Fall 2018, CMPSC 465
Homework #3 solutions

1

Suppose we are given a sequence S of n elements, each of which is colored red or blue. Assuming S is represented as an array, give an in-place $\mathrm{O}(n)$ method for ordering S so that all the red elements are listed before all the blue elements.
Solution:
Maintain and update two indices i and j to perform the red element and blue element swaps. i is incremented starting from 1 (array start) and j is decremented starting from n (end of array). i is updated so that all elements to the left of $S[i]$ are red. j is decremented so that all elements to the right of $S[j]$ are blue. When i and j cross, we are done. We have four possibilities when comparing $S[i]$ and $S[j]$:

1. $S[i]$ is red and $S[j]$ is red. We increment i until we reach a blue element. We then swap the elements at $S[i]$ and $S[j]$ so that $S[i]$ is again red. We then increment i and decrement j.
2. (The approach for the three remaining cases can also be similarly filled in.)
3.
4.

The algorithm is linear time because we update both i and j for every comparison and do not backtrack.

2

Consider the following variant of Mergesort, where instead of partitioning the array into two equal halves, you split the array into 5 nearly-equal parts. What is the running time recurrence relation for this algorithm? How does the running time compare asymptotically to the running time of normal Mergesort? Clearly state any assumptions you make when deriving the closed form expression for the running time.
Solution:
The running time recurrence is $T(n) = 5T(n/5) + \mathrm{O}(n)$. Using the Master theorem, we have that $T(n) = \mathrm{O}(n \log n)$, which matches the bounds of the regular two-way merge sort. The main assumption is that the five sorted arrays of size $n/5$ can be sorted in $\mathrm{O}(n)$ time.

3

Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements. Give an efficient solution to this problem using divide-and-conquer. What is the recurrence relation for the running time? Solve the recurrence to determine a closed-form expression for the running time.
Solution:

Assume that k is a power of 2. Let us divide the array into two sets, with $k/2$ elements each. Recursively merge the arrays within the two sets until we are just left with one sorted array. We can write down a recurrence for running time in terms of k:
$T(k) = 2T(k/2) + O(kn)$.
Solving this recurrence gives us that $T(k) = O(nk \log k)$.

4

Professor Caesar wishes to develop a matrix-multiplication algorithm that is asymptotically faster than Strassen's algorithm. His algorithm will use the divide-and-conquer method, dividing each matrix into pieces of size $n/4 \times n/4$, and the divide and combine steps together will take $\Theta(n^2)$ time. He needs to determine how many subproblems his algorithm has to create in order to beat Strassen's algorithm. If his algorithm creates a subproblems, then the recurrence for the running time $T(n)$ becomes $T(n) = aT(n/4) + \Theta(n^2)$. What is the largest integer value of a for which Professor Caesar's algorithm would be asymptotically faster than Strassen's algorithm?
Solution:
By comparing the running time recurrences for Caeser's and Strassen's algorithm, we have that Caeser's algorithm is asymptotically faster than Strassen's algorithm when $\log_4 a < \log_2 7$, or $a < 49$.

5

A complex number $a + bi$, where $i = \sqrt{-1}$, can be represented by the pair (a, b). Describe a method performing only three real-number multiplications to compute the pair (e, f) representing the product of $a + bi$ and $c + di$.
Solution:
$e = ac - bd$ and $f = ad + bc$, and so we require four real-number multiplications and two additions/subtractions.
Instead, if we compute f as $f = (a + b)(c + d) - (ac + bd)$, then we require just three real-number multiplications instead of four.

6

Suppose you are consulting for a bank that is concerned about fraud detection, and they come to you with the following problem. They have a collection of n bank cards that they have confiscated, suspecting them of being used in fraud. Each bank card is a small plastic object, containing a magnetic stripe with some encrypted data, and it corresponds to a unique account in the bank. Each account can have many bank cards corresponding to it, and we will say that two bank cards are equivalent if they correspond to the same account.

It is very difficult to read the account number off a bank card directly, but the bank has a high-tech "equivalence tester" that takes two bank cards and, after performing some computations, determines whether they are equivalent.

Their question is the following: among the collection of n cards, is there a set of more than $n/2$ of them that are all equivalent to one another? Assume that the only feasible operations you can

do with the cards are to pick two of them and plug them in to the equivalence tester. Show how to decide the answer to their question with only $O(n \log n)$ invocations of the equivalence tester.
Solution:
Split the collection of cards S into S_1 and S_2, two sets of $n/2$ cards each. If S has a card v occurring more than $n/2$ times, then v must also be a majority element of S_1 or S_2 or both. We recursively compute the majority elements of S_1 and S_2, and then check if one of them is a majority element of S. The equality checks take $O(n)$ time. The running time recurrence is thus $T(n) = 2T(n/2) + \mathrm{O}(n)$, which gives us that $T(n) = \mathrm{O}(n \log n)$.

7

Suppose you are given a set of small boxes, numbered 1 to n, identical in every respect except that each of the first i contain a pearl whereas the remaining $n - i$ are empty. You also have two magic wands that can each test whether a box is empty or not in a single touch, except that a wand disappears if you test it on an empty box. Show that, without knowing the value of i, you can use the two wands to determine all the boxes containing pearls using at most $\mathrm{o}(n)$ wand touches. Express, as a function of n, the asymptotic number of wand touches needed.
Solution:
Let me first describe two approaches that require $\Theta(n)$ wand touches in the worst case, and so they are *not* valid solutions to this problem.

- Start inspecting boxes sequentially from 1 to n. We require $i + 1$ touches in this case and will only use 1 wand. The worst case happens when $i = n - 1$, leading to n wand touches.
- Try a binary search-like approach. Inspect box $n/2$ first. If it is not empty, then we know $i > n/2$ and that the first non-empty box belongs to $[n/2+1, n]$. We have a new problem of half the size, and we can continue this process by inspecting box $3n/4$, and so on. If box $n/2$ is empty, then we lose one wand. Use the other wand to sequentially inspect boxes 1 through $n/2$. The worst case wand touches is $n/2$, which is not $\mathrm{o}(n)$.

The problem statement should lead us to the solution strategy. We want $\mathrm{o}(n)$ wand touches, so $\Theta(n^{0.8})$ is a valid solution, and so is $\Theta(n^{0.7})$. How do we get such arbitrary exponents on n? One way to think about these boxes 1 through n, is as p sequences of size n/p each. Now the idea is to inspect boxes p, $2p$, $3p$, and so on sequentially. If we hit an empty box (say, box kp), we lose the first wand and then use the second wand to sequentially inspect the boxes $(k-1)p + 1$ through $kp - 1$. The total number of wand touches is thus at most $p + \frac{n}{p}$. If $p = n^{0.9}$, we get the wand touches to be $\Theta(n^{0.9}) = \mathrm{o}(n)$. $p = n^{0.5}$ gives us $\Theta(n^{0.5})$ wand touches.