# AMORTIZED ANALYSIS

INSTRUCTOR: VALERIE KING
SCRIBE: VALERIE KING

Amortized analysis looks at the cost of a worst case sequence of operations. Despite the fact that some operations may be costly, amortized analysis is used argue that the average cost of each operation in a sequence of sufficiently many operations must be low.

## 1. Binary counter

We assume there is a binary counter with $n$ bits:

(1) 1111111
(2) 0000000
(3) 00000001
(4) 00000010
(5) etc.

How many flips of digits are needed?
Worst case for one increment is $n$ flips.
Analsyis of aggregate cost:
If there are $n$ operations:

- $n$ ones are written, plus
- bit 0 flips to zero every time an even number (after 0) is reached or $\lfloor n/2 \rfloor$
- bit 1 flips to zero every time a number (after 0) divisible by 4 is reached
- bit $i$ flips to zero every time a number (after 0 ) divisible by $2^i$ is reached
- etc.

so $\sum_0^{\lfloor \lg n \rfloor} \lg n \lfloor n/2^i \rfloor < n * 2$ in total.
But there's another way to see this called the *accounting method:*
Spend a dollar each time you write a 1, put a dollar on the 1, and use it to pay for its change to 0.

**Theorem 1.1.** The total cost is no more than $n + 2(k - 1)$.

*Proof.* Initially you might spend $n$, then 2 dollars after that, so cost is no more than $n + 2(k - 1)$.

$\square$

Note that I'm skipping the potential method.

## 2. Dynamic tables

This is an "online" problem where elements are being inserted one by one. We are required to keep them in a table. We are allowed to rebuild the table periodically. Let $x$ be the empty spaces in the table and $y$ be the number of times an element is inserted or copied over to another table. We want to minimize $x + y$.
Idea of algorithm: Start with an array of size 1.Insert an element. When the table is full, allocate an array twice as big and copy the old info into the table.

**Doubling algorithm**
$\{Initialize\}$
$tablesize = 1$
$numElements = 0$
create array $T$ of size $tablesize$.

$\{Process\ Insertion\ of\ element\ into\ T\ \}$
If $tablesize = numElements$ then begin:
    $tablesize \leftarrow 2 * tablesize$
    create array $T'$ of size $2 * tablesize$
    for $i = 1$ to $tablesize$ do
        $T'(i) \leftarrow T(i)$
    $T \leftarrow T'$; $tablesize \leftarrow 2 * tablesize$
$numElements \leftarrow numElements + 1$
$T(numElements) \leftarrow element$

### 2.1. Analysis.

We determine the cost per insertion. Let $n$ be the number of elements inserted so far. In the worst case, when the $n^{th}$ element is inserted, the first $n-1$ elements are copied over and the last is added for a total cost of $y = n$ for the inserting and copying and $x = n - 1$ for the maximum extra space in the table.

- To analyze the average case:
  1 if there's no copying or $(\sum_{i=1}^{j=2^l} j)+n$ where $l = \lceil \lg n \rceil - 1$, which is $2^{l+1}+n-1 < 3n$ for $y$.
- To analyze this using the accounting method:
  Charge 3 dollars on each new element which is inserted, 1 to pay for the insertion, 1 to stay on the new element, one to go on an old element in the table which doesn't have a dollar already on it. Use these dollars pay for the elements to be copied when the table fills up. Therefore $y \leq 3n$.