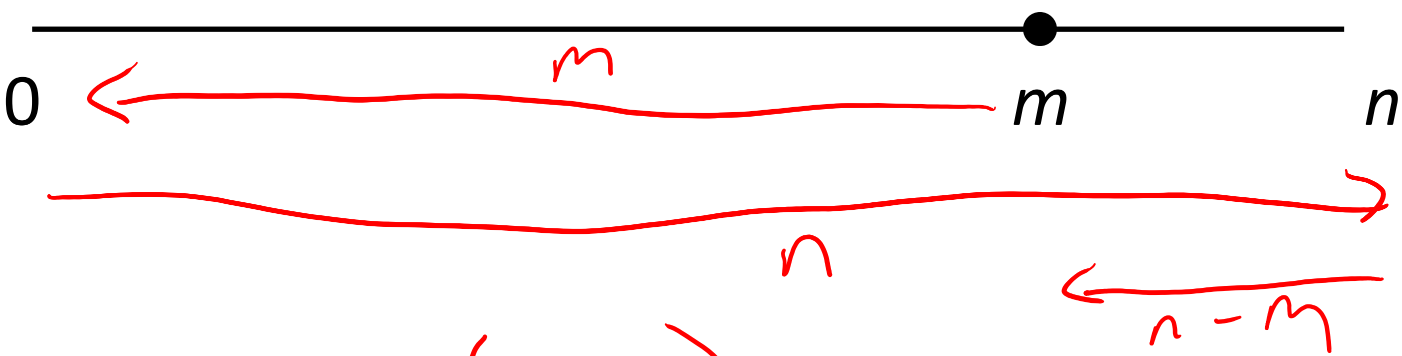
Amortized Analysis

Amortized Analysis: Example

- Suppose you are at point *m* on a number line of length *n*. $0 < m < n$
- You must go to 0, then to *n*, then back to *m*

$2n$

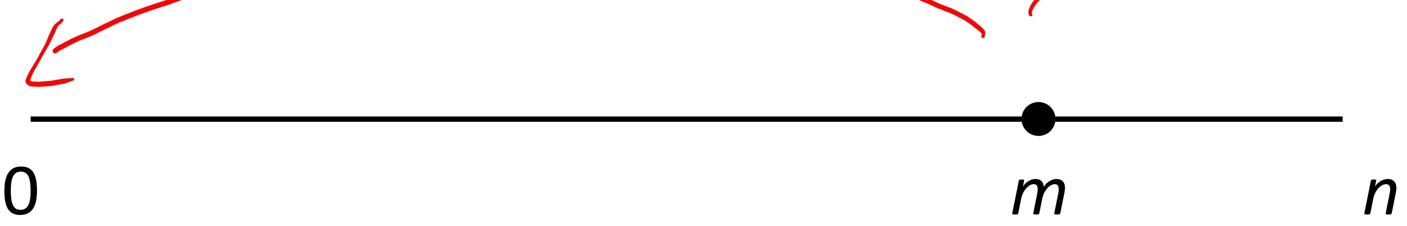
→ Walk2O
→ Walk2N
→ Walk2M(m)



$m + n + (n - m) = 2n$

Amortized Analysis: Example

- Upper bound:
  - Going to 0 takes at most *n* steps   n
  - Going from 0 to *n* takes exactly *n* steps   n
  - Going from *n* to *m* takes at most *n* steps   n
  - Total of 3*n* steps (upper bound) - not a tight upper bound!

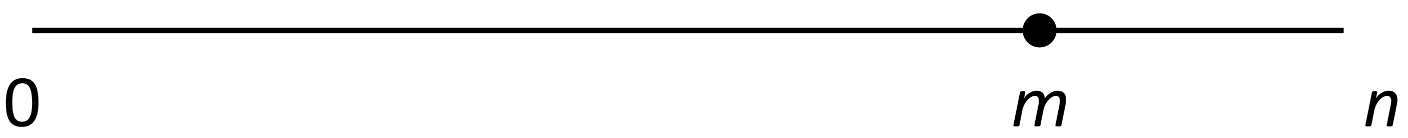


dependencies between time for each

Amortized Analysis: Example

- Can we do better?
  - Distance from m to 0 and from n to m are not independent!

    if $= n$ then $= 0$
  - Their sum is always n
  - Can get a tighter bound: $2n$ steps



Amortized Analysis: Definition

- The amortized cost of a sequence of *n* operations is the ***average cost per operation, for the worst-case sequence*** – allows you to take advantage of dependencies
- How is this different from the expected running time analysis that we saw before (e.g., Quicksort)?

equivalently, report ~~to~~ the total cost for the sequence, taking dependencies into account

Amortized Analysis: Basic Idea

- Keep a bank:
  - When you perform a cheap operation, put the savings in the bank
  - When you perform an expensive operation, use your savings to pay for it
- Just make sure that it never goes negative!

In-Class Exercise: Getting a Cup of Coffee

- Suppose there is a coffee-pot in some room
- People come and pour a cup of coffee; if the coffee-pot is empty, they have to make a new pot of coffee before they pour.
- Each pot contains 10 cups.
- Pouring a cup takes 1 unit of time
- Brewing a new pot takes 10 units of time
- What is the worst-case running time for getting one cup of coffee? The amortized running time?

