Problem Set 3

Data Structures and Algorithms, Fall 2021

Due: September 23 23:59:59 (UTC+8), mail to `dsalg21ps@chaodong.me`.

Problem 1

(a) Show the solution of the recurrence $T(n) = 2T(n/2) + n$ is $\Omega(n \lg n)$. That is, show $T(n) \in \Omega(n \lg n)$. You need to prove your answer.

(b) Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = T(n - 2) + T(n/2) + n$. Use the substitution method to prove your answer is correct.

Problem 2

(a) Give an asymptotic *tight* bound for the recurrence $T(n) = T(\alpha) + T(n - \alpha) + cn$, where $\alpha \in \mathbb{N}^+$ is a constant and $c > 0$ is also a constant. You do *not* need to prove your answer.

(b) Give an asymptotic *tight* bound for the recurrence $T(n) = T(\alpha n) + T((1 - \alpha)n) + cn$, where $\alpha \in (0, 1)$ is a constant and $c > 0$ is also a constant. You do *not* need to prove your answer.

Problem 3

Devise an algorithm that squares any n-digit binary number in $O(n^{\lg 3})$ time. First give an overview of your algorithm, then provide pseudocode, and finally analyze its running time.

Problem 4

You are given an infinite array $A[\cdot]$ in which the first n cells contain integers in sorted order and the rest of the cells are filled with ∞. You are *not* given the value of n. Describe an algorithm that takes an integer x as input and finds a position in the array containing x, if such a position exists, in $O(\lg n)$ time. (If you are disturbed by the fact that the array A has infinite length, assume instead that it is of length n, but that you don't know this length, and that the implementation of the array data type in your programming language returns the error message ∞ whenever elements $A[i]$ with $i > n$ are accessed.)

Problem 5

You are a visitor at a political convention with n delegates; each delegate is a member of exactly one political party. It is impossible to tell which political party any delegate belongs to; in particular, you will be summarily ejected from the convention if you ask. However, you can determine whether any pair of delegates belong to the same party by introducing them to each other (i.e., a "pairwise meeting"). Members of the same political party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.

Suppose more than half of the delegates belong to the same political party. Describe an efficient algorithm that identifies all members of this majority party. You need to argue the correctness of your algorithm, and analyze its runtime (in terms of the number of "pairwise meetings").

Problem 6

Recall the FINDMAXIMUMSUBARRAY algorithm introduced in Section 4.1 in the textbook CLRS. Modify it to obtain an $O(n)$ time divide-and-conquer algorithm for the maximum subarray problem. Your algorithm should not leverage the dynamic programming technique.[1] In particular, your algorithm should not look like an answer for Exercise 4.1-5 in CLRS. *(Hint: (a) The recurrence for the runtime of the modified algorithm should be $T(n) = 2T(n/2) + O(1)$. (b) In the "conquer" step, instead of just computing the maximum subarray in $A[low, \cdots, high]$, compute and return some more information so that the "combine" step is faster. In particular, finding "maximum crossing subarray" should be fast in the combine step.)*

Problem 7

(a) Show how to find the second largest element in a binary max-heap in constant time.

(b) Show how to find the k^{th} largest element in a binary max-heap in $O(k \lg k)$ time. First give an overview of your algorithm, then provide pseudocode, and finally analyze its running time. *(Hint: Using another heap might help.)*

[1] If you do not know what is "dynamic programming", just ignore this sentence.