Algorithms

Lecture Topic: Amortized Analysis

Anxiao (Andrew) Jiang

Roadmap of this lecture:

1. Define "Amortized Analysis".

2. Amortized analysis by the "Aggregate Analysis" technique.

   2.1 Understand "Aggregate Analysis" through the example of "Stack Operations".

   2.2 Understand "Aggregate Analysis" through the example of "Counter Incrementation".

3. Amortized analysis by the "Accounting Method" technique.

   2.1 Understand "Accounting Method" through the example of "Stack Operations".

   2.2 Understand "Accounting Method" through the example of "Counter Incrementation".

Amortized Analysis

for i from 1 to x

$\{$

Operations

$O(y)$

$\}$

$\Rightarrow$ $O(xy)$?

"Amortized Analysis" can sometimes help us get tighter bounds for time complexity.

Quiz question:

1. Why is amortized analysis needed for analyzing time complexity?

Roadmap of this lecture:

1. Define "Amortized Analysis".

2. Amortized analysis by the "Aggregate Analysis" technique.

    2.1 Understand "Aggregate Analysis" through the example of "Stack Operations".

    2.2 Understand "Aggregate Analysis" through the example of "Counter Incrementation".

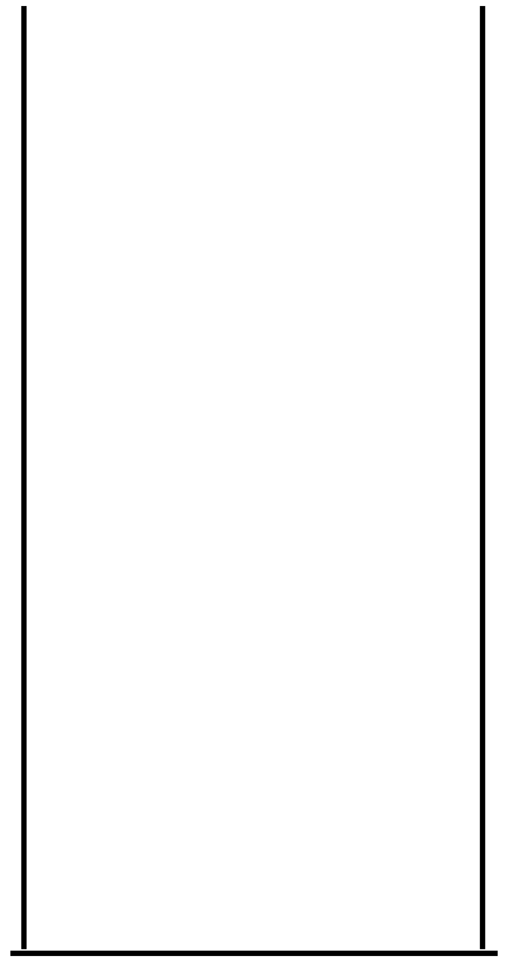
3. Amortized analysis by the "Accounting Method" technique.

    2.1 Understand "Accounting Method" through the example of "Stack Operations".

    2.2 Understand "Accounting Method" through the example of "Counter Incrementation".

Technique 1: Aggregate Analysis

Example: Stack Operations

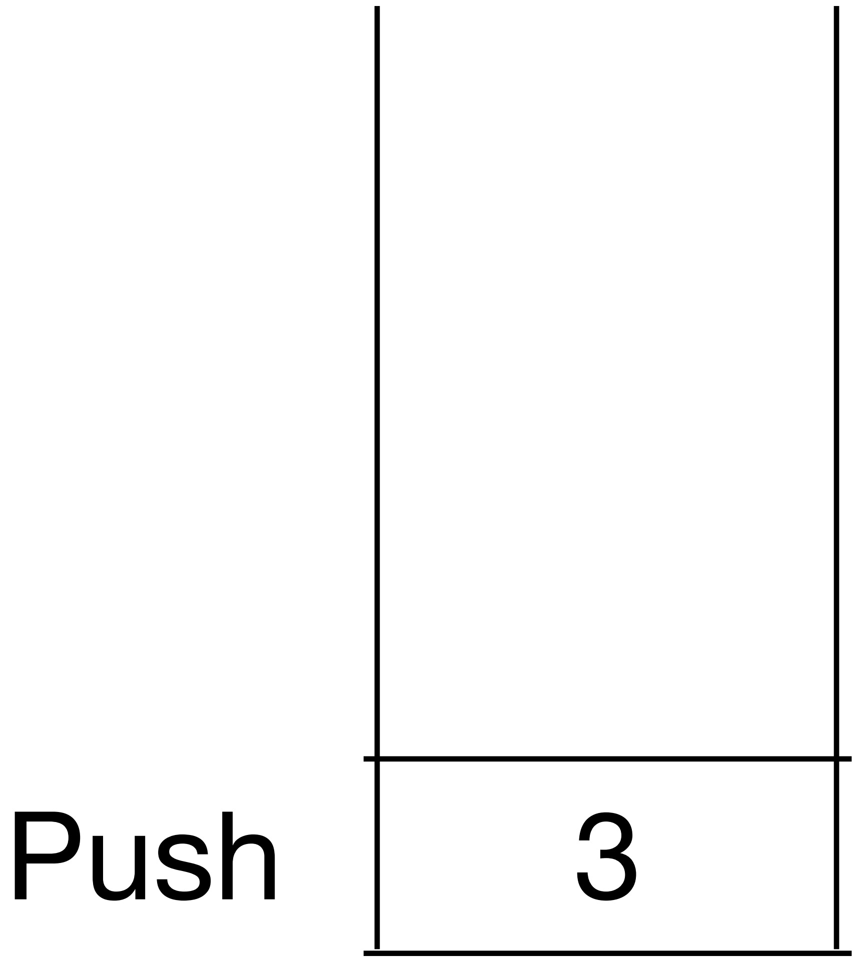


Size of Stack: $|S| = 0$

Stack: first-in-last-out (FILO)

