Lecture 2 of Analysis of Algorithms: Solutions to exercises

1. (a.) No, this is not possible. A running time of $\Omega(n^3)$ implies that there exists an input for which at least some constant times n^3 operations are performed, and then the running time cannot be $O(n^2 \log n)$ time in the worst case.

   (b.) Yes. The running time is also $\Omega(n^2)$, but the claim of $\Omega(n \log n)$ is a weaker statement and therefore also true.

   (c.) Yes. The same as (b.).

   (d.) In neither.

2. The running time according to the description is $O(n \log n) + O(n) + O(n) \cdot O(\log n)$. This is also $O(n \log n)$. Just take the biggest constant c and biggest constant n_0 that occurs in any of the four $O(..)$ bounds, and use the same n_0 and $3c^2$ to prove $O(n \log n)$ time overall.

3. On a sorted array, the running time is also $\Theta(n^2)$.

4. The brute-force version has a loop $i \leftarrow 0$ to $n-1$ and nested inside a loop $j \leftarrow i+1$ to $n-1$ and performs the test $A[i] + A[j] = x$. The running time is clearly $O(n^2)$.

   The more efficient version uses MergeSort to sort the array in $O(n \log n)$ time. Then we set $i \leftarrow 0$ and $j \leftarrow n-1$, and in a single loop that terminates when $i = j$, we test compare $A[i] + A[j]$ to x. If they are equal, we know the answer and quit the loop. If x is larger, we increase i and loop. If x is smaller, we decrease j and loop.

   As every time the statements in the loop are performed, $O(1)$ time is spent, and either i is increased, or j decreased, or we are done, we can see that the loop is executed at most n times. After that, $i = j$ and we are done too. So the running time is $O(n \log n) + O(n) = O(n \log n)$.

5. The brute-force version has three nested loops and will take $\Theta(n^3)$ time.

   A more efficient version computes all $\Theta(n^2)$ sums of two values, and puts them in a new array of size $\Theta(n^2)$. This new array is sorted in $O(n^2 \log(n^2)) = O(n^2 \log n)$ time. The original array is also sorted, taking $O(n \log n)$ time. Then a scan through both arrays simultaneously yields the answer. This scan takes $O(n) + O(n^2)$ time, which is $O(n^2)$ time. Hence, we can solve the whole problem in $O(n^2) + O(n^2 \log n) + O(n \log n) + O(n^2)$ time, which is $O(n^2 \log n)$ time.

6. Consider this embedding where any edge has at most one intersection with another edge. Make a new graph where a vertex is created at the place of this intersection. This cuts at most two edges into two pieces, giving four arcs in the new graph when there were two intersecting edges in the original graph. The new graph is planar, and it has at most $n + (3n-5)/2$ vertices. Therefore, it has at most $3(n + (3n-5)/2) - 5$ edges, which is linear in n. Also, the new graph has at least as many edges as the nearly-planar graph we started out with. This completes the argument.

7. No, it does not help. One triangle of the adjacency matrix still has $\Theta(n^2)$ entries. To be precise, it has $\sum_{i=1}^{n-1} i$ entries.

8. In an adjacency matrix representation it takes $\Theta(n)$ time to determine how many neighbors a vertex has. In an adjacency list presentation, this takes time $O(1+k)$ if the vertex has k neighbors. In general this is more efficient, although if a vertex has a linear number of neighbors, it is the same.