CSCI-GA.1170-003/004 Fundamental Algorithms February 25, 2019

Problem Set 4

Lecturer: Yevgeniy Dodis *Due: Thursday, February 28*

Problem 4-1 (Humble Numbers) 10 points

A number $k > 1$ is called *humble* if the only prime factors of k are 3 and 5. Consider the task of, on input n, outputting the n smallest humble numbers and the following algorithm to do it:

```
HUMBLE(n):
    count = 0, prevOutput = 0
    HEAP.INSERT(3)
    HEAP.INSERT(5)
    while (count < n)
        cur = HEAP.EXTRACTMIN
        if cur ≠ prevOutput then
            output cur
            HEAP.INSERT(3*cur)
            HEAP.INSERT(5*cur)
            count = count + 1
            prevOutput = cur
```

(a) (4 points) Argue that the algorithm above (1) outputs numbers in increasing order, (2) does not output any number twice, (3) only outputs humble numbers, and (4) outputs all of the first n humble numbers.

(b) (2 points) Derive an *exact* (i.e., no O-notation) bound on the number of times HEAP.INSERT is called.

(c) (2 points) Bound *exactly* the number of times HEAP.EXTRACTMIN is called.
(**Hint**: Use (b).)

(d) (2 points) Use the answers to (b) and (c) above to argue that HUMBLE runs in $O(n \log n)$ time. Assume that arithmetic can be performed in $O(1)$ time.

Problem 4-2 (The k Smallest Elements) 12 points

We wish to implement a data structure D that maintains the k smallest elements of an array A. The data structure should allow the following procedures:

- $D \leftarrow$ INITIALIZE(A, n, k) that initializes D for a given array A of n elements.
- TRAVERSE(D), that returns the k smallest elements of A *in sorted order*.

- INSERT(D, x), that updates D when an element x is inserted in the array A.

We can implement D using one of the following data structures: (i) an unsorted array of size k; (ii) a sorted array of size k; (iii) a max-heap of size k.

(a) (4 points) For each of the choices (i)-(iii), show that the INITIALIZE procedure can be performed in time $O(n + k \log n)$.

(b) (3 points) For each of the choices (i)-(iii), compute the best running time for the TRAVERSE procedure you can think of. (In particular, tell your procedure.)

(c) (5 points) For each of the choices (i)-(iii), compute the best running time for the INSERT procedure you can think of. (In particular, tell your procedure.)

Problem 4-3 (Three-way partitioning) 8 points

Recall that quicksort selects an element as pivot, partitions an array around the pivot, and recurses on the left and on the right of the pivot. Consider an array that contains many duplicates and observe that for such an array, quicksort recurses on all duplicates of the pivot element. In this task you are to develop a new partitioning procedure that works well on arrays with many duplicates. The idea is to partition the array into elements less than the pivot, equal to the pivot and greater than the pivot.

(a) (4 points) Develop this idea into a partitioning algorithm and provide pseudocode. Make sure your algorithm is in-place (i.e., do not use more than a constant amount of extra space).

(b) (2 points) Use your partitioning algorithm to come up with a sorting algorithm. Analyze the worst-case running time of your algorithm.

(c) (2 points) Find an array on which the original quicksort runs in time $\Theta(n^2)$ but your algorithm from (b) in $\Theta(n)$.

Problem 4-4 (QuickSort vs Insertion Sort) 14 Points

We say that an array A is *c-nice* for a *constant* c if for all $1 \leq i < j \leq n$ such that $j - i \geq c$, we have that $A[i] \leq A[j]$. For example, a 1-nice array is completely sorted (in ascending order). In this problem we will sort such c-nice arrays A using INSERTIONSORT and QUICKSORT and compare the results.

(a) (4 Points) In asymptotic notation (remember that c is a constant) what is the worst-case running time of INSERTIONSORT on a c-nice array?

Consider now a run of QUICKSORT on a c-nice array (where the pivot element is chosen (deterministically) as the last element of the array).

(b) (2 points) Derive a lower bound on the rank q of the pivot.[1]

[1] Among n elements, the i^{th} smallest element has rank i.

(c) (3 points) Argue that after partitioning, the two subarrays $A[1 \ldots q-1]$ and $A[q+1 \ldots n]$ to the left and to the right of the pivot, respectively, are both c-nice.

(d) (4 points) From the lecture you already know that the running time of quicksort on sorted arrays is $\Theta(n^2)$. Let $B(n)$ denote the *best*-case running time of QUICKSORT on c-nice array with n elements. Using your results from (b) and (c), derive a recurrence for $B(n)$ and solve it.

(e) (1 point) Asymptotically, which is faster on c-nice arrays: the worst-case running time of insertion sort or the best-case running time of quicksort?