Technique 1: Aggregate Analysis

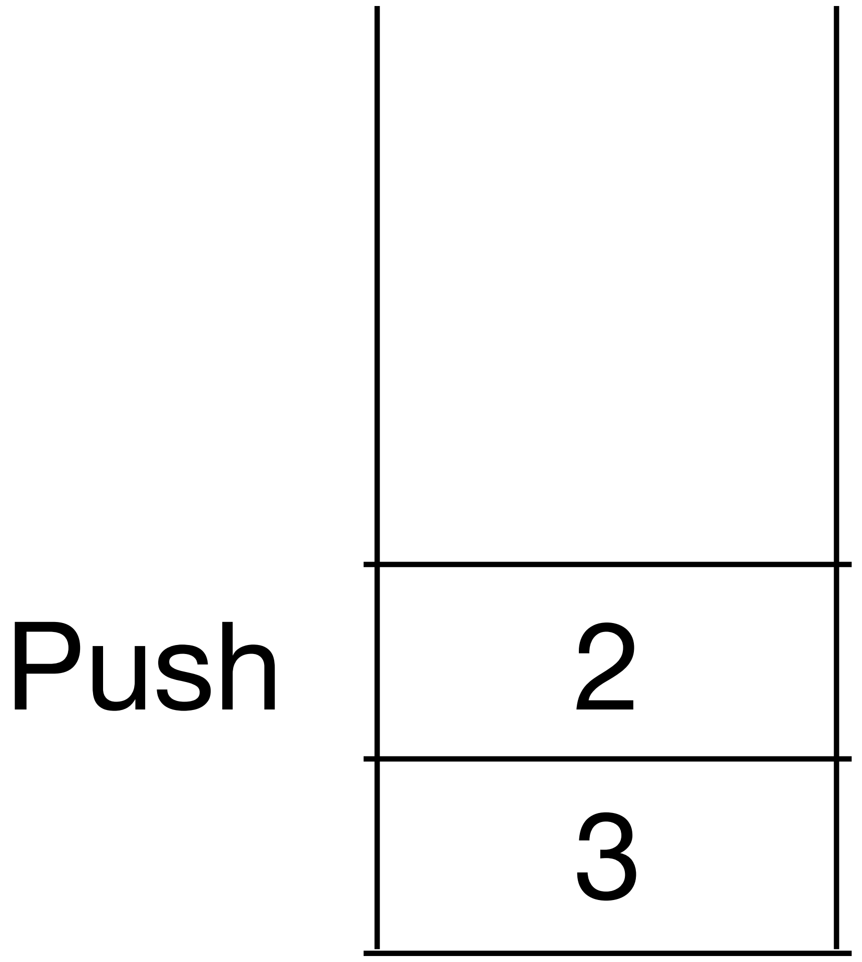
Example: Stack Operations



Size of Stack: $|S| = 1$

Technique 1: Aggregate Analysis

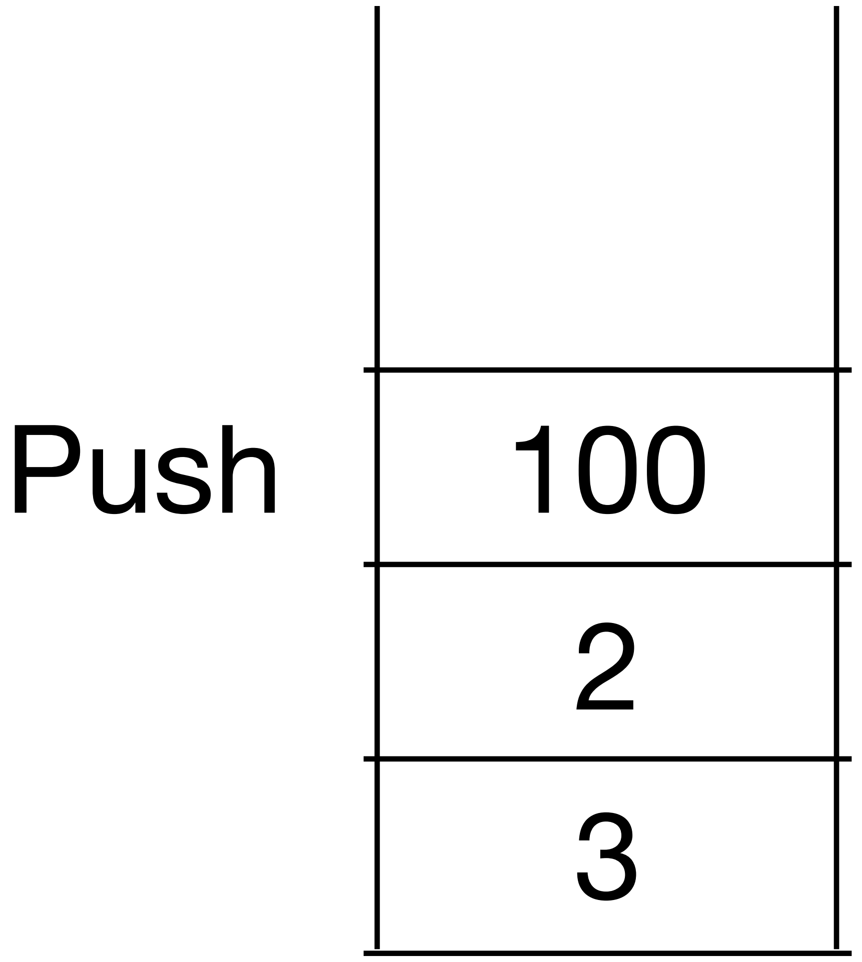
Example: Stack Operations



Size of Stack: $|S| = 2$

Technique 1: Aggregate Analysis

Example: Stack Operations

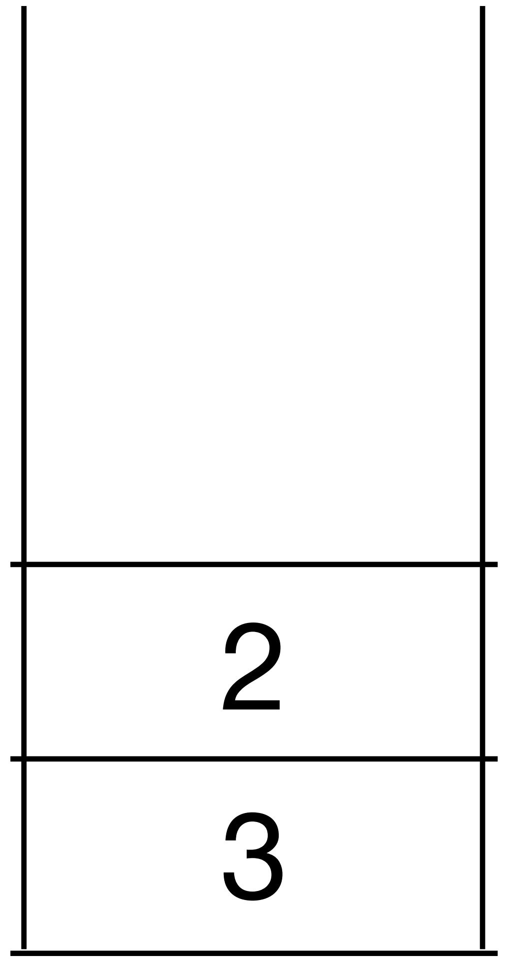


Size of Stack: $|S| = 3$

Technique 1: Aggregate Analysis

