

Problem Set 2

Data Structures and Algorithms, Fall 2020

Due: September 24, in class.

Problem 1

Prove that the solution of recurrence $T(n) = 2T(\lfloor n/2 \rfloor + 1) + n$ is in $O(n \lg n)$.

Problem 2

(a) Use the recursion tree method to find a good asymptotic upper bound for the recurrence $T(n) = T(n-2) + T(n/4) + n$. Then, use the substitution method 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

Let A be an array containing n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an *inversion* of A .

(a) List all inversions of the array $\langle 2, 3, 8, 6, 1 \rangle$.

(b) What is the relationship between the running time of insertion sort and the number of inversions in the input array? To get full credit, prove your answer is correct.

(c) Give an $O(n \lg n)$ time algorithm that can count the number of inversions of a size n array. Your algorithm does not need to list all inversions. You do *not* need to prove your algorithm is correct. (*Hint: Modify the merge sort algorithm.*)

Problem 4

In this class, we often assume that parameter passing during procedure calls takes constant time, even if an N -element array is being passed. This assumption is valid in many systems because a pointer to the array is passed, not the array itself. This problem examines the implications of three different parameter-passing strategies:

1. An array is passed by pointer. Time = $\Theta(1)$.
2. An array is passed by copying. Time = $\Theta(N)$, where N is the size of the array.
3. An array is passed by copying only the subrange that might be accessed by the called procedure. Time = $\Theta(q - p + 1)$ if the subarray $A[p, \dots, q]$ is passed.

For each of the three methods above, give the recurrence for the worst-case running time of merge sort when arrays are passed using that method, and give a good upper bound on the solution of the recurrence. Use N to denote the size of the original problem and n to denote the size of a subproblem.

Problem 5

Denote the running time of the *fastest* algorithm to square an n -bit integer as $T_1(n)$, and denote the running time of the *fastest* algorithm to multiply two n -bit integers as $T_2(n)$. Professor F. Lake claims that $T_1(n) = o(T_2(n))$, i.e., it is asymptotically faster to square an n -bit integer than to multiply two n -bit integers. Is this claim true or false? Prove your answer.

Problem 6

Recall the FINDMAXIMUMSUBARRAY algorithm introduced in Section 4.1 in the textbook CLRS.¹ Modify it to obtain an $O(n)$ time divide-and-conquer algorithm for the maximum subarray problem. Your algorithm should not leverage dynamic programming.² 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 $A[\text{low}, \dots, \text{high}]$, 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

You are having a party with n other friends, each of which plays either as a citizen or a werewolf. You do not know who are citizens or who are werewolves, but all your friends do. There are always more citizens than there are werewolves, but n can be an arbitrary positive integer.

Your allowed “query” operation is as follows: You pick two people and ask each of them if the other person is a citizen or a werewolf. When you do this, a citizen must tell the truth about the identity of the other person, whereas a werewolf does not have to. (That is, a werewolf may lie about the identity of the other person.) Your algorithm should be correct regardless of the behavior of the werewolves.

(a) Devise an algorithm that, given a player as input, can determine whether the player is a citizen using $O(n)$ queries. Prove the correctness of your algorithm carefully.

(b) Devise a divide-and-conquer algorithm that finds a citizen using $O(n \lg n)$ queries. Prove the correctness of your algorithm carefully. Do not devise a linear time algorithm here, as we would like you to practice with divide-and-conquer. (*Hint: Split a group of people into two groups, think about what invariant must hold for at least one of these two groups. Also, you may find solution for part (a) useful.*)

(c) **[Bonus Question]**³ Devise an algorithm that finds a citizen using $O(n)$ queries. Prove the correctness of your algorithm carefully. (*Hint: Don’t be afraid to sometimes “throw away” a pair of people once you have asked them to identify each other.*)

¹Recall CLRS refers to “Introduction to Algorithms (3ed)”.

²If you do not know what is “dynamic programming”, just ignore this sentence.

³You are *not* required to solve bonus questions. But you get extra credit if you do!