Basic Algorithms CSCI-UA.0310

Homework 2

Due: February 15th, 4 PM EST

Instructions

Please answer each **Problem** on a separate page. Submissions must be uploaded to your account on Gradescope by the due date and time above.
Please note that no late submission will be accepted for this homework.

Note

You have to wait until the lecture on February 9th to solve some of the problems.

Problems to submit

Problem 1 (5+5+13 points)

Let $A[1, \ldots, n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called a **reverse pair** for A.

(a) List the five reverse pairs of the array $A = [2, 3, 8, 6, 1]$.

(b) What is the relationship between the running time of INSERTION_SORT and the number of reverse pairs in the input array? Justify your answer.

(c) Modify MERGE_SORT in order to give an algorithm which determines the number of reverse pairs in an array of n numbers in $\Theta(n \log n)$ run time.
Fully explain your algorithm and also justify why your algorithms satisfies the required time complexity bound.

Problem 2 (5+8+8 points)

Recall that MERGE_SORT splits the input array into two halves of almost equal sizes, and after each half is sorted recursively, both of them are merged into a sorted array.
Let's now consider a variant of MERGE_SORT, where instead of splitting the array into two equal size parts, we split it into four parts of almost equal sizes, recursively sort out each part, and at the end, merge all four parts together into a sorted final array.
In what follows, we want to find the time complexity of this variant of MERGE_SORT.

(a) Define $T(n)$ as the worst-case running time of this variant on any input of size n. Find a recursive formula for $T(n)$, i.e. write $T(n)$ in terms of $T(k)$ for some $1 \leq k < n$.

(b) Draw the corresponding recursion tree and find the explicit answer for $T(n)$.

(c) Prove your result in part (b) formally using strong induction.
For full credit, your answer must use $\Theta(.)$ notation, i.e., you should obtain both an upper bound and a lower bound.

Problem 3 (32 points)

For each of the following recursions, draw the recursion tree and find the height of the tree, the running time of each layer, and the sum of running times for all layers. Then use this information to find the explicit answer for $T(n)$.
For full credit, your answers must use $\Theta(.)$ notation, i.e., you should obtain both an upper bound and a lower bound.

(a) $T(n) = T(n-1) + n$.

(b) $T(n) = 2T(n/4) + \sqrt{n}$.
You may assume that n is a power of 4, i.e., $n = 4^k$ for some positive integer k.

(c) $T(n) = 9T(n/3) + n^2$.
You may assume that n is a power of 3, i.e., $n = 3^k$ for some positive integer k.

(d) $T(n) = T(n/2) + 1$.
You may assume that n is a power of 2, i.e., $n = 2^k$ for some positive integer k.

Problem 4 (10 points)

Recall that we showed the stability of INSERTION_SORT.
Show that MERGE_SORT is a stable sorting algorithm. For that, you should show that the stability of the input is preserved under all the steps undertaken by MERGE_SORT.

Problem 5 (14 points)

Let $A[1 \ldots n]$ be an array consisting of n distinct numbers. Develop a divide-and-conquer algorithm to find the largest number among the elements of A. Write down the pseudo-code for your algorithm.

Bonus problem

The following problems are optional. They will NOT be graded and they will NOT appear on any of the exams. Work on them only if you are interested. They will help you to develop problem solving skills for your future endeavors.
You are highly encouraged to think on them and discuss them with your peers on a separate thread on the course page on Campuswire.

Bonus Problem 1

Recall the variant of SELECTION_SORT discussed in Homework 1 Problem 5. In each iteration, we needed to find the maximum and the second maximum elements among k given elements. Note that this is feasible by using at most $2k - 3$ comparisons.

(a) Can you improve the number of comparisons used?

(b) What is the minimum number of comparisons you can use?

(c) Can you use at most $k + \log_2 k - 2$ comparisons?

(d) Show how improving the number of comparisons used affects the time complexity of the variant algorithm.