CS3230: Design and Analysis of Algorithms
Semester 2, 2021-22, School of Computing, NUS

Practice Problems: Recurrences & Divide-and-Conquer

February 17, 2022

Instructions

- This problem set is **completely optional**. There is no need to submit the solutions.
- Solutions will be available later. You are strongly encouraged to first try to solve the problems by yourselves and then check the solutions.
- Post on the LumiNUS forums if you will face any problem while solving the questions.

Question 1: For each of these, try figuring out why you can't use master theorem to solve these recurrences, you do not need to be formal. I also invite you to get as tight bound as possible, for both upper and lower.

1. $T(n) = 2T(n-1) + \Theta(1)$
2. $T(n) = T(n-1) + T(n-2) + \Theta(1)$
3. $T(n) = T(\sqrt{n}) + \Theta(1)$
4. $T(n) = 2T(\sqrt{n}) + \Theta(1)$
5. $T(n,m) = T(\frac{n}{2}, m) + \Theta(m)$
6. $T(n) = (\sqrt{n}+1)T(\sqrt{n}) + \sqrt{n}$

Question 2: Notice that in case 1 of master theorem for example, the condition states: $f(n) \in O(n^{\log_b(a)-\epsilon})$ for some $\epsilon > 0$. The point of this question is to show that this way of framing the condition is crucial.

Prove that $f(n) \in O(n^{\log_b(a)-\epsilon}) \implies f(n) \in o(n^{\log_b(a)})$.

Prove that there exists functions $f(n)$ such that $f(n) \in o(n^{\log_b(a)})$ but $f(n) \notin O(n^{\log_b(a)-\epsilon})$.

Question 3: Consider the function Fun(x, y). What is its output? What is the invariant for the while loop? Can you show that this algorithm correctly compute the output?

`Fun`(x, y):

1. $ans = 0, p = x, q = y$
2. While $q > 0$ do
   (a) $r \leftarrow q \mod 2$
   (b) $q \leftarrow q/2$
   (c) $ans \leftarrow ans + r \times p$
   (d) $p \leftarrow p \times 2$
3. **return** ans

Question 4: Given an array A of integers, a pair (i, j) is said to be an *inversion* if $i < j$ and $A[i] > A[j]$. Design an $O(n \log n)$ time algorithm that counts the number of inversions in a given array of size n.

Question 5: Given an array A of n integers suppose we know that there exists an integer that appears more than $n/2$ times in A. Design a divide-and-conquer algorithm to find that element in $O(n \log n)$ time. You are no allowed to sort the array A.

Question 6: Can you design an $O(n)$ time algorithm for the problem stated in Question 5? (Note, this is not a divide-and-conquer algorithm.)

Question 7: Given an array A of n integers (possibly 0 or negative as well), find the largest possible value c that can be obtained by summing up the values in some contiguous subarray of A, i.e., $c = A[i] + A[i+1] + A[i+2] + \cdots + A[i+t]$ for some i, t. Think of a divide and conquer solution that does it in $O(n \log n)$ time. As a bonus, you can then think about how to improve it to $O(n)$ by some minor modifications.

Question 8: Consider an array of distinct integers sorted in increasing order. The array has then been rotated (anti-clockwise) k number of times, i.e., all the numbers in the sorted array have been (cyclically) shifted k places on the leftside. Now given such an array, find the value of k.

Question 9: Consider the problem of finding a *peak* in a 2D-array of size $m \times n$, as described in Tutorial 4. In the tutorial we have seen an algorithm with running time $O(m \log n)$. Can you modify that algorithm to achieve running time $O(m + n)$? (**Hint:** In the tutorial we reduced the problem of size $m \times n$ to that of size $m \times n/2$. Now try to come up with some argument so that you can reduce the problem of size $m \times n$ to that of size $m/2 \times n/2$.)