Lecture 2 of Analysis of Algorithms: Exercises

1. Food-for thought questions:

   (a.) Can an algorithm have a running time of $\Omega(n^3)$ and $O(n^2 \log n)$ time in the worst case, simultaneously? How, or why not?

   (b.) If an algorithm has a running time of $O(n^2)$ on *all* inputs of size n, can it then have a running time of $\Omega(n \log n)$ in the worst case?

   (c.) If an algorithm has a running time of $\Theta(n^2)$ on *all* inputs of size n, can it then have a running time of $\Omega(n \log n)$ in the worst case?

   (d.) InsertSort takes $\Theta(n^2)$ time on any input (even if the array is already sorted), while BubbleSort may take only $\Theta(n)$ time on certain inputs, but on others it will take $\Theta(n^2)$ time. Can we see this possible difference in the worst case upper bounds in $O(..)$ notation, in the worst case lower bounds in $\Omega(..)$ notation, or in neither?

2. Suppose a task is solved by an algorithm that first does a step that takes $O(n \log n)$ time in the worst case, then a task that takes $O(n)$ time in the worst case, and then a task that has a loop that is executed n times, and in each execution, a task is done that takes $O(\log n)$ time. What is the lowest possible upper bound you can give on the running time of this algorithm?

3. The InsertionSort algorithm takes an array of n numbers. It scans the array to find the largest value, exchanges it with the last number in the array, and then repeats the method with a scan on an array that has one number less (only the first $n-1$ numers). Can you see from this simple textual description that the worst-case running time of InsertionSort is $O(n^2)$ in the worst case? Can you see what the running time is on a sorted array?

4. Write an algorithm in pseudo-code that takes an array $A[0..n-1]$ and a number x, and tests if there are two $i \neq j$ such that $A[i] + A[j] = x$. First, write a brute-force version of the algorithm and analyze its running time. Then, write a more efficient version and analyze its running time.

5. Write an algorithm in pseudo-code that considers all triples of different entries in an array $A[0..n-1]$, and tests if the sum of two of the values is the third value. First, write a brute-force version of the algorithm and analyze its running time. Then, write a more efficient version and analyze its running time.

6. Let us define a *nearly-planar graph* to be a graph that has a planar embedding such that every edge intersects at most one other edge. Can you give a convincing argument why a nearly-planar graph has only a linear number of edges (linear in the number of vertices, n)?

7. An adjacency matrix representation is redundant, because the upper right triangle contains exactly the same information as the bottom left triangle (the matrix is symmetric). And the main diagonal contains only zero's. Suppose we would only store one triangle instead of the whole matrix (somehow). Would this help to get a storage requirement for a graph lower than $\Theta(n^2)$ if the graph has fewer than $\Theta(n^2)$ edges?

8. In an adjacency matrix representation, how much time does it take in $O(..)$ notation to determine the number of neighbors of a vertex i? How much time does it take in the adjacency list representation?