Amortized Analysis

**Problem 1** (10 points) In Java and many other high-level languages, list types (such as Vector) automatically manage their storage (memory allocation) using a simple algorithm: the initial capacity is set to some small number, and the size of the list (the number of items actually in the list) is kept track of. If inserting an item would cause the size to be greater than the capacity of the list, then the capacity is doubled; memory for the new list is allocated, and all elements are copied from the old list into the new list.

In the worst case, this implies that inserting the $n^{th}$ item into such a list has cost $O(n)$, the cost of copying $n - 1$ items after doubling. Give an amortized analysis of the problem of inserting $n$ items into such a list to show that the amortized cost of inserting any item is $O(1)$.

Union-Find

**Problem 2** (10 points) Consider an undirected graph $G = (V, E)$. Design an algorithm that counts the connected components using a union-find data structure. Provide a complexity analysis of your solution.

Dynamic Programming

**Problem 3** (20 points) Do problem 6.1 in the book.

*Problem 4* (20 points) Do problem 6.2 in the book.

*Problem 5* (20 points) Do problem 6.4 in the book.

*Problem 6* (20 points) Do problem 6.7 in the book.