Amortized Analysis

0000 → 1 flip
0001 → 2 flips
0010 → 1 flip
0011 → 3
0100

Q: if we repeatedly increment a binary counter, how many bit flips do we need per increment on average?

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$f(n)$ = flips		1	2	1	3	1	2	1	4	1	2	1	3	1	2	1	5
$T(n)$ = total flips		1	3	4	7	8	10	11	15	16	18	19	22	23	25	26	31

← "ruler function"

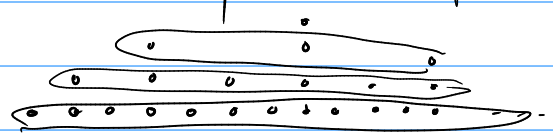
Let $T(n)$ = total # of bit flips to count $1 \ldots n$
$f(n)$ = # of bit flips from $n-1$ to n.

Conjecture: • $T(2^k) = 2^{k+1} - 1$
• $T(n) < 2n$

If this is true, then average # of bit flips needed per increment is $\frac{T(n)}{n} < 2 = \Theta(1)$.

How to prove this?

① Direct counting method — just come up with a formula.
aka find a way to add up all the operations.



- last bit (bit 0) is flipped every increment, for a total of n.
- bit 1 is flipped every other time, for a total of $\lfloor \frac{n}{2} \rfloor$ flips.
- bit 2 ... 4th time ... $\lfloor \frac{n}{4} \rfloor$ flips.

Hence, $T(n) = n + \lfloor \frac{n}{2} \rfloor + \lfloor \frac{n}{4} \rfloor + \cdots + 1$

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

$$\leq n\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\right) = 2n. \checkmark$$

Method 2. The Accounting Method.

Idea:

- Come up w/ cost model, ie. say how much each primitive operation costs. (things we want to count).
- Charge a fee for each operation.
- Show that we will never go broke. ie we always have enough \$\$ on hand to pay for primitive operations.

For binary increments, imagine we are running a counter service.

- Every bit flip cost \$1
- We will charge \$2 per increment.

This only happens once. →

- Every time we flip $0 \to 1$, use \$1 to pay for it and store \$1 next to the 1 bit.

This could happen a bunch. →

- Every time we flip $1 \to 0$, pay for it with the \$1 stored next to the 1.

$$0000 \xrightarrow{\$2} 0001_{\$} \xrightarrow{\$2} 0010_{\$}$$

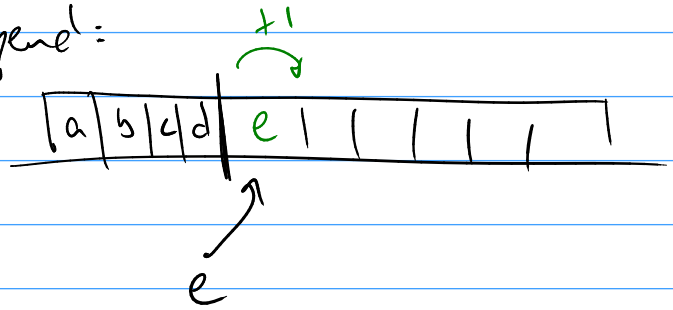
This works since we start at 0 and only do a single $0 \to 1$ flip each increment.

Since we can get away with charging \$2 per increment and will always have enough to cover costs, avg. cost per increment is ≤ 2, hence $\Theta(1)$.

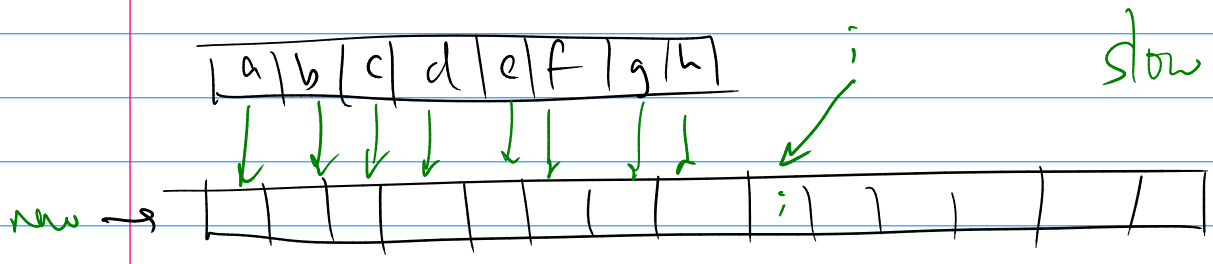
Example: Extensible arrays (ArrayList, Python lists) etc

When we run out of space, allocate a new, larger array + copy everything over.

append:



fast $O(1)$.



slow: $O(n)$.