Example: Stack Operations

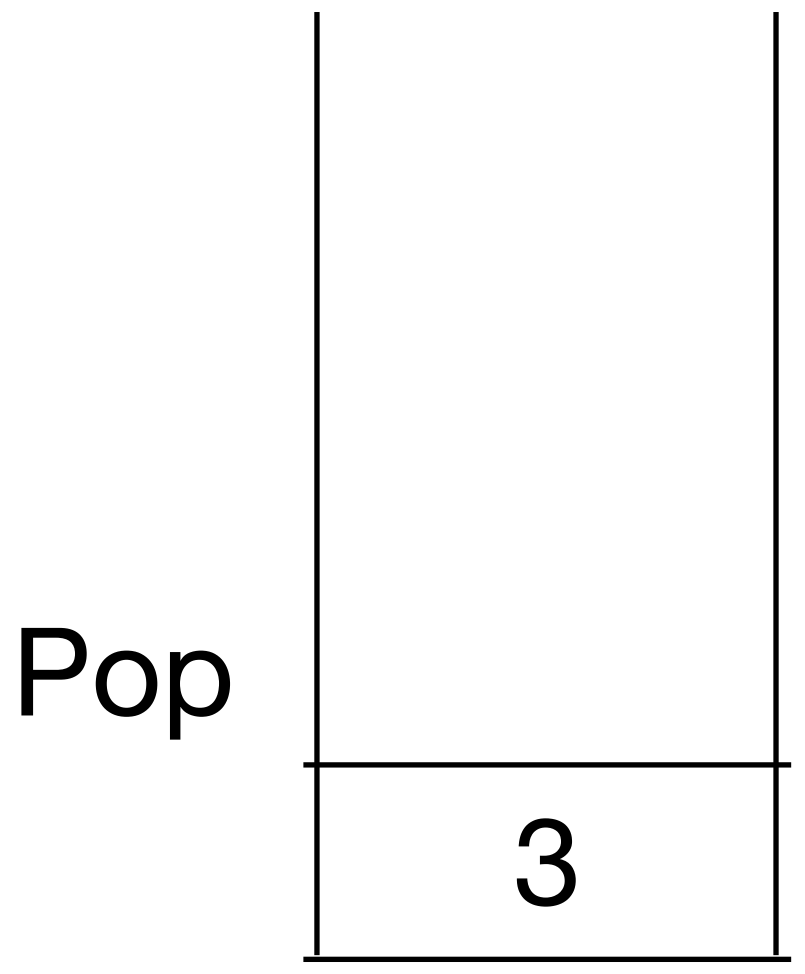
Pop



Size of Stack: $|S| = 2$

Technique 1: Aggregate Analysis

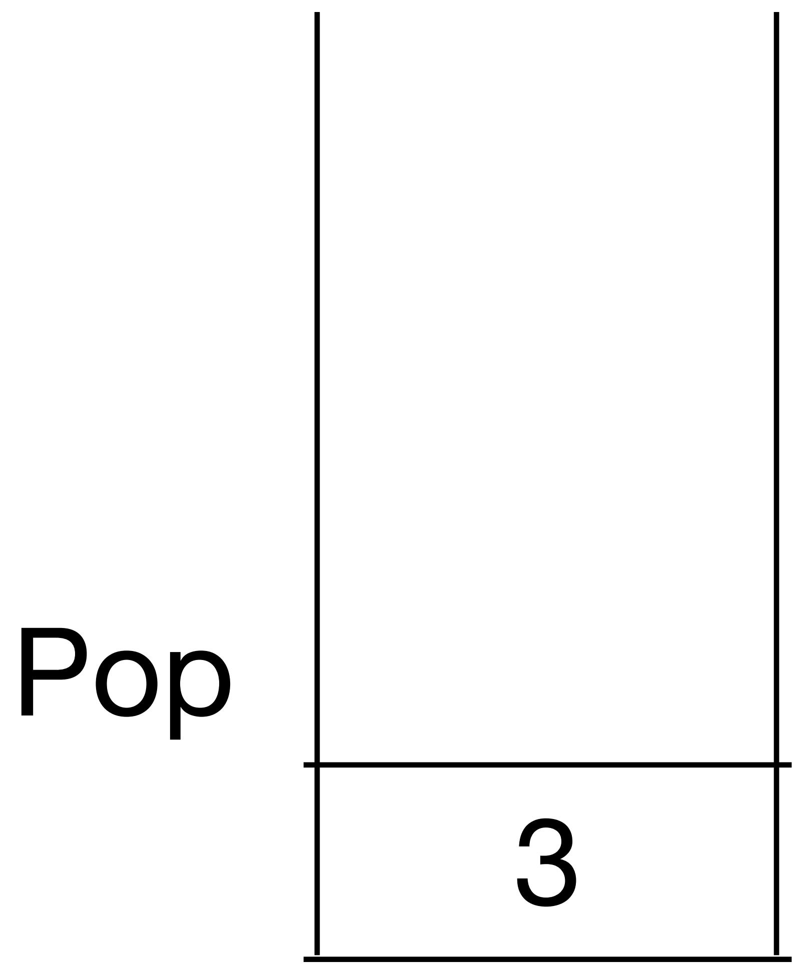
Example: Stack Operations



Size of Stack: $|S| = 1$

Technique 1: Aggregate Analysis

Example: Stack Operations



Size of Stack: $|S| = 1$

Operations:

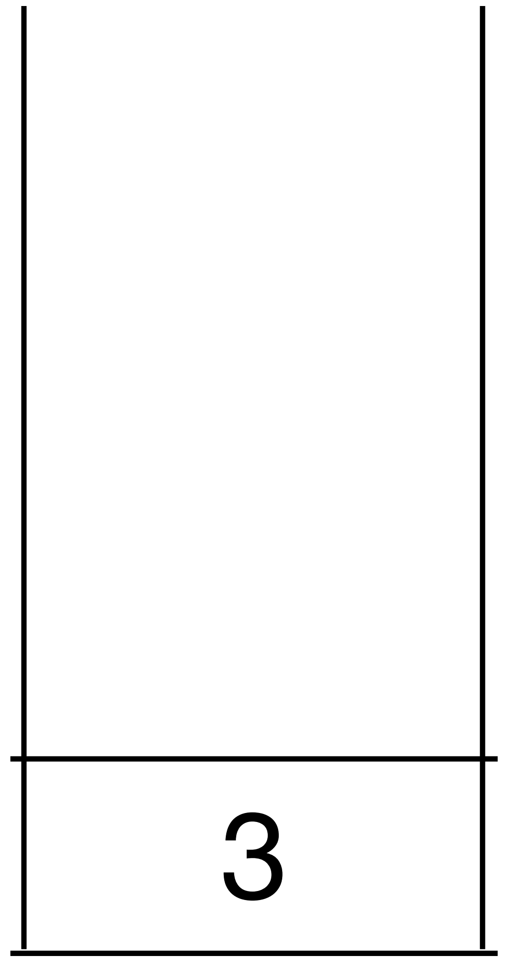
1) PUSH: push a number into stack

