CS325: Analysis of Algorithms, Fall 2020 – Midterm

Nov 3, 2020

- This exam is 150 minutes long. It starts at 10AM. Submissions are accepted until 12:30PM.
- Submit your solutions on canvas. If there is any problem with canvas, email them to `nayyeria@oregonstate.edu` with title "CS325 Midterm, [YOUR NAME]". You can type your answers, or write them and scan (or take a readable picture). Ideally, submit a pdf file in the end, and if not possible submit a zip file.
- This is a closed book exam.
- *I don't know policy:* you may write "I don't know" *and nothing else* to answer a question and receive 25 percent of the total points for that problem whereas a *completely* wrong answer will receive zero.
- There are 6 problems in this exam.
- These formula may be useful:
  - $$1 + c + c^2 + \ldots + c^n = \sum_{i=1}^{n} c^i = \begin{cases} \Theta(1), & \text{if } c < 1 \\ \Theta(n), & \text{if } c = 1 \\ \Theta(c^n), & \text{if } c > 1 \end{cases}$$
  - $$1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2} = \Theta(n^2)$$

Problem 1. **[6 pts]** Mark True or False for each of the following statements.

(a) If $f(n) = 3n^2 - 21$ then $f(n) = \Omega(n)$.

(b) If $f(n) = 34n^{10} + n + 1$ then $f(n) = \Theta(n^{10})$.

(c) If $f(n) = 2f(n-1)$, and $f(1) = 1$ then $f(n) = O(n^2)$.

(d) If $f(n) = 2n \log n + 12n$ then $f(n) = \Omega(1)$.

(e) If $f(n) = 2^{2+n}$ then $f(n) = O(2^n)$.

(f) If $f(n) = \sqrt{n} + \log n$ then $f(n) = O(n)$.

Problem 2. **[5 pts]** Bob thinks it is possible to obtain a faster sorting algorithm by breaking the array into four pieces (instead of two). He came up with this idea after he found a linear time algorithm to merge four sorted arrays. He implemented his algorithm and called the procedure MERGE$(A[1 \ldots n], j, k, \ell)$. Assuming $A[1 \ldots j]$, $A[j+1, \ldots, k]$, $A[k+1, \ldots, \ell]$ and $A[\ell+1, \ldots, n]$ are sorted, MERGE merges them into one sorted array in $O(n)$ time. Based on this procedure, Bob has designed his recursive algorithm that follows.

```
1: procedure SORT(A[1 .. n])
2:     if n > 1 then
3:         j ← ⌊n/4⌋
4:         k ← ⌊2n/4⌋
5:         ℓ ← ⌊3n/4⌋
6:         SORT(A[1, ..., j)
7:         SORT(A[j + 1, ..., k)
8:         SORT(A[k + 1, ..., ℓ)
9:         SORT(A[ℓ + 1, ..., n)
10:        MERGE(A[1 ... n], j, k, ℓ)
```

What is the running time of this sorting algorithm? Write the recursion for the running time, and use the recursion tree method to solve it.

Problem 3. **[4 pts]** Here is a divide and conquer algorithm for computing the maximum number in an array, which is *not* necessarily sorted.

```
1: procedure RECMAX(A[1 .. n])
2:     if n > 1 then
3:         m ← ⌊n/2⌋
4:         ℓ1 ←RECMAX(A[1 .. m])
5:         ℓ2 ←RECMAX(A[m + 1 .. n])
6:         RETURN max(ℓ1, ℓ2)
7:     else
8:         RETURN A[1]
```

Do you think this maximum finder is faster than the regular iterative one? What is its running time? Write the recursion for the running time, and use the recursion tree method to solve it.

Problem 4. **[5 pts]** Alice has two sorted arrays $A[1, \ldots, n]$, $B[1, \ldots, n-1]$. She knows that A is composed of distinct positive numbers, and B is derived from A by deleting one element. She would like to know the index of the deleted element of A. Unfortunately, she does not have enough time to search the arrays, so she is looking for a faster algorithm. She wonders if you can design and analyze a fast algorithm for her to find the index of the deleted elements. (Also, she has told me not to give many points for algorithms with running time much larger than $O(\log n)$).

She has provided the following example to ensure that the problem statement is clear.

$$A : 1, 3, 4, 6, 7, 8, 9, 20$$
$$B : 1, 3, 6, 7, 8, 9, 20.$$

Your algorithm should return 3 in this case, which is the index of the 4 that is deleted from A.

Problem 5. **[6 pts]** Ternary strings are strings that are composed of 0, 1 and 2. Let $P(n)$ be the number of ternary strings of length n that do not have two consecutive zeros (i.e. they do not

have "00" as a substring). For example:

$$P(1) = 3 : 0, 1, 2$$
$$P(2) = 8 : 01, 02, 10, 11, 12, 20, 21, 22$$

(a) Design a recursive algorithm to compute $P(n)$. Justify the correctness of your algorithm.

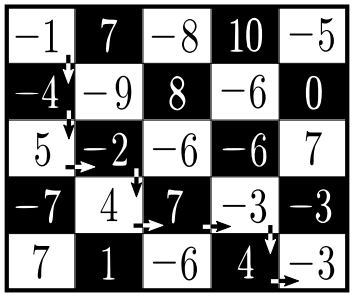
(b) Turn the recursive algorithm of part (a) into a dynamic programming.

(c) What is the running time of your dynamic programming?

Problem 6. **[6 pts]** Vankin is facing a more challenging problem recently, playing a game on the chessboard. This game is played on an $n \times n$ chessboard, whose top left square is white. The player starts the game by placing a token on the top left square. Then on each turn, the player moves the token either one square to the right or one square down. The game ends when the token is on the bottom right corner.

The player starts with a score of zero; whenever the token lands on a ***white*** square, the player adds its value to his/her score, and whenever the token lands on a ***black*** square, the player subtracts its value from his/her score. The goal for the player is to maximize the final score.

For example, given the 5×5 chessboard below, the player can score $(-1) - (-4) + (5) - (-2) + (4) - (7) + (-3) - (4) + (-3) = -3$ moving down, down, right, down, right, right, down, right. Note that this is not necessarily the best score; it is just an example.



(a) Describe an efficient algorithm to compute the maximum possible score for a game of White Squares, given the $n \times n$ array of values as input.

(b) What is the running time of your algorithm?