Policy for making bigger array?

(1) Increase size by c (say, 1000).

- Cost model: inserting or copying one array element costs $\underline{1}$.

Total # of operations needed to do a sequence of n appends?

$$\underbrace{1+1+1+\dots+1}_{c} + \underbrace{c}_{\text{copy}} + \underbrace{1+1+\dots+1}_{c} + \underbrace{2c}_{\text{copy}} + \dots$$

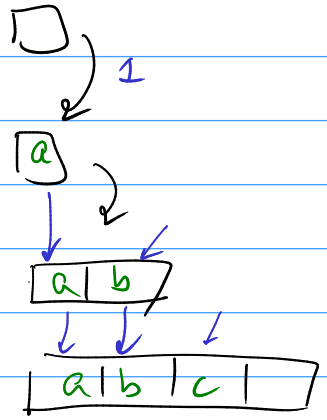
$$= n + c + 2c + 3c + \dots + n$$

$$= n + c\left(1 + 2 + 3 + \dots + \frac{n}{c}\right)$$

$$= n + c \cdot \Theta\left(\left(\frac{n}{c}\right)^2\right) = \Theta(n^2).$$

Strategy 2 : double the size.

① Direct counting argument:



etc.

After inserting 2^k items, we have to do an extra 2^k copy operations.

