Problem Set 3

Data Structures and Algorithms, Fall 2020

Due: October 10, in class.

Problem 1

(a) Prove that in a heap containing n nodes, there are at most $\lceil n/2^{h+1} \rceil$ nodes of height h.

(b) Assume operation HEAPDEL(A, i) deletes the element at index position i from max-heap A. Give an implementation of HEAPDEL(A, i) that runs in $O(\lg n)$ time for an n-element max-heap.

*Problem 2 [Bonus Problem]

Prove that when all elements are distinct, the best-case running time of HEAPSORT is $\Omega(n \lg n)$.

Problem 3

We call an $m \times n$ matrix *magical* if: (a) for each row, elements in that row are in sorted order (i.e., non-decreasing) from left to right; and (b) for each column, elements in that column are in sorted order from top to bottom. Some entries in a magical matrix may be ∞, and we treat them as if no elements were in those entries. Therefore, a magical matrix can store at most mn elements.

(a) Draw a 4×4 magical matrix containing elements $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$.

(b) Describe an algorithm to implement EXTRACTMIN on a nonempty $m \times n$ magical matrix that runs in $O(m + n)$ time. That is, the algorithm finds the minimum element in the matrix, removes it from the matrix, and returns the value of that element. The modified matrix should still be a magical matrix. *(Hint: Think about* MAXHEAPIFY.*)* You need to prove the correctness of your algorithm, and you need to argue your algorithm indeed runs in $O(m + n)$ time.

(c) Describe an algorithm that can insert an element into a non-full $m \times n$ magical matrix in $O(m + n)$ time. You need to prove the correctness and argue the time complexity of your algorithm.

(d) Describe an algorithm to use an $n \times n$ magical matrix to sort n^2 numbers in $O(n^3)$ time. Your algorithm should not use any other sorting method as a subroutine. You need to prove the correctness and argue the time complexity of your algorithm.

(e) Describe an $O(m + n)$ time algorithm to determine whether a given number is in a given $m \times n$ magical matrix. You do *not* need to prove the correctness of your algorithm, but you need to argue the time complexity of your algorithm.

Problem 4

(a) Prove that quicksort's best-case running time is $\Omega(n \lg n)$.

(b) Recall the randomized quicksort introduced in class. When randomized quicksort runs, how many calls are made to the random number generator RANDOM in the worst case? How about in the best case? Give your answers in terms of Θ-notation. You do not need to prove your answers.

Problem 5

(a) When all input elements are equal, what is the running time of quicksort? What about randomized quicksort? Give your answers using asymptotic notations. You do not need to prove your answers.

(b) Modify the PARTITION procedure to produce a procedure PARTITION$'(A, p, r)$, which permutes the elements of $A[p, \cdots, r]$ and returns two indices q and t, where $p \leq q \leq t \leq r$, such that

- all elements of $A[q, \cdots, t]$ are equal;
- each element in $A[p, \cdots, q-1]$ is less than $A[q]$;
- each element in $A[t+1, \cdots, r]$ is greater than $A[q]$.

Also, discuss whether your PARTITION$'(A, p, r)$ procedure is stable. Notice, to get full credit, your PARTITION$'(A, p, r)$ procedure should take $\Theta(r-p)$ time and be in-place. You do not need to prove the correctness of your procedure.

(c) Redo part (a), assuming using your PARTITION$'(A, p, r)$ from part (b) as the partition procedure.

Problem 6

One way to improve randomized quicksort is to partition around a pivot that is chosen more carefully than by picking a random element from the subarray. One common approach is the median-of-3 method: choose the pivot as the median of a set of 3 elements randomly selected from the subarray. For this problem, let us assume that the elements in the input array $A[1, \cdots, n]$ are distinct and that $n \geq 3$. We denote the sorted output array by $A'[1, \cdots, n]$. Using the median-of-3 method to choose the pivot element x, define $p_i = \Pr(x = A'[i])$.

(a) Give an exact formula for p_i as a function of n and i for $i = 2, 3, \cdots, n-1$. (Note that $p_1 = p_n = 0$.)

(b) By what amount have we increased the likelihood of choosing the pivot as $A'[\lfloor (n+1)/2 \rfloor]$, the median of $A[1, \cdots, n]$, compared with the ordinary implementation? Assume that $n \to \infty$, and give the limiting ratio of these probabilities.

