Problem 17.1-1 (30 points):
If the set of stack operations included a MULTIPUSH operation, which pushes k items onto the stack, would the O(1) bound on the amortized cost of stack operations continue to hold?
Solution:
NO. In worst case for series of n MULTIPUSH(s,k) operations, the total cost would be O(k)*n = O(k*n). Hence, the amortized cost in this case would be O(k*n)/n = O(k).

Problem 17.2-1 (30 points):
Suppose we perform a sequence of stack operations on a stack whose size never exceeds k. After every k operations, we make a copy of the entire stack for backup purposes. Show that the cost of n stack operations, including copying the stack, is O(n) by assigning suitable amortized costs to the various stack operations.
Solution:
We have already seen the total cost to perform sequence of n operations is O(n). Now, we are additionally making a copy of the stack after a series of k operations. We charge twice as much for each stack operation. So after a series of k stack operations, we will have accumulated a total of k$ in the stack. Now as stack size will never exceed k, then the copy operation will require at most k$ which can be paid using the already accumulated k$ in the stack. Thus, the total cost for sequence of n stack operations with additional condition of backing up stack after k operation will also be equal to O(n).

Problem 22.2.7 (40 points):
There are two types of professional wrestlers: “babyfaces” (“good guys”) and “heels” (“bad guys”). Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries. Give an O(n+r) time algorithm that determines whether it is possible to designate some of the wrestlers as babyfaces and the remainder as heels such that each rivalry is between a babyface and a heel. If it is possible to perform such a designation, your algorithm should produce it.
Solution:
We can represent the rivalries as a graph, G = (V, E) where the vertices are the wrestlers, and the edges are the rivalries. Therefore, |V| = n, and |E| = r.

Algorithm:
1. Discard all vertices with degree 0. These are wrestlers who have no rivalries. We are not concerned with them.
2. Separate all connected components of the remaining graph. Process each component individually in the following steps.
3. Let \( C = (V_c, E_c) \) be the connected component. Choose one vertex \( r \) at random from \( V_c \). Let \( r \) be classified as babyface.
4. We use BFS to traverse \( C \) starting from root. All vertices with odd path lengths from the root are labeled as Heels and all the vertices with even path lengths from the root are labeled babyfaces.
5. Next we check every edge in \( E_c \). If the edge is between two vertices whose path lengths from root are both even or both odd, then it means that we have established a rivalry between two babyfaces or two heels. If this is the case, we return FALSE i.e. it is not possible to partition the wrestlers.
6. We run BFS for every such connected component of the graph. If we find every edge in every connected component to join a babyface and a heel, then we have successfully partitioned the wrestlers in step 4.

**Complexity:**
Since the BFS takes time \( O(n + r) \) and the checking all the edges takes \( O(r) \), the total runtime is \( O(n + r) \). Hence, proved.