CMPSC 465 Data Structures & Algorithms
Spring 2021 Paul Medvedev and Chunhao Wang

HW 3

Due February 11, 10:00 pm

Instructions: You may work in groups of up to three people to solve the homework. You must write your own solutions and explicitly acknowledge up everyone whom you have worked with or who has given you any significant ideas about the HW solutions. You may also use books or online resources to help solve homework problems. All consulted references must be acknowledged.

You are encouraged to solve the problem sets on your own using only the textbook and lecture notes as a reference. This will give you the best chance of doing well on the exams. Relying too much on the help of group members or on online resources will hinder your performance on the exams.

Late HWs will be accepted until 11:59pm with a 20% penalty. HWs not submitted by 11:59pm will receive 0. There will be no exceptions to this policy, as we post the solutions soon after the deadline. However, you will be able to drop the three lowest HW grades.

For the full policy on HW assignments, please consult the syllabus.

1. **(0 pts.) Acknowledgements.** The assignment will receive a 0 if this question is not answered.
   (a) If you worked in a group, list the members of the group. Otherwise, write "I did not work in a group."
   (b) If you received significant ideas about the HW solutions from anyone not in your group, list their names here. Otherwise, write "I did not consult without anyone my group members".
   (c) List any resources besides the course material that you consulted in order to solve the material. If you did not consult anything, write "I did not consult any non-class materials."

2. **(16pt pts.) Solving recurrences** Solve the following recurrence relations and give a Θ bound for each of them.

   Keep in mind that we've seen several examples of how to solve recurrences. You can try to apply the Master's theorem or use a recursion tree. You can also try to "unfold" the recurrences, as we saw with $T(n) = T(n-1) + O(n)$ in lecture. Another technique is applicable if the work term is not a polynomial: 1) take the polynomial upper bound of it, and see what Master theorem gives you, 2) take the polynomial lower bound of it, and see what Master theorem gives you, and 3) if these agree, then you have an answer.

   (a) $T(n) = 11T(n/5) + 13n^{1.3}$
   (b) $T(n) = 6T(n/2) + n^{2.8}$
   (c) $T(n) = 5T(n/3) + \log^2 n$
   (d) $T(n) = T(n-2) + \log n$

3. **(16 pts.) Sorted Array**

   Given a sorted array A of n (possibly negative) distinct integers, you want to find out whether there is an index i for which $A[i] = i$. Give a divide-and-conquer algorithm that runs in time $O(\log n)$. Provide only the main idea and the runtime analysis. You can assume that n is a power of 2 for simplicity. (Hint: Look at the value of $A[\frac{n}{2}]$ and decide how to recurse based on that)

4. (16 pts.) Linear Time Sorting

Show that any array of integers $x[1 \ldots n]$ can be sorted in $O(n+M)$ time, where

$$M = \max_i x_i - \min_i x_i$$

For small M, this is linear time: why doesn't the $\Omega(n \log n)$ lower bound apply in this case? (Hint: Think about what a teenager would do in real life if they were given a thousand cash bills (each bill being a single, five, ten, twenty, etc) and asked to put them in sorted order. I doubt they would do a merge sort.)

5. (10 pts.) Binary Search Lower Bound. You are given an array $A[1, \ldots, n]$ that is sorted. You are then given an element x and you want to find its location in A, or report that it does not exist. The binary search algorithm runs in time $O(\log n)$. Show that any comparison-based algorithm must take $\Omega(\log n)$ time. By "comparison-based", we mean that it can only access the elements of A by checking if they are above or below a given value.

Solution: As in the $\Omega(n \log n)$ lower bound for sorting on page 52 we can look at a comparison-based algorithm for search in a sorted array as a binary tree in which a path from the root to a leaf represents a run of the algorithm: at every node a comparison takes place and, according to its result, a new comparison is performed. A leaf of the tree represents an output of the algorithm, i.e. the index of the element x that we are searching or a special value indicating the element x does not appear in the array. Now, all possible indices must appear as leaves or the algorithm will fail when x is at one of the missing indices. Hence, the tree must have at least n leaves, implying its depth must be $\Omega(\log n)$, i.e. in the worst case it must perform at least $\Omega(\log n)$ comparisons.