Analysis of Algorithms, Spring 2020: Homework 2

Due: Wednesday, February 12, 11:59pm

Homework contains 66 points; will be graded out of 63

Problem 1 (10 points)

Consider the following divide-and-conquer algorithm that assumes a global array A of integers:

```
MYSTERY(integer left, integer right):
    if left == right:
        return (1, 1, 1)
    else:
        m = (left + right) / 2       (integer divide)
        (part1Lrun, part1Rrun, part1maxrun) = MYSTERY(left, m)
        (part2Lrun, part2Rrun, part2maxrun) = MYSTERY(m+1, right)

        if A[m] < A[m+1]:
            maxrun = max(part1maxrun, part2maxrun, part1Rrun+part2Lrun)
            if part1maxrun == m - left + 1:
                Lrun = part1maxrun + part2Lrun
            else:
                Lrun = part1Lrun
            if part2maxrun == right - m:
                Rrun = part2maxrun + part1Rrun
            else:
                Rrun = part2Rrun
        else:
            maxrun = max(part1maxrun, part2maxrun)
            Lrun = part1Lrun
            Rrun = part2Rrun

        return (Lrun, Rrun, maxrun)
```

Before running the algorithm, we ask the user to enter n integers that we store in the array A. Then we run `MYSTERY(0,n-1)`.

a) State the recurrence for $T(n)$ that captures the running time of the algorithm as closely as possible.

b) Use either the "unrolling the recurrence" (telescoping) or mathematical induction technique to find a tight upper bound on $T(n)$.

c) What does the algorithm do? Specify what the three returned values represent.

Problem 2 (8 points)

Use the Master Theorem to provide tight asymptotic bounds for the following recurrences. In each case, also specify the values of a, b, and $f(n)$, and indicate which case of the Master Theorem applies.

(a) $T(n) = 7T(n/8) + n$

(b) $T(n) = 2T(n/4) + \sqrt{n}$

(c) $T(n) = 9T(n/3) + n^2$

(d) $T(n) = 4T(n/2) + n \log^2 n$.

Problem 3 (16 points: 12 for implementation / 4 for writeup)

Given are n points $\{p_1, p_2, ..., p_n\}$ on the 3-dimensional integer lattice restricted to the range $[-n, n]$. In other words, each point, p_i, is of the form $p_i = (x_i, y_i, z_i)$, where x_i, y_i, z_i are all integers and $-n \leq x_i, y_i, z_i \leq n$.

Design an $O(n)$ algorithm to determine if there exists any sphere centered at the origin $(0, 0, 0)$ having two or more of the n points on its surface.

(Every point lies on the surface of one sphere centered at the origin. This question just asks you to determine if two points happen to lie on the surface of the *same* sphere.)

As a reminder, dictionaries/HashMaps are never the solution I'm looking for. There will always be an alternative solution that does not rely on the expected $O(1)$ performance of operations involving a hash table. You will not receive any credit for a solution utilizing dictionaries/HashMaps.

Problem 4 (16 points: 12 for implementation / 4 for writeup)

As chef at the soup kitchen, you do everything you can to make sure no food goes to waste. Perishable food items are donated at arbitrary times. Each comes with a shelf life before it goes bad. Each day you prepare a meal using one donated perishable food item. Design an $O(n \log n)$ algorithm to determine if it will be possible to use all the donated food without any going to waste.

Problem 5 (16 points: 12 for implementation / 4 for writeup)

It's time for Algoville's yearly parade. The participants have all arrived but they have arranged themselves out of the desired order of appearance in the parade. You help out by orchestrating the swapping of neighboring pairs of participants repeatedly until order is achieved. It's hot out, however, and each participant has limited patience. That patience is eroded each time they are forced to swap places with a neighbor. How many participants are going to lose their cool before the parade even starts?

Design an $O(n \log n)$ algorithm that computes the number of parade participants whose patience is exhausted before the parade can even start.