COMP 3711 Design and Analysis of Algorithms
Spring 2015
Written Assignment 1

1. For each pair of expressions (A, B) below, indicate whether A is O, Ω, or Θ of B. Note that zero, one, or more of these relations may hold for a given pair. List all correct ones. It often happens that some students will get the directions wrong, so please write out the relation in full, i.e., $A = O(B), A = \Omega(B)$, or $A = \Theta(B)$ when in doubt.

   (a) $A = n^3 - 100n, B = n^2$;

   (b) $A = \log n, B = \log_{1.1} n$;

   (c) $A = 2^{2n}, B = 2^{3n}$;

   (d) $A = 2^{\log n}, B = n$;

   (e) $A = \log\log n, B = 10^{100}$ (this number, which is 1 followed by 100 zeroes, is called "googol").

2. Give asymptotic upper bounds for $T(n)$ under the following recurrences. Make your bounds as tight as possible. A correct answer will gain full credits; otherwise, showing the steps may gain you partial credits.

   (a) $T(1) = 1; T(n) = T(n/2) + 1$ for $n > 1$.

   (b) $T(1) = 1; T(n) = 2T(n/2) + n^2$ for $n > 1$.

   (c) $T(1) = 1; T(n) = T(2n/3) + 1$ for $n > 1$.

   (d) $T(1) = 1$; $T(n) = 3T(n/2) + n$ for $n > 1$.

   (e) $T(n) = 1$ for $n \leq 2$; $T(n) = T(\sqrt{n}) + 1$ for $n > 2$.

3. Suppose you have k sorted arrays, each with n numbers, and you want to combine them into a single sorted array of kn numbers. We are going to use the MERGE procedure in mergesort for this task. Recall that given two sorted arrays of sizes x and y, MERGE combines them into one sorted array in $O(x + y)$ time.

   (a) One strategy is to first MERGE the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the running time of this algorithm (in terms of k and n)?

   (b) Give a more efficient solution to this problem, and analyze its running time. For full credits your algorithm should run in time $O(nk \log k)$.

Turn to the next page...

4. Suppose we are given an array $A[1 \ldots n]$ of distinct numbers with the special property that $A[1] > A[2]$ and $A[n-1] < A[n]$. We say that an element $A[x]$ is a *local minimum* if it is less than both its neighbors, or more formally, if $A[x-1] > A[x]$ and $A[x] < A[x+1]$. For example, there are two local minima (3 and 1) in the following array:

9	3	7	2	1	4	5

   We can obviously find a local minimum in $O(n)$ time by scanning through the array. Describe and analyze an algorithm that finds a local minimum in $O(\log n)$ time. If there is more than one local minimum, finding any of them is fine. [Hint: With the given boundary conditions, the array must have at least one local minimum.]

5. We know that given a sorted array A and a value x, we can use binary search to find x in A (if it exists) in $O(\log n)$ time. If A is unsorted, then binary search does not work, and the best deterministic algorithm would just scan the entire A to look for x, which takes n comparisons in the worst case. Below we explore if it is possible to reduce the running time by randomization.

   Consider the following randomized strategy: pick a random index i into A. If $A[i] = x$, then we terminate; otherwise, we continue the search by picking a new random index into A. We continue picking random indices into A until we find an index j such that $A[j] = x$ or until we have checked every element of A. Note that we pick from the whole set of indices each time, so that we may examine a given element more than once.

   (a) Write pseudocode for a procedure RandomSearch to implement the strategy above. Be sure that your algorithm terminates when all indices into A have been picked. You may use an additional array.

   (b) Suppose that there is exactly one index i such that $A[i] = x$. What is the expected number of indices into A that we must pick before we find x and RandomSearch terminates?

   (c) Generalizing your solution to part (b), suppose that there are $k \geq 1$ indices i such that $A[i] = x$. What is the expected number of indices into A that we must pick before we find x and Random-Search terminates? Your answer should be a function of n and k.

   (d) Suppose that there are no indices i such that $A[i] = x$. What is the expected number of indices into A that we must pick before we have checked all elements of A and RandomSearch terminates?