Amortized Analysis

Recall binary counter example:

0000 → flip 1 bit
0001 → 2 bits
0010 → 1 bit
0011 → 3 bits.
0100

n	0	1	2	3	4	5	6	7	8
$f(n)$ = flips		1	2	1	3	1	2	1	4
total		1	3	4	7	8	10	11	15

$f(n)$ = bit flips needed to increment $n-1 \to n$

$T(n) = f(1) + f(2) + \dots + f(n)$.

Conjecture: $T(n) < 2n$.

↳ If true, average # of bit flips per increment is $\frac{T(n)}{n} < 2$.

Hence we say incrementing a counter does $O(1)$ bit flips on average, or we say it takes $O(1)$ amortized time.

How can we prove our conjecture?

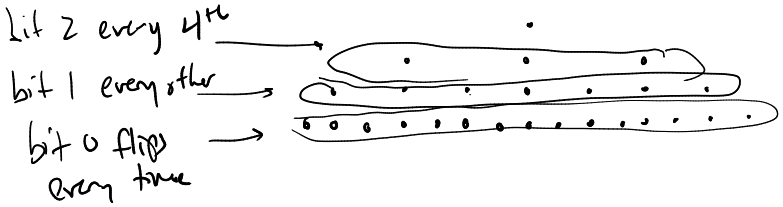
---

① Direct Counting.

"Just" come up with an algebraic formula, show it satisfies desired inequality.

$$T(n) = 1+2+1+3+1+2+1+4+\dots$$

often: change of perspective / regrouping helps!



$$T(n) = n + \lfloor \tfrac{n}{2} \rfloor + \lfloor \tfrac{n}{4} \rfloor + \dots 1.$$

bit 0 flips — bit 1 flips — bit 2 flips

$$\leq n + \frac{n}{2} + \frac{n}{4} + \cdots + 1$$

$$= n\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{n}\right)$$

$$< n\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots\right)$$

$$= 2n.$$

1	1/2	1/4	1/8

Therefore $T(n) < 2n$.

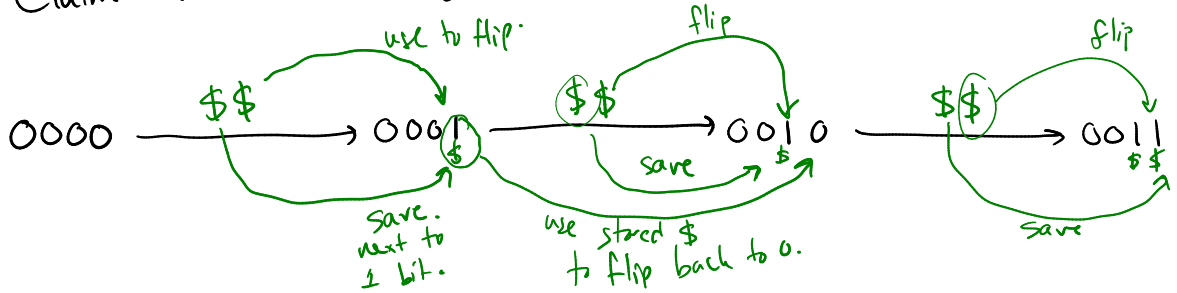
---

(2) Accounting method.

Imagine we are running a counter service. We will keep counters for customers and they can
- Ask for the current value
- Ask to increment.

It costs us \$1 to flip a bit.
How can we set price of increment so we never run out of \$\$?

Claim: we can charge \$2 per increment.



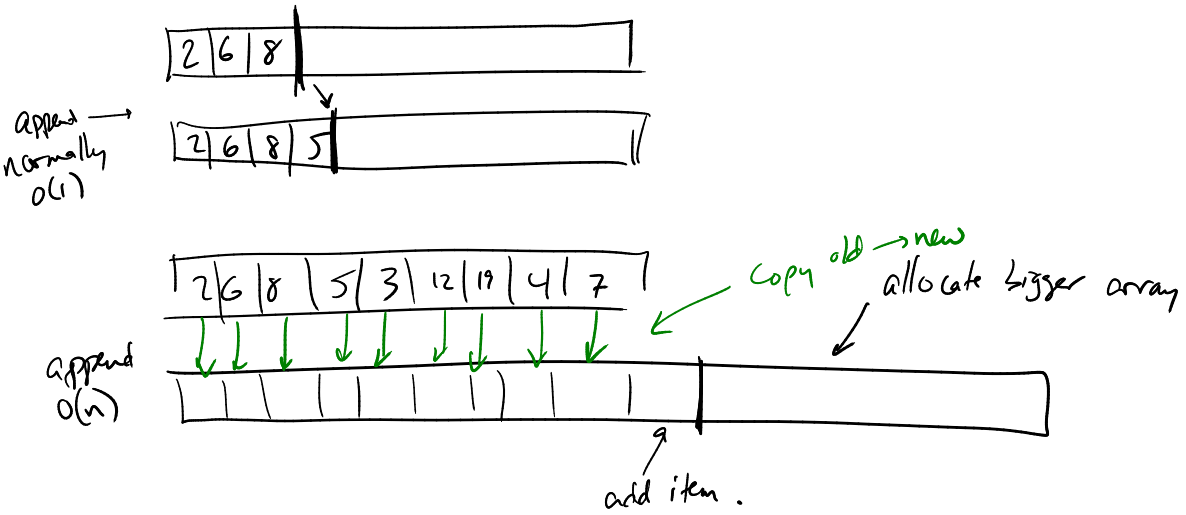
- Always keep \$ saved next to each 1 bit.
- Every increment, - use saved \$ to flip each $1 \to 0$
  - use 1\$ to flip single $0 \to 1$
  - save the other \$ next to the new 1.

Hence, since total paid is always $\geq$ total cost,
the average cost per increment is 2.

Example: Extensible arrays (eg. ArrayList, python lists, Vector, ...)

idea: we can (1) add stuff on the end
(2) look up by index in O(1).

Allocate array w/ extra space



How long does append take on average?