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

Technique 1: Aggregate Analysis

Example: Stack Operations

Pop



Size of Stack: $|S| = 1$

Operations:

1) PUSH: push a number into stack

   Cost: 1

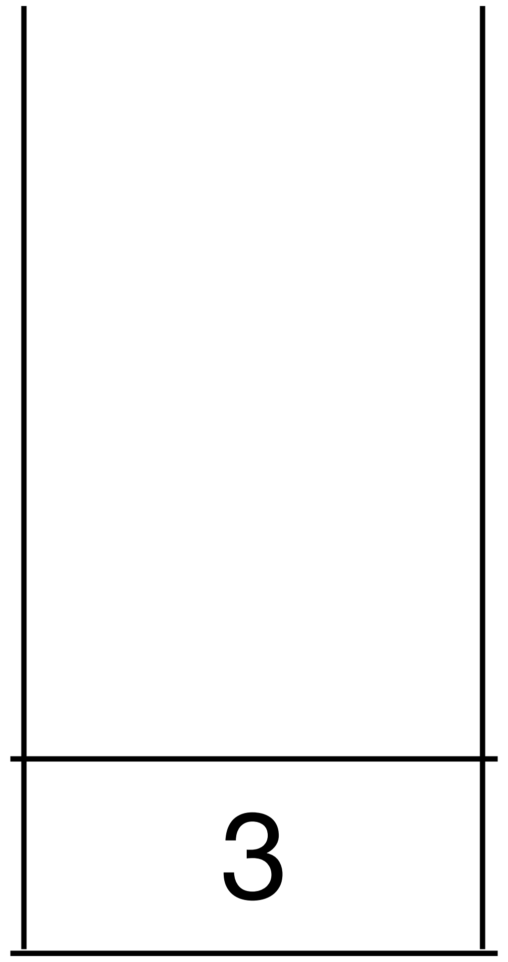
2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

   Cost: $\min\{k, |S|\}$

Technique 1: Aggregate Analysis

Example: Stack Operations

Pop



Size of Stack: $|S| = 1$

Operations:

