Selected Solutions for Chapter 9: Medians and Order Statistics

Solution to Exercise 9.3-1

For groups of 7, the algorithm still works in linear time. The number of elements greater than x (and similarly, the number less than x) is at least

$$4\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{7}\right\rceil\right\rceil-2\right)\geq\frac{2n}{7}-8\,,$$

and the recurrence becomes

$$T(n)\leq T(\lceil n/7\rceil)+T(5n/7+8)+O(n)\,,$$

which can be shown to be $O(n)$ by substitution, as for the groups of 5 case in the text.

For groups of 3, however, the algorithm no longer works in linear time. The number of elements greater than x, and the number of elements less than x, is at least

$$2\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{3}\right\rceil\right\rceil-2\right)\geq\frac{n}{3}-4\,,$$

and the recurrence becomes

$$T(n)\leq T(\lceil n/3\rceil)+T(2n/3+4)+O(n)\,,$$

which does not have a linear solution.

We can prove that the worst-case time for groups of 3 is $\Omega(n\lg n)$. We do so by deriving a recurrence for a particular case that takes $\Omega(n\lg n)$ time.

In counting up the number of elements greater than x (and similarly, the number less than x), consider the particular case in which there are exactly $\left\lceil\frac{1}{2}\left\lceil\frac{n}{3}\right\rceil\right\rceil$ groups with medians $\geq x$ and in which the "leftover" group does contribute 2 elements greater than x. Then the number of elements greater than x is exactly $2\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{3}\right\rceil\right\rceil-1\right)+1$ (the -1 discounts x's group, as usual, and the $+1$ is contributed by x's group) $=2\lceil n/6\rceil-1$, and the recursive step for elements $\leq x$ has $n-(2\lceil n/6\rceil-1)\geq n-(2(n/6+1)-1)=2n/3-1$ elements. Observe also that the $O(n)$ term in the recurrence is really $\Theta(n)$, since the partitioning in step 4 takes $\Theta(n)$ (not just $O(n)$) time. Thus, we get the recurrence

$$T(n)\geq T(\lceil n/3\rceil)+T(2n/3-1)+\Theta(n)\geq T(n/3)+T(2n/3-1)+\Theta(n)\,,$$

from which you can show that $T(n)\geq cn\lg n$ by substitution. You can also see that $T(n)$ is nonlinear by noticing that each level of the recursion tree sums to n.

In fact, any odd group size ≥ 5 works in linear time.

Solution to Exercise 9.3-3

A modification to quicksort that allows it to run in $O(n \lg n)$ time in the worst case uses the deterministic PARTITION algorithm that was modified to take an element to partition around as an input parameter.

SELECT takes an array A, the bounds p and r of the subarray in A, and the rank i of an order statistic, and in time linear in the size of the subarray $A[p\,..\,r]$ it returns the ith smallest element in $A[p\,..\,r]$.

```
BEST-CASE-QUICKSORT(A, p, r)
if p < r
    i = ⌊(r − p + 1)/2⌋
    x = SELECT(A, p, r, i)
    q = PARTITION(x)
    BEST-CASE-QUICKSORT(A, p, q − 1)
    BEST-CASE-QUICKSORT(A, q + 1, r)
```

For an n-element array, the largest subarray that BEST-CASE-QUICKSORT recurses on has $n/2$ elements. This situation occurs when $n = r - p + 1$ is even; then the subarray $A[q+1\,..\,r]$ has $n/2$ elements, and the subarray $A[p\,..\,q-1]$ has $n/2 - 1$ elements.

Because BEST-CASE-QUICKSORT always recurses on subarrays that are at most half the size of the original array, the recurrence for the worst-case running time is $T(n) \le 2T(n/2) + \Theta(n) = O(n \lg n)$.

Solution to Exercise 9.3-5

We assume that are given a procedure MEDIAN that takes as parameters an array A and subarray indices p and r, and returns the value of the median element of $A[p\,..\,r]$ in $O(n)$ time in the worst case.

Given MEDIAN, here is a linear-time algorithm SELECT$'$ for finding the ith smallest element in $A[p\,..\,r]$. This algorithm uses the deterministic PARTITION algorithm that was modified to take an element to partition around as an input parameter.

```
SELECT′(A, p, r, i)
if p == r
    return A[p]
x = MEDIAN(A, p, r)
q = PARTITION(x)
k = q − p + 1
if i == k
    return A[q]
elseif i < k
    return SELECT′(A, p, q − 1, i)
else return SELECT′(A, q + 1, r, i − k)
```

Because x is the median of $A[p\,..\,r]$, each of the subarrays $A[p\,..\,q-1]$ and $A[q+1\,..\,r]$ has at most half the number of elements of $A[p\,..\,r]$. The recurrence for the worst-case running time of SELECT′ is $T(n) \le T(n/2) + O(n) = O(n)$.

Solution to Problem 9-1

We assume that the numbers start out in an array.

a. Sort the numbers using merge sort or heapsort, which take $\Theta(n \lg n)$ worst-case time. (Don't use quicksort or insertion sort, which can take $\Theta(n^2)$ time.) Put the i largest elements (directly accessible in the sorted array) into the output array, taking $\Theta(i)$ time.

Total worst-case running time: $\Theta(n \lg n + i) = \Theta(n \lg n)$ (because $i \le n$).

b. Implement the priority queue as a heap. Build the heap using BUILD-HEAP, which takes $\Theta(n)$ time, then call HEAP-EXTRACT-MAX i times to get the i largest elements, in $\Theta(i \lg n)$ worst-case time, and store them in reverse order of extraction in the output array. The worst-case extraction time is $\Theta(i \lg n)$ because

- i extractions from a heap with $O(n)$ elements takes $i \cdot O(\lg n) = O(i \lg n)$ time, and
- half of the i extractions are from a heap with $\ge n/2$ elements, so those $i/2$ extractions take $(i/2)\Omega(\lg(n/2)) = \Omega(i \lg n)$ time in the worst case.

Total worst-case running time: $\Theta(n + i \lg n)$.

c. Use the SELECT algorithm of Section 9.3 to find the ith largest number in $\Theta(n)$ time. Partition around that number in $\Theta(n)$ time. Sort the i largest numbers in $\Theta(i \lg i)$ worst-case time (with merge sort or heapsort).

Total worst-case running time: $\Theta(n + i \lg i)$.

Note that method (c) is always asymptotically at least as good as the other two methods, and that method (b) is asymptotically at least as good as (a). (Comparing (c) to (b) is easy, but it is less obvious how to compare (c) and (b) to (a). (c) and (b) are asymptotically at least as good as (a) because n, $i \lg i$, and $i \lg n$ are all $O(n \lg n)$. The sum of two things that are $O(n \lg n)$ is also $O(n \lg n)$.)