CMPS 101, Spring 2016: Midterm

Merrick Swaffar

6 March 2016

Q1 (2 points): Compare the functions $n \log_2 n$, n, and $n^{1.1}$ in $O(\cdot)$ notation. Which of the following is true?

- (A) $n^{1.1} = O(n \log_2 n)$, $n = O(n^{1.1})$, but $n^{1.1}$ is not $\Omega(n)$.
- (B) $n = O(n \log_2 n)$, $n \log_2 n = O(n^{1.1})$, but $n \log_2 n$ is not $\Omega(n^{1.1})$.
- (C) $n = O(n \log_2 n)$ and $n \log_2 n = \Theta(n^{1.1})$.

Answer: The answer is (B), based on the limit definitions of O, Θ, and Ω. $n = O(n \log_2 n)$ because $\lim_{n\to\infty} n/(n \log_2 n) = \lim_{n\to\infty} 1/(\log_2 n)$ which converges to 0. $n \log_2 n = O(n^{1.1})$ because $\lim_{n\to\infty}(n \log_2 n)/n^{1.1} = \lim_{n\to\infty}(\log_2 n)/n^{0.1}$ which also converges to 0 (using L'Hopital's rule). Because the previous limit converges to 0, $n \log_2 n$ can not be $\Omega(n^{1.1})$.
The answer can not be (A), because $\lim_{n\to\infty} n^{1.1}/(n \log_2 n)$ goes to infinity (L'Hopital's rule), implying that $n^{1.1} \neq O(n \log_2 n)$.
The answer can not be (C), because based on the calculations from (B), $n \log_2 n \neq \Omega(n^{1.1})$, which means that $n \log_2 n$ is also not $\Theta(n^{1.1})$.

Q5 (4 points): You're a financial analyst, and have another stock market problem to solve. The input is A, an array of stock prices. For day i, the best trade is the maximum profit that can be achieved by buying at day i and selling on a subsequent day. For convenience, you can define the best trade for the last day to simply be $-A[n]$ (because if you buy on the last day, you cannot sell and you lost all your money).

Give the pseudocode of an algorithm that returns an array containing the maximum profit for every day in A. Prove that your algorithm is correct, and do a complexity analysis. I'm looking for an $O(n \log_2 n)$ algorithm using divide and conquer. In class, we solve the problem of finding the best trade over all possible buying days. You might want to revisit that algorithm, and modify it to get the algorithm for this problem.

There is a completely different $O(n)$ algorithm for this problem. If you can find that, I'll give you two bonus extra credit points. (If you didn't get this in the exam, you still have to give the $O(n \log_2 n)$ algorithm.)

Solution:

Algorithm:

```
profit(A)
    n = A.length
    P = []
    P[n] = -A[n]
    if n < 2:
        return P
    best = 0
    for i = n to 2
        best = max(best, A[i])
        profit = best - A[i-1]
        P[i-1] = profit
    return P
```

Time Complexity:

This algorithm runs in $\theta(n)$ time. It loops through the array only once, performing only constant time operations as it goes.

Proof:

Base Case: Length of A is equal to 1. The maximum profit for A[i] is -A[i], since there are no more elements in the array.

Loop Invariant: After i iterations of the loop, 'best' contains the largest element from A[i] to A[n] and P contains the maximum profit for elements from A[i-1] to A[n].

Induction: 'best' contains the largest element from A[i+1] to A[n]. The algorithm compares the ith element in the array with best and puts the max back into best. best now contains the largest element from A[i] to A[n]. P contains the max profit for elements from A[i] to A[n]. The algorithm subtracts A[i-1] from the max from A[i] to A[n] and puts it into P. P now contains the maximum profit for elements from A[i-1] to A[n]. The loop terminates when i reaches 1 and the result is returned.