CIS 675 Fall 2018 Final Exam Review

Problem 1: In class, we discussed a stack with three operations: Push, Pop, and MultiPop(k). The MultiPop(k) operation is implemented as a series of k pops. Push and Pop each take 1 unit of time, and MultiPop(k) takes k unit of time. Using amortized analysis, we determined that the average cost of an operation, over n operations, was 2 units of time. Suppose that in addition to MultiPop(k), we want to implement a new operation MultiPush(k), which is a series of k pushes. Can we modify our amortized analysis to this case? Does amortized analysis help in this case? Why or why not?

Problem 2: We saw two ways of performing amortized analysis: the 'banking' method and the 'total occurrences' method. In the banking method, cheap operations put extra money into the bank, and these savings paid for expensive operations. In the total occurrences method, we counted how many times operations of each type happened, and found the sum of total costs. When should you select one method over the other? When should you use amortized analysis rather than standard running time analysis? Suppose you use amortized analyis to find the total running time for a sequence of n operations, and find that it is O(n log n), so the average running time for one operation is O(log n). What is the maximum possible worst-case running time for a single operation in this sequence, using standard analysis? (Note: the purpose of this exercise is to help you think about amortized analysis. There is no mathematical problem to solve here.)

Problem 3: Recall the stack with MultiPop from class. In the problem from class, a stack had three operations: Push, Pop, and MultiPop. In the MultiPop operation, the user specifies a value k, and k elements are popped from the stack. Each Push costs \$1, each Pop costs \$1, and each MultiPop cost \$$k$.

We change the problem as follows: In the new version of the problem, each MultiPop costs \$2k, instead of \$$k$ (i.e., \$2 per element that gets popped). Push costs \$2, and Pop costs \$3. Perform an amortized running time analysis on this problem.

Problem 4: In the DOUBLE LONG CYCLE problem, a user inputs an unweighted, undirected graph G and an integer k. The problem is to determine if G contains *two* disjoint cycles, each containing k nodes. Two cycles are *disjoint* if they have no nodes or edges in common. Show that Double Long Cycle is NP-Complete.

Problem 5: 3. In class, we discussed the Sat problem extensively. The 3-Sat problem is the same as Sat, except each clause may only contain three literals (in this version of 3-Sat, suppose every clause must contain *exactly* 3 literals, not just at most 3 literals). The 4-Sat problem is defined analogously. Show that 4-Sat is NP-complete.

Problem 6: In the original Hamiltonian Path problem, we said that each node had to be visited exactly once (and so, clearly, every edge can be used at most once). Suppose we modify the problem and instead say that each node must be visited *at least* once, and individual edges may be used multiple times. Is the problem of determining whether an undirected graph has a modified Hamiltonian Path still NP-Complete?

Problem 7: Suppose we modify the original Hamiltonian Path problem by restricting it only to DAGs. Is it still NP-Complete?

Problem 8. Suppose that we assume that P $\neq$ NP. Suppose that we know that the Traveling Salesman Problem is NP-Complete. Which of the following can you conclude?
a. There is no algorithm that solves arbitrary instances of the Traveling Salesman Problem.
b. There is no efficient algorithm that solves arbitrary instances of the Traveling Salesman Problem.
c. There is an efficient algorithm that solves arbitrary instances of the Traveling Salesman Problem, but no one has found it yet.
d. Any algorithm that solves the Traveling Salesman Problem will take exponentially long for every input.
e. If we show that Problem A reduces to Traveling Salesman Problem, what does that tell us about Problem A?
f. If we show that Traveling Salesman Problem reduces to Problem A, what does that tell us about Problem A?

Problem 8: In the Sat problem, you are given a set of literals and clauses. Each clause contains a set of literals (or their negations), and your job is to find a truth assignment such that each clause has at least one literal that evaluates to True. For instance, if you are given the two clauses $(a \vee b \vee \bar{c})$, $(\bar{a} \vee \bar{b} \vee \bar{c})$, then one way to ensure that each clause has at least one true literal is to set $a =$ False, $b =$ False, $c =$ True. That way, in the first clause, $\bar{c}$ evaluates to True, and in the second clause, $\bar{a}$ evaluates to True.

In the False-Sat problem, you are again given a set of literals and clauses, the same as in Sat. Now, you have to find a truth assignment such that each clause contains at least one *False* literal. The assignment given above also works for this problem, as the first clause contains a false literal (a), as does the second clause ($\bar{c}$).

Prove that False-Sat is NP-Complete.