Problem 1 (Problem 17.1-3)

Solution:

1) Let c_i be the cost of i th operation: then $c_i = i$ if i is an exact power of 2 else 1, for example we will have this below table:

Operation	Cost
1	1
2	2
3	1
4	4
5	1
6	1
…	…

n operations will then cost:

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{lgn} 2^j = n + (2n - 1) < 3n$$

Thus we have,

Average cost of operations = Total cost (3n) / number of operations(n) < 3

So the amortized cost per operation is O(1).

Problem 2 (Problem 17.2-3)

Solution

The basic idea is to use an increment algorithm that is very similar to the one described in CLR: the only modification is that we maintain a pointer (a global variable, that we can call A:max) to the high order bit. Reset is implemented starting at the highest significant bit and iterating through the vector setting each bit to zero.

Initially, A:max is set to 1, since the low-order bit of A is at index 0, and there are initially no 1's in A. The value of A:max is updated as appropriate when the counter is incremented or reset, and we use this value to limit how much of A must be looked at to reset it. By controlling the cost of RESET in this way, we can limit it to an amount that can be covered by credit from earlier INCREMENTs.

As for the counter in the book, we assume that it costs $1 to flip a bit. In addition, we assume it costs $1 to update A:max. Setting and resetting of bits by INCREMENT will work exactly as for the original counter in the book: $1 will pay to set one bit to 1; $1 will be placed on the bit that is set to 1 as credit; the credit on each 1 bit will pay to reset the bit during incrementing. In addition, we'll use $1 to pay to update max, and if max increases, we'll place an additional $1 of credit on the new high-order 1. (If max doesn't increase, we can just waste that $1—it won't be needed.)

Since RESET manipulates bits at positions only up to A:max, and since each bit up to there must have become the high-order 1 at some time before the high-order 1 got up to A:max, every bit seen by RESET has $1 of credit on it. So the zeroing of bits of A by RESET can be completely paid for by the credit stored on the bits. We just need $1 to pay for resetting max. Thus charging $4 for each INCREMENT and $1 for each RESET is sufficient, so the sequence of n INCREMENT and RESET operations takes O(n) time.