(c) If we define a "good" split to mean choosing the pivot as $x = A'[i]$, where $n/3 \leq i \leq 2n/3$, by what amount have we increased the likelihood of getting a good split compared with the ordinary implementation? *(Hint: Approximate the sum by an integral.)*

(d) Does the median-of-3 method reduce the (asymptotic) best-case or worst-case running time of randomized quicksort? If your answer is negative, discuss why the method could be considered as an "improvement" for randomized quicksort.

Problem 7

STOOGESORT$(A[0, \cdots, n-1])$

```
1: if (n == 2 and A[0] > A[1]) then
2:     SWAP(A[0], A[1]).
3: else if (n > 2) then
4:     m ← ⌈2n/3⌉.
5:     STOOGESORT(A[0, ··· , m − 1]).
6:     STOOGESORT(A[n − m, ··· , n − 1]).
7:     STOOGESORT(A[0, ··· , m − 1]).
```

(a) What is the running time of the algorithm? Prove your answer. *(Hint: Ignore the ceiling.)*

(b) Prove that the algorithm actually sorts its input.

(c) Is the algorithm still correct if we replace $m \leftarrow \lceil 2n/3 \rceil$ with $m \leftarrow \lfloor 2n/3 \rfloor$. Prove your answer.

(d) Prove that the number of swaps executed by the algorithm is at most $\binom{n}{2}$.

For Problem 8 and Problem 9, you only need to solve one of them. Pick the one you prefer! Nevertheless, you are welcome to submit solution for both.

Problem 8 [Randomized Algorithm]

The following algorithm finds the second smallest element in an unsorted array:

RANDOM2NDMIN($A[1, \cdots, n]$)

```
1: min1 ← ∞, min2 ← ∞.
2: for (i = 1 to n in random order) do
3:     if (A[i] < min1) then
4:         min2 ← min1.
5:         min1 ← A[i].
6:     else if (A[i] < min2) then
7:         min2 ← A[i].
8: return min2.
```

(a) How many times does the algorithm execute line 4 in the worst-case?

(b) What is the probability that line 4 is executed during the n^{th} iteration of the loop?

(c) What is the exact expected number of executions of line 4?

(d) How many times does the algorithm execute line 7 in the worst-case?

(e) What is the probability that line 7 is executed during the n^{th} iteration of the loop?

(f) What is the exact expected number of executions of line 7?

(Hint: You may find "linearity of expectation" helpful when solving (c) and (f).)

Problem 9 [Divide-and-Conquer Algorithm]

In this problem we will develop a divide-and-conquer algorithm for the following geometric task.

The Closest Pair Problem:
Input: A set of n points in the plane, $\{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \cdots, p_n = (x_n, y_n)\}$.
Output: The closest pair of points. That is, the pair $p_i \neq p_j$ that minimizes $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

For simplicity, assume that n is a power of two, and that all the x-coordinates x_i are distinct, as are the y-coordinates. Here's a high-level overview of the algorithm:

- Find a value x for which exactly half the points have $x_i < x$, and half have $x_i > x$. On this basis, split the points into two groups, L and R.
- Recursively find the closest pair in L and in R. Say these pairs are $p_L, q_L \in L$ and $p_R, q_R \in R$, with distances d_L and d_R respectively. Let d be the smaller of these two distances.
- It remains to be seen whether there is a point in L and a point in R that are less than distance d apart from each other. To this end, discard all points with $x_i < x - d$ or $x_i > x + d$ and sort the remaining points by y-coordinate.
- Now, go through this sorted list, and for each point, compute its distance to the *seven* subsequent points in the list. Let p_M, q_M be the closest pair found in this way.
- The answer is one of the three pairs $\{p_L, q_L\}, \{p_R, q_R\}, \{p_M, q_M\}$, whichever is closest.

(a) Prove that the algorithm is correct. *(Hint: You may find the following property helpful: any square of size $d \times d$ in the plane contains at most four points of L.)*

(b) Write down the pseudocode for the algorithm and argue its running time is $O(n \lg^2 n)$.

(c) Can you improve the algorithm so that its running time is reduced to $O(n \lg n)$?