CSE 102: Spring 2021

Homework Assignment # 3

Divide and Conquer

1. (2 points)

   Consider the problem of multiplying two large n-bit integers in a machine where the word size is one bit. The first three sub-problems will be discussed in the lecture.

   (a) (0 point) Describe the straightforward algorithm that takes n^2 bit-multiplications.

   (b) (0 point) Find a way to compute the product of the two numbers using three multiplications of $n/2$ bit numbers (you will also have to do some shifts, additions, and subtractions, and ignore the possibility of carries increasing the length of intermediate results). Describe your answer as a divide and conquer algorithm.

   (c) (0 point) Assume that adding/subtracting numbers takes time proportional to the number of bits involved, and shifting takes constant time. Derive a recurrence relation for the running time of your divide and conquer algorithm. Use the master theorem to get an asymptotic solution to the recurrence.

   (d) (2 points) Now assume that we can find the product of two n-bit numbers using *some* number of multiplications of $n/3$-bit numbers (plus some additions, subtractions, and shifts). What is the largest number of $n/3$ bit number multiplications that leads to an asymptotically faster algorithms than the $n/2$ divide and conquer algorithm above?

2. (4 points) You are given two *sorted* arrays of same size n. Design a divide-and-conquer $O(\log n)$ algorithm to compute the median of the these two sorted array. Provide pseudocode *that you understand* [do not copy and paste solutions found somewhere on the internet] including

the base case of two arrays each of size 1 or size 2. Clearly state the size of subproblems, number of subproblems, and the order of the work required to combine the subproblems. Thus, establish the asymptotic computational complexity of the algorithm.

3. (3 points) Assume you have an array $A[1..n]$ of n elements. A *majority element* of A is any element occurring in more than $n/2$ positions (so if $n = 6$ or $n = 7$, any majority element will occur in at least 4 positions). Assume that elements *cannot* be ordered or sorted, but can be compared for equality. (You might think of the elements as chips, and there is a testor that can be used to determine whether or not two chips are identical.)

   Design a $O(n \lg n)$ divide and conquer algorithm to find a majority element in A (or determine that no majority element exists). Establish the computational complexity.

   Remark: There is also an $O(n)$ algorithm for this problem, but that problem is harder.

4. (3 points) A round-robin tournament is a collection of games in which each team plays each other exactly once. A schedule can be represented as an array of triplets, where the triplet (d, i, j) means that team i will play teal j on day d. A schedule is *reasonable* if no team plays more than one game per day, and *optimal* if it uses the smallest number of days out of all reasonable schedules.

   (a) Design an *optimal* schedule for the tournament using divide-and-conquer assuming that n is a power of 2, where n is the number of teams. Provide a brief argument that the algorithm is optimal.

   (b) Establish the computational complexity of the algorithm.

5. (5 points) Quicksort Algorithm: Read the description of Quicksort Algorithm [CLRS Section 7.1, pages 184].

   (a) (1 point) Problem 7.1.1, CLRS Page 173: Illustrate the operation of PARTITION on the array $A = 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11$.

(b) (1 point) Problem 7.1.3, CLRS Page 174: Give a brief argument that the running time of PARTITION algorithm on a subarray of size n is $\theta(n)$.

(c) (1 point) Worst-case Partitioning: Establish that the worst-case running time algorithm for quicksort is $\theta(n^2)$.

(d) (1 point) Best-case Partitioning: establish thatbthe best-case running time algorithm for quicksort is

(e) (1 point) Proportional Partitioning: Assume that the split at each level of recursion in quicksort is a constant ratio of $r : 1$, where r is a constant. Establish that the running time of quicksort is still $\theta(n \ln n)$.