NAME: _____

CSE 421
Introduction to Algorithms
Midterm Exam Spring 2023

Anup Rao May 3, 2023

DIRECTIONS:

- Answer the problems on the exam paper.
- Justify all answers with proofs, unless the facts you need have been proved in class or in the book.
- If you need extra space use the back of a page
- You have 50 minutes to complete the exam.
- Please do not turn the exam over until you are instructed to do so.
- Good Luck!

1. (40 points, 5 each) For each of the following problems answer **True** or **False** and BRIEFLY JUSTIFY you answer.

   (a) $2^{n \log_2 n} = O(n^{2n})$.

   Solution: True. $2^{n \log_2 n} = (2^{\log_2 n})^n = n^n < n^{2n}$.

   (b) Suppose we are given a connected undirected graph with m edges, such that all edge weights are distinct, except for two edges, which have the same weight. Then this graph can have at most two MSTs.

   Solution: True, we have shown in homework and class, that every MST corresponds to a sorting of the edges according to weights. There are only two possible sorted orders.

   (c) If the running time of an algorithm satisfies the recurrence

   $$T(n) = T(0.1n) + T(0.2n) + T(0.7n) + cn,$$

   for some positive constant c, and $T(1) = 1$, then $T(n) = O(n)$.

   Solution: False. Every level of the recursion tree contributes the same, so the running time of this algorithm is proportional to $n \log n$.

   (d) The Fast-Fourier transform algorithm leads to an algorithm for multiplying $n \times n$ matrices in $O(n \log n \log \log n)$ time.

   Solution: False. The FFT does not help to multiply matrices.

(e) Suppose you are given a collection of sets $S_1, \ldots, S_n$ such that their union is equal to $\{1, 2, \ldots, n\}$, as well as non-negative numbers $w_1, \ldots, w_n$. Your goal is to find a subcollection $i_1, \ldots, i_t$ such that the union of $S_{i_1}, \ldots, S_{i_t}$ is $\{1, 2, \ldots, n\}$, and $w_{i_1} + \ldots + w_{i_t}$ is minimized. Then the greedy algorithm given in class can be modified to find a solution that is within a factor of $\log n$ from the optimal solution.

Solution: True. The greedy algorithm would pick the set that covers the most elements per weight. If the optimal solution has weight k, then in each step some set must cover elements at the rate of $1/k$. This leads to a solution of cost at most $k \log n$.

(f) There is a polynomial time algorithm for computing a vertex cover that is within a factor of 2 of optimal.

Solution: True. We gave a greedy algorithm for this in class.

(g) There is a polynomial time algorithm for the weighted interval scheduling problem.

Solution: True. We saw a dynamic programming algorithm for this problem.

(h) If $T(n) = T(n-1) + O(n)$, then $T(n)$ is at most $O(n^2)$.

Solution: True. The solution is $O(1)(1 + 2 + \ldots + n) = O(n^2)$.

2. (20 points) You are given a rooted complete binary tree on n vertices. Each vertex v is labelled by an integer value x_v. Say that a vertex is a *local minimum* if its label is less than the labels of each of its neighbors (neighbors includes the parent and the children). Assuming that all the labels are distinct, give an $O(\sqrt{n})$ time algorithm to find a local minimum in the tree.

   Solution: The algorithm is as follows:

   (a) Let v be the root.

   (b) If v is a local minimum, output v.

   (c) Otherwise, update v to be a child that has a smaller value, and go to step (b).

   Since, the algorithm moves to a child in each step, the running time is at most the depth of the tree, which is $O(\log n)$.

   To prove that the algorithm is correct, we need to show that it either finds a local minimum, or finds a child of lower value in step (c).

   In each step, we update v to be a child of lower value, so v always has a value that is less than that of its parent. So, if v is not a local minimum, it must have a child of lower value. If v is a leaf, the algorithm will output v, since v is guaranteed to have a parent of higher value. This proves that the algorithm must terminate with a local minimum.

3. (20 points) Given an array of elements $A[1, ..., n]$, give an $O(n \log n)$ time algorithm to find a majority element, namely an element that is stored in more than $n/2$ locations, if one exists. Note that the elements of the array are not necessarily integers, so you can only check whether two elements are equal or not, and not whether one is larger than the other. HINT: Observe that if x is the majority element, then it must be a majority element in either $A[1, ..., n/2]$ or $A[n/2 + 1, ..., n]$.

   Solution: Algorithm:

   (a) Recursively compute the majority element m_1 of the first half of the array, and the majority element of the second half of the array m_2.

   (b) Scan the whole array to count how many times m_1 occurs and how many times m_2 occurs. If either occurs more than $n/2$ times, output that element, otherwise output that there is no majority element.

   As in the hint, if x is a majority element, it must be returned in one of the recursive calls, so the algorithm will output x. Moreover, whenever the algorithm outputs an element it is guaranteed to be a majority element because the algorithm checks that this is the case.

   The running time satisfies

$$T(n) = 2T(n/2) + O(n),$$

   which has the solution $T(n) = O(n \log n)$.