Total cost looks like

$$1 + 1 + 1 + 2 + 1 + 1 + 4 + 1 + 1 + 1 + 1 + 8 \ldots$$

$$= \underbrace{1 + 1 + \ldots + 1}_{2^k} + (1 + 2 + 4 + 8 + \ldots + 2^{k-1})$$

$$= 1 + 2 + 4 + 8 + \ldots + 2^k$$

$$= 2^{k+1} - 1 = \Theta(2^k)$$

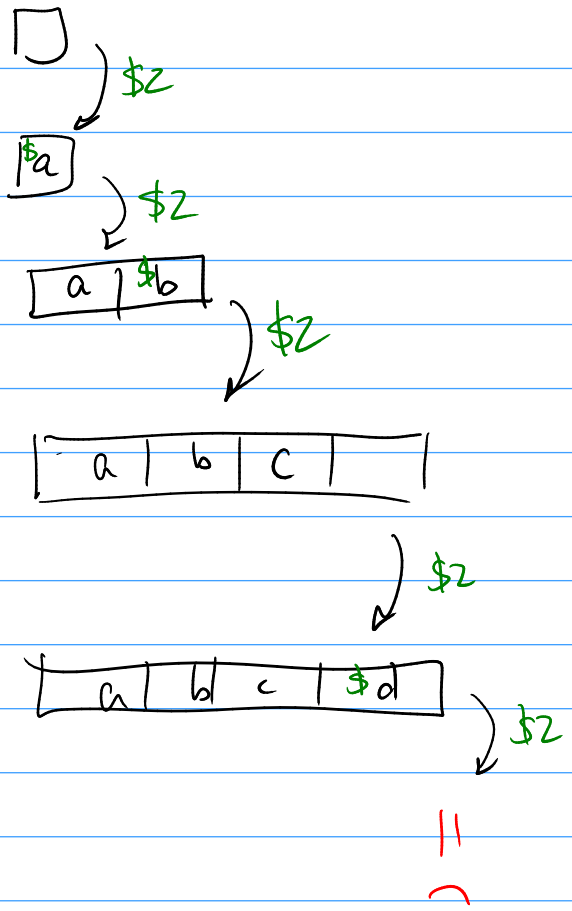
$$= \Theta(n) \text{ if } n = 2^k.$$

---

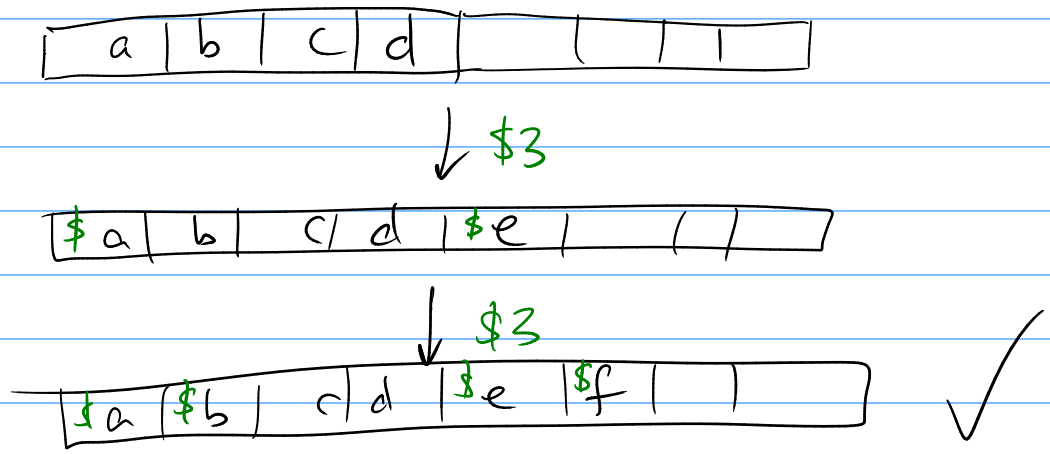
② Accounting Method.

Idea: - overcharge for appends
- by the time we have to do a copy, we have enough \$\$ saved.

How much do we charge? \$2?



What about $3 ?



Growing + shrinking ?

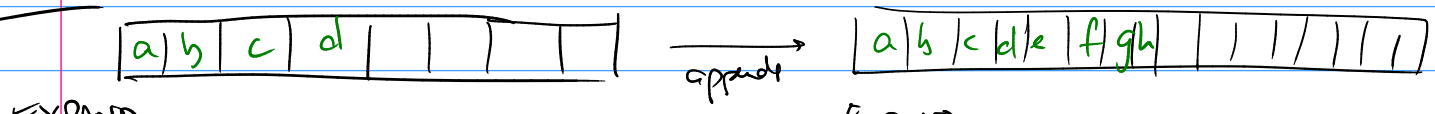
- When full, double size + copy
- When $\leq 1/4$ full, halve size + copy.

Claim: <u>any arbitrary sequence</u> of append + remove_last operations will take $O(1)$ each on average.

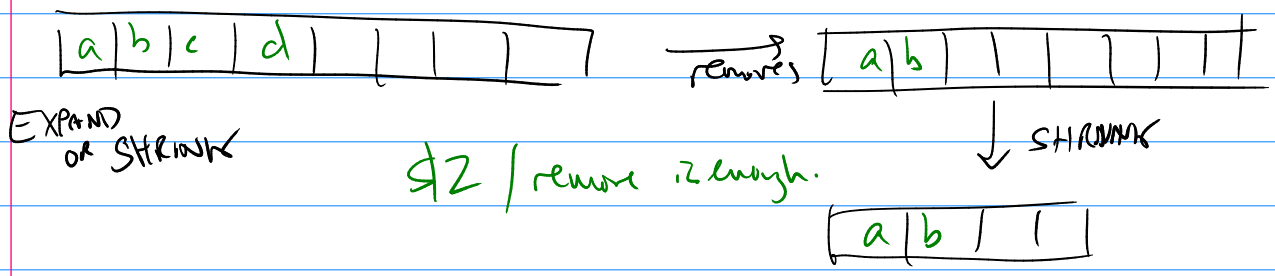
aka, any arbitrary sequence of n append + remove_last operations will take $O(n)$ time in total.

4 cases

(1)



Need \$3 per append.



\$2 / remove is enough.

So charging \$3 / operation is still enough.