think about the buckets such that each has O(1) points on average, and you can concatenate the outputs to get the final answer.

Lower bound for comparison-based sorting: $\Omega(n\log n)$

Question: Suppose we consider the problem: "order the input array so that the smallest $n/2$ come before the largest $n/2$"? Does our lower bound still hold for that problem, or where does it break down? How fast can you solve that problem?

Answer: No, the proof does not still hold. It breaks down because any given input can have multiple correct answers. E.g., for input `[2 1 4 3]`, we could output any of $[a_1, a_2, a_3, a_4]$, $[a_2, a_1, a_3, a_4]$, $[a_1, a_2, a_4, a_3]$, or $[a_2, a_1, a_4, a_3]$. In fact, not only does the lower bound break down, but we can actually solve this problem in linear time: just run the linear-time median-finding algorithm and then make a second pass putting elements into the first half or second half based on how they compare to the median.

Q. Does there exist a comparison sort of 5 numbers that uses at most 6 comparisons in the worst case?

Solution: **False.** The number of leaves of a decision tree which sorts 5 numbers is 5! and the height of the tree is at least $\lg(5!)$. Since $5! = 120$, $2^6 = 64$, and $2^7 = 128$, we have $6 < \lg(5!) < 7$. Thus at least 7 comparisons are required.