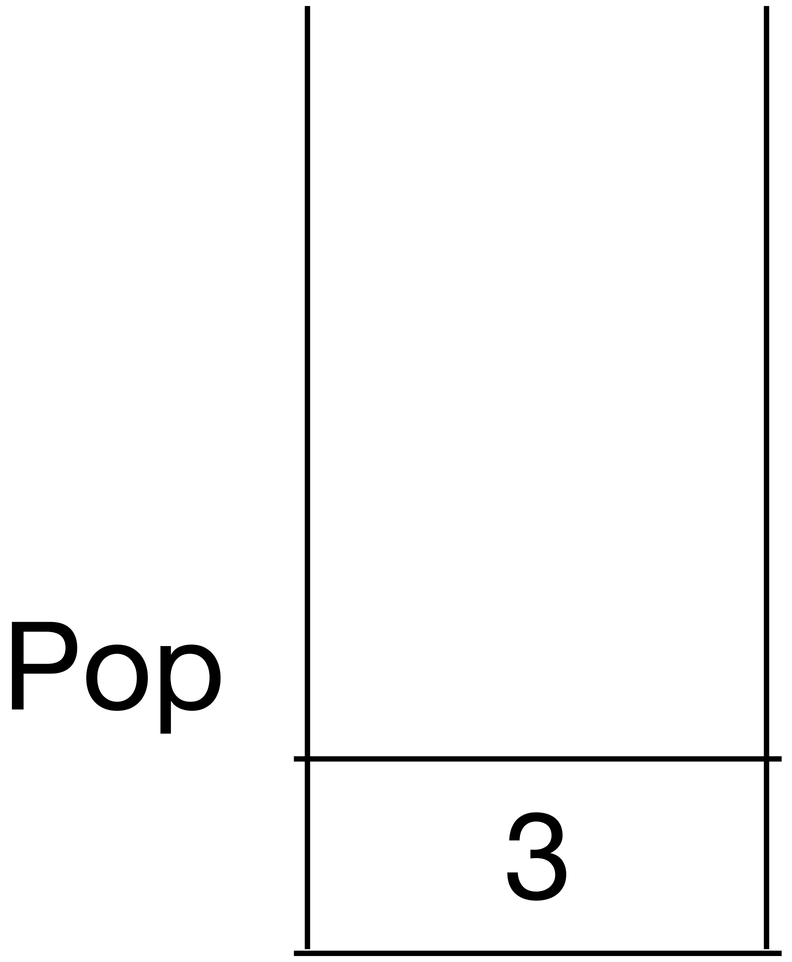
1) PUSH: push a number into stack
   Cost: 1

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.
   Cost: $\min\{k, |S|\}$

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Technique 1: Aggregate Analysis

Example: Stack Operations



Size of Stack: $|S| = 1$

Operations:

1) PUSH: push a number into stack

Cost: 1

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

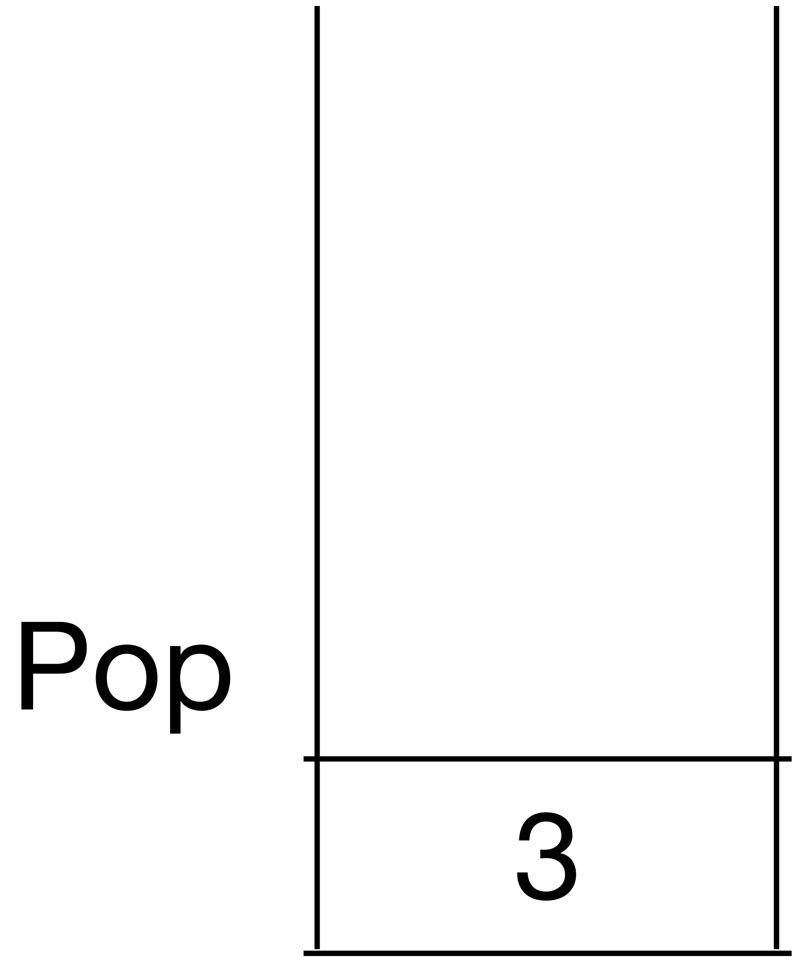
Cost: $\min\{k, |S|\}$

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Is it $O(n^2)$?

Technique 1: Aggregate Analysis

Example: Stack Operations



Size of Stack: $|S| = 1$

Operations:

1) PUSH: push a number into stack
   Cost: 1

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

Cost: $\min\{k, |S|\}$

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Is it $O(n^2)$?

It is actually $O(n)$

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Observation: The stack is initially empty. To pop out a number, we first need to push it in to the stack.

