CSE 222 (ADA) Homework Assignment 1 (Theory)

Deadline : Feb 11 (Friday) 1:15 PM.

The theory assignment has to be done in a team of at most two members, as already selected by you. The solutions are to be typed either as a word document or latex-ed and uploaded as pdf on GC. We shall strictly not accept solutions written in any other form. Remember that both team members need to upload the HW solution on GC. Collaboration across teams or seeking help from any sources other than the lectures, notes and texts mentioned on the homepage will be considered an act of plagiarism

There are 3 problems in this homework.

Problem 1.

(a) (10 points) Describe an algorithm that sorts an input array $A[1 \cdots n]$ by calling a subroutine SqrtSort(k), which sorts the subarray $A[k+1 \cdots k+\sqrt{n}]$ in place, given an arbitrary integer k between 0 and $n - \sqrt{n}$ as input. (To simplify the problem, assume that $\sqrt{n}$ is an integer.) Your algorithm is ***only*** allowed to inspect or modify the input array by calling SqrtSort; in particular, your algorithm must not directly compare, move or copy array elements. How many times does your algorithm call SqrtSort in the worst case? Give pseudocode. Remember that you cannot use anything other than calling the SqrtSort routine and maybe some loops. No formal proof of correctness is mandatory. But write a few sentences justifying your approach.

(b) (10 points) Prove that your algorithm from part (a) is optimal up to constant factors. In other words, if $f(n)$ is the number of times your algorithm calls SqrtSort, prove that no algorithm can sort using $o(f(n))$ calls to SqrtSort. Note that here we are assuming that these algorithms **cannot do** anything other than calling SqrtSort repeatedly.

(Hint: Think of a typical worst case example for any sorting algorithm)

(c) (10 points) Now suppose SqrtSort is implemented recursively, by calling your sorting algorithm from part (a). For example, at the second level of recursion, the algorithm is sorting arrays roughly of size $n^{1/4}$. What is the worst-case running time of the resulting sorting algorithm? (To simplify the analysis, assume that the array size n has the form 2^{2^k}), so that repeated square roots are always integers.)

Problem 2. (10 points)

Let $n = 2^l - 1$ **1** for some positive integer l. Suppose someone claims to hold an unsorted array $A[1 \cdots n]$ of *distinct* l-bit strings; thus, exactly one l-bit string does *not* appear in A. Suppose further that the ***only*** way we can access A is by calling the function FetchBit(i, j), which returns the j^{th} bit of the string $A[i]$ in $O(1)$ time. Describe an algorithm to find the missing string in

A using only $O(n)$ calls to FETCHBIT. Again, give either pseucode or write a generic description. Demonstrating the algorithm on a specific example will not fetch any marks.

Problem 3. (10 points)

Use recursion tree to solve the following recurrence.

$$T(n) = T(n/15) + T(n/10) + 2T(n/6) + \sqrt{n}$$