PourCoffee()

11 units

11·N units of time

average ≈ ? 2

Getting a Cup of Coffee

i	cost
1	1
2	1
3	1
⋮	⋮
10	1
11	11
12	1
13	1
⋮	⋮
20	1
21	11

as $N \to \infty$

$\to 1$

$\to \frac{30}{20} = 1.5$

if we look at first...

10, cost = 1

20, cost = 1.5

30, cost = $\frac{50}{30} = 1\,{}^{2}/_{3}$

40, cost = $\frac{70}{40} = 1.75$

⋮

∞, cost $\to 2$

Getting a Cup of Coffee

Suppose cup costs \$1
pot of coffee costs \$10

Each operation pays \$2: \$1 → cup
\$1 → bank

When an expensive op. occurs (pot is empty, brew a new pot), take money from bank!

Getting a Cup of Coffee

i	actual cost	bank at end of operation
1	\$1 (cup)	\$1
2	\$1 (cup)	\$2
⋮		
10	\$1 (cup)	\$10
11	\$1 (cup) bank (pot of coffee)	\$1
12	\$1 (cup)	\$2
⋮		
20	\$1	\$10

N PC() ops

= \$11N worst case

actual total

= \$2N

Amortized Analysis: Extensible Arrays

- Extensible array: don't need to specify size at creation , only insertion allowed
  - Initially allocate 1 element / unit of space
  - Whenever array of size *k* gets full, allocate 2*k* units of space and copy the old data over
- How long does it take to insert *n* elements? insert copy

Actual cost model: write takes $1

allocating new memory is free

Amortized Analysis: Extensible Arrays

- Standard analysis:
  - $\$n+1$ $\qquad$ $\$1$
  - Each insertion takes O(n) in the worst case: O(1) to insert and O(n) to copy elements to new array
  - $\$n$
  - There are n insertions
  - Total O(n^2)

$$\$n \cdot (n+1) = \$O(n^2)$$

Amortized Analysis: Extensible Arrays

insert
copy/expand

- Amortized analysis:
  - Suppose writing an element costs \$1
  - Every time we insert an element, we will pay \$3: \$1 for insertion, \$2 for the bank
  - If we need to double the array, use money from the bank to pay for it and it'll be covered

i	costs	
1	\$1 + \$1 = 2	[ ]
2	\$1 + \$2 = 3	[x _]
3	\$1	[x x _ _]
4	\$1 + \$4 = 5	[x x x _]
5	\$1	[x x x x _ _ _ _]

* as $N \to \infty$, what is total cost?

Amortized Analysis: Extensible Arrays

- If array is currently of size $k = 2^p$, how much money is needed to pay for doubling it?

$\$k = \underbrace{\$2^p}$

$1 to copy each element

- How much money was added to the bank since the last doubling?

operation 2^{p-1}: last expensive op

$2^p - 2^{p-1} = 2^{p-1}$ opers. in between, so at least $\$2 \cdot 2^{p-1}$ in bank

$\$2^p$

- What is the total running time?

each op. pays $3, so total = $\$3N = O(N)$

In-Class Exercise: Binary Counter

actually

each flip costs \$1

- Suppose you have a binary array
- You are using it as a counter: ~~0000~~
  - In each increment, increase value by 1
  - E.g., [0, 0, 1, 1, 1, 0, 1] -> [0, 0, 1, 1, 1, 1, 0]

00000
00001
00010
00011
00100
⋮

- If there are *n* increments, and in the worst case, each might require changing all values, running time = $O(\cancel{n^2})$ log N

01111
10000

$N \cdot \$(\log N) = O(N \log N)$

In-Class Exercise: Binary Counter

amortized cost per increment = \$2

$\boxed{\$2n}$ for sequence = $O(n)$

- Suppose you have a binary array
- You are using it as a counter:
  - In each increment, increase value by 1
  - E.g., [0, 0, 1, 1, 1, 0, 1] -> [0, 0, 1, 1, 1, 1, 0]
- Suppose that it costs \$1 to flip a bit
- Keep a "bank" for each bit

0 -> 1

- Suppose whenever you flip a bit from 0 to 1, you pay \$2: \$1 to flip, \$1 to that bit's bank, 1 -> 0 flip, take from bank
- Perform amortized running-time analysis and what is total cost for N operations?

In-Class Exercise: Binary Counter

1. What is the algorithm for incrementing a binary counter?
   - start at rightmost bit, if 1, flip to 0 and move left
   - continue til we find a 0, flip to 1 and stop

2. How many 0 → 1 flips can be in an increment? How many 1 → 0 flips?

   1 → \$2
   \$1 for flip
   \$1 for bank

   unlimited

   actual cost of \$1
   amortize cost of \$0

3. If a bit is currently set to 1, what was the last thing done to it? How much is in its bank?

   \$1

   0 → 1 flip

Binary Counter: Another Solution

rightmost

- Over n increments, how often do we flip the first bit? n bit 0
- How often do we flip the second bit? $n/2$ bit 1
- How often do we flip the k^{th} bit? $n/(2^k)$
- What is the total running time?

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

0 0 1 1 1

0 1 0 0 0

1 1 1 1