

Incrementing a binary counter

As another example of aggregate analysis, consider the problem of implementing a k -bit binary counter that counts upward from 0. We use an array $A[0..k-1]$ of bits, where $\text{length}[A] = k$, as the counter. A binary number x that is stored in the counter has its lowest-order bit in $A[0]$ and its highest-order bit in $A[k-1]$, so that $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$. Initially, $x = 0$, and thus $A[i] = 0$ for $i = 0, 1, \dots, k-1$. To add 1 (modulo 2^k) to the value in the counter, we use the following procedure.

```
INCREMENT(A)
1   $i \leftarrow 0$ 
2  while  $i < \text{length}[A]$  and  $A[i] = 1$ 
3      do  $A[i] \leftarrow 0$ 
4           $i \leftarrow i + 1$ 
5  if  $i < \text{length}[A]$ 
6      then  $A[i] \leftarrow 1$ 
```

Figure 17.2 shows what happens to a binary counter as it is incremented 16 times, starting with the initial value 0 and ending with the value 16. At the start of each iteration of the **while** loop in lines 2–4, we wish to add a 1 into position i . If $A[i] = 1$, then adding 1 flips the bit to 0 in position i and yields a carry of 1, to be added into position $i + 1$ on the next iteration of the loop. Otherwise, the loop ends, and then, if $i < k$, we know that $A[i] = 0$, so that adding a 1 into position i , flipping the 0 to a 1, is taken care of in line 6. The cost of each INCREMENT operation is linear in the number of bits flipped.

~~As with the stack example,~~ a cursory analysis yields a bound that is correct but not tight. A single execution of INCREMENT takes time $\Theta(k)$ in the worst case, in which array A contains all 1's. Thus, a sequence of n INCREMENT operations on an initially zero counter takes time $O(nk)$ in the worst case.

We can tighten our analysis to yield a worst-case cost of $O(n)$ for a sequence of n INCREMENT's by observing that not all bits flip each time INCREMENT is called. As Figure 17.2 shows, $A[0]$ does flip each time INCREMENT is called. The next-highest-order bit, $A[1]$, flips only every other time: a sequence of n INCREMENT operations on an initially zero counter causes $A[1]$ to flip $\lfloor n/2 \rfloor$ times. Similarly, bit $A[2]$ flips only every fourth time, or $\lfloor n/4 \rfloor$ times in a sequence of n INCREMENT's. In general, for $i = 0, 1, \dots, \lfloor \lg n \rfloor$, bit $A[i]$ flips $\lfloor n/2^i \rfloor$ times in a sequence of n INCREMENT operations on an initially zero counter. For $i > \lfloor \lg n \rfloor$,

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	1	0	0	4
4	0	0	0	0	1	0	0	0	7
5	0	0	0	0	1	1	0	0	8
6	0	0	0	0	1	1	1	0	10
7	0	0	0	0	1	1	1	1	11
8	0	0	0	1	0	0	0	0	15
9	0	0	0	1	0	0	0	0	16
10	0	0	0	1	0	1	0	0	18
11	0	0	0	1	0	1	0	0	19
12	0	0	0	1	1	0	0	0	22
13	0	0	0	1	1	0	0	0	23
14	0	0	0	1	1	1	0	0	25
15	0	0	1	0	0	0	0	0	26
16	0	0	1	0	0	0	0	0	31

Figure 17.2 An 8-bit binary counter as its value goes from 0 to 16 by a sequence of 16 INCREMENT operations. Bits that flip to achieve the next value are shaded. The running cost for flipping bits is shown at the right. Notice that the total cost is never more than twice the total number of INCREMENT operations.

bit $A[i]$ never flips at all. The total number of flips in the sequence is thus

$$\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} \\ = 2n,$$

by equation (A.6). The worst-case time for a sequence of n INCREMENT operations on an initially zero counter is therefore $O(n)$. The average cost of each operation, and therefore the amortized cost per operation, is $O(n)/n = O(1)$.