So the cost of POP operations can never be more than the cost of PUSH operations.
(No matter what k is in POP(k).)

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Observation: The stack is initially empty. To pop out a number, we first need to push it in to the stack.

So the cost of POP operations can never be more than the cost of PUSH operations.
(No matter what k is in POP(k).)

Total cost of PUSH operations $\leq n$

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Observation: The stack is initially empty. To pop out a number, we first need to push it in to the stack.

So the cost of POP operations can never be more than the cost of PUSH operations.
(No matter what k is in POP(k).)

Total cost of PUSH operations $\leq n$

Total cost of POP operations $\leq n$

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Observation: The stack is initially empty. To pop out a number, we first need to push it in to the stack.

So the cost of POP operations can never be more than the cost of PUSH operations.
(No matter what k is in POP(k).)

Total cost of PUSH operations $\leq n$

Total cost of POP operations $\leq n$

Total cost of n operations: $\leq 2n$, or $O(n)$

Quiz questions:

1. What is the dependency between push and pop (including pop(k)) operations?

2. How is the above dependency used for "aggregate analysis"?

Roadmap of this lecture:

1. Define "Amortized Analysis".
2. Amortized analysis by the "Aggregate Analysis" technique.
   - 2.1 Understand "Aggregate Analysis" through the example of "Stack Operations".
   - 2.2 Understand "Aggregate Analysis" through the example of "Counter Incrementation".
3. Amortized analysis by the "Accounting Method" technique.
   - 2.1 Understand "Accounting Method" through the example of "Stack Operations".
   - 2.2 Understand "Accounting Method" through the example of "Counter Incrementation".

Technique 1: Aggregate Analysis

Example: Counter Incrementation

Counter: 0, 1, 2, 3, ……

Large Binary Counter

0 0 0 0 0 0 0 0 0 0 0

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

cost: 1

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

cost: 1

cost: 2

Cost of incrementing counter:

Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

cost: 1

cost: 2

cost: 1

Cost of incrementing counter:

Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

cost: 1
cost: 2
cost: 1
cost: 3

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

cost: 1
cost: 2
cost: 1
cost: 3
cost: 1

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	
7	0 0 0 0 0 0 0 1 1 1	
8	0 0 0 0 0 0 1 0 0 0	
9	0 0 0 0 0 0 1 0 0 1	
10	0 0 0 0 0 0 1 0 1 0	
11	0 0 0 0 0 0 1 0 1 1	

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

cost: 1
cost: 2
cost: 1
cost: 3
cost: 1
cost: 2
cost: 1

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	
9	0 0 0 0 0 0 1 0 0 1	
10	0 0 0 0 0 0 1 0 1 0	
11	0 0 0 0 0 0 1 0 1 1	

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

cost: 1
cost: 2
cost: 1
cost: 3
cost: 1
cost: 2
cost: 1
cost: 4
cost: 1

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	cost: 1
9	0 0 0 0 0 0 1 0 0 1	cost: 2
10	0 0 0 0 0 0 1 0 1 0	
11	0 0 0 0 0 0 1 0 1 1	

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0	
		cost: 1
1	0 0 0 0 0 0 0 0 0 1	
		cost: 2
2	0 0 0 0 0 0 0 0 1 0	
		cost: 1
3	0 0 0 0 0 0 0 0 1 1	
		cost: 3
4	0 0 0 0 0 0 0 1 0 0	
		cost: 1
5	0 0 0 0 0 0 0 1 0 1	
		cost: 2
6	0 0 0 0 0 0 0 1 1 0	
		cost: 1
7	0 0 0 0 0 0 0 1 1 1	
		cost: 4
8	0 0 0 0 0 0 1 0 0 0	
		cost: 1
9	0 0 0 0 0 0 1 0 0 1	
		cost: 2
10	0 0 0 0 0 0 1 0 1 0	
		cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0 0 0 0 0 0 0 0 0 0 0
cost: 1
1 0 0 0 0 0 0 0 0 0 1
cost: 2
2 0 0 0 0 0 0 0 0 1 0
cost: 1
3 0 0 0 0 0 0 0 0 1 1
cost: 3
4 0 0 0 0 0 0 0 1 0 0
cost: 1
5 0 0 0 0 0 0 0 1 0 1
cost: 2
6 0 0 0 0 0 0 0 1 1 0
cost: 1
7 0 0 0 0 0 0 0 1 1 1
cost: 4
8 0 0 0 0 0 0 1 0 0 0
cost: 1
9 0 0 0 0 0 0 1 0 0 1
cost: 2
10 0 0 0 0 0 0 1 0 1 0
cost: 1
11 0 0 0 0 0 0 1 0 1 1

What is the total cost of n increments?

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0	
		cost: 1
1	0 0 0 0 0 0 0 0 0 1	
		cost: 2
2	0 0 0 0 0 0 0 0 1 0	
		cost: 1
3	0 0 0 0 0 0 0 0 1 1	
		cost: 3
4	0 0 0 0 0 0 0 1 0 0	
		cost: 1
5	0 0 0 0 0 0 0 1 0 1	
		cost: 2
6	0 0 0 0 0 0 0 1 1 0	
		cost: 1
7	0 0 0 0 0 0 0 1 1 1	
		cost: 4
8	0 0 0 0 0 0 1 0 0 0	
		cost: 1
9	0 0 0 0 0 0 1 0 0 1	
		cost: 2
10	0 0 0 0 0 0 1 0 1 0	
		cost: 1
11	0 0 0 0 0 0 1 0 1 1	

What is the total cost of n increments?

$O(n^2)$?

$O(n \log n)$?

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0	
		cost: 1
1	0 0 0 0 0 0 0 0 0 1	
		cost: 2
2	0 0 0 0 0 0 0 0 1 0	
		cost: 1
3	0 0 0 0 0 0 0 0 1 1	
		cost: 3
4	0 0 0 0 0 0 0 1 0 0	
		cost: 1
5	0 0 0 0 0 0 0 1 0 1	
		cost: 2
6	0 0 0 0 0 0 0 1 1 0	
		cost: 1
7	0 0 0 0 0 0 0 1 1 1	
		cost: 4
8	0 0 0 0 0 0 1 0 0 0	
		cost: 1
9	0 0 0 0 0 0 1 0 0 1	
		cost: 2
10	0 0 0 0 0 0 1 0 1 0	
		cost: 1
11	0 0 0 0 0 0 1 0 1 1	

What is the total cost of n increments?

$O(n^2)$?

$O(n \log n)$?

It is actually: $O(n)$

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

Cost of the last bit: n

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

Cost of the last bit: n

Cost of 2nd last bit: $\leq \frac{n}{2}$

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

Cost of the last bit: n

Cost of 2nd last bit: $\leq \dfrac{n}{2}$

Cost of 3rd last bit: $\leq \dfrac{n}{4}$

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

Cost of the last bit: n

Cost of 2nd last bit: $\leq \frac{n}{2}$

Cost of 3rd last bit: $\leq \frac{n}{4}$

Cost of 4th last bit: $\leq \frac{n}{8}$

……

Cost of incrementing counter:
Number of bits that are changed.

Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

Cost of incrementing counter:
Number of bits that are changed.

Cost of the last bit: n

Cost of 2nd last bit: $\leq \frac{n}{2}$

Cost of 3rd last bit: $\leq \frac{n}{4}$

Cost of 4th last bit: $\leq \frac{n}{8}$

......

Total cost $\leq n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} \cdots \leq 2n$

O(n)

Quiz questions:

1. For each bit position, how often does the bit there change with the increment operations?

2. How is the above property used for “aggregate analysis”?

Roadmap of this lecture:

1. Define "Amortized Analysis".

2. Amortized analysis by the "Aggregate Analysis" technique.

    2.1 Understand "Aggregate Analysis" through the example of "Stack Operations".

    2.2 Understand "Aggregate Analysis" through the example of "Counter Incrementation".

3. Amortized analysis by the "Accounting Method" technique.

    2.1 Understand "Accounting Method" through the example of "Stack Operations".

    2.2 Understand "Accounting Method" through the example of "Counter Incrementation".

Technique 2: Accounting Method

Consider n operations.

For $i = 1, 2, \cdots, n$,

let C_i be the real cost of the i-th operation,

let $\hat{C}_i$ be the amortized cost of the i-th operation.

such that

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Technique 2: Accounting Method

Consider n operations.

For $i = 1,2,\cdots,n$,

let C_i be the real cost of the i-th operation,

let $\hat{C}_i$ be the amortized cost of the i-th operation.

such that

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

If we get an upper bound for the amortized cost $\sum_{i=1}^{n} \hat{C}_i$,

we also get an upper bound for the real cost $\sum_{i=1}^{n} C_i$

Quiz questions:

1. Why do we require the total amortized cost to be no less than the total real cost?

2. For an individual operation, is its amortized cost more or less than its real cost?

Roadmap of this lecture:

1. Define "Amortized Analysis".

2. Amortized analysis by the "Aggregate Analysis" technique.

   2.1 Understand "Aggregate Analysis" through the example of "Stack Operations".

   2.2 Understand "Aggregate Analysis" through the example of "Counter Incrementation".

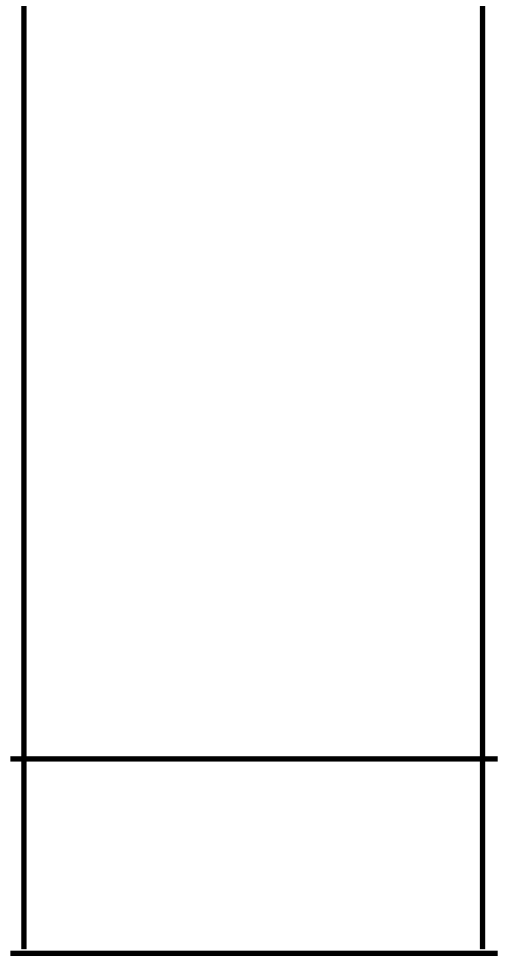
3. Amortized analysis by the "Accounting Method" technique.

   2.1 Understand "Accounting Method" through the example of "Stack Operations".

   2.2 Understand "Accounting Method" through the example of "Counter Incrementation".

Technique 2: Accounting Method

Example: Stack Operations



Size of Stack: $|S| = 0$

Operations:

1) PUSH: push a number into stack

   Cost: 1

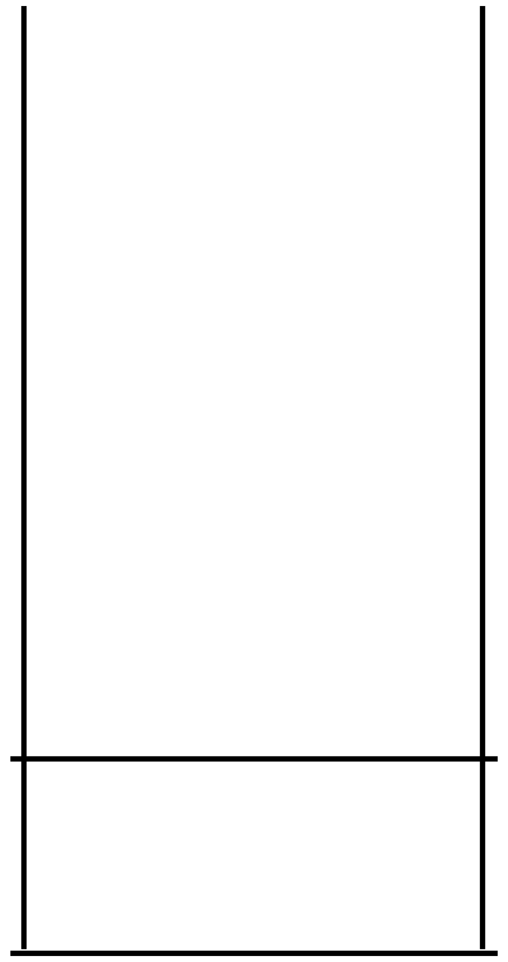
2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

   Cost: $\min\{k, |S|\}$

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Technique 2: Accounting Method

Example: Stack Operations



Size of Stack: $|S| = 0$

Operations:

1) PUSH: push a number into stack

Cost: 1 Amortized Cost: 2

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

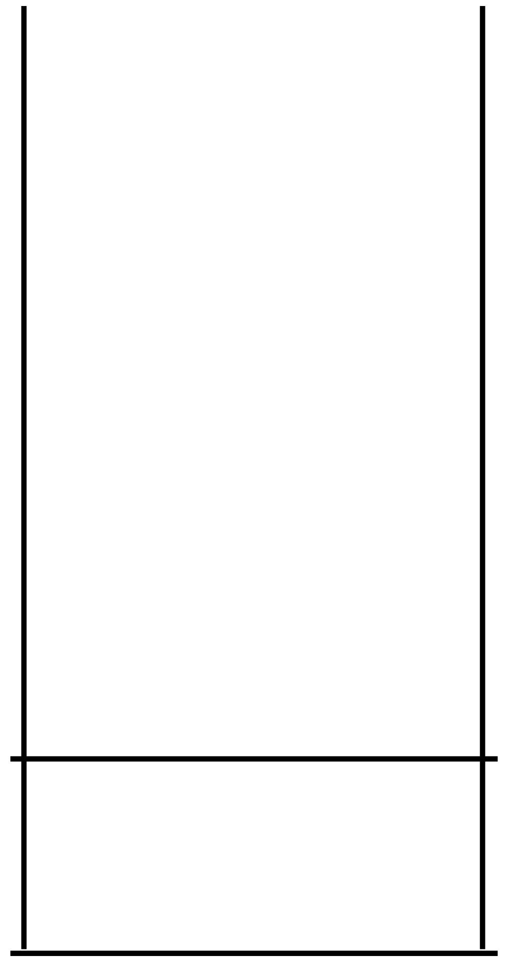
Cost: $\min\{k, |S|\}$

Amortized Cost: 0

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Technique 2: Accounting Method

Example: Stack Operations



Size of Stack: $|S| = 0$

Operations:

1) PUSH: push a number into stack

Cost: 1 Amortized Cost: 2

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

Cost: $\min\{k, |S|\}$

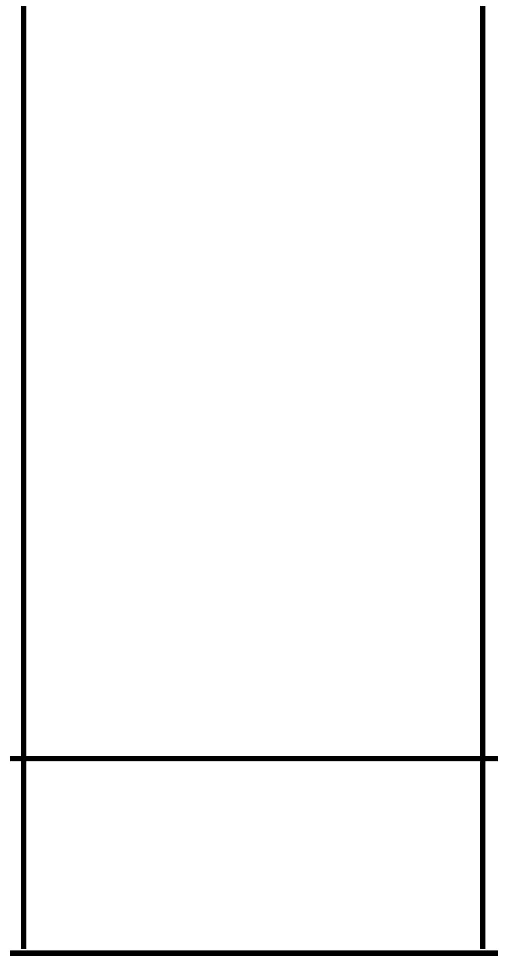
Amortized Cost: 0

Is it true that $\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$?

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Technique 2: Accounting Method

Example: Stack Operations



Size of Stack: $|S| = 0$

Total real cost: 0

Total amortized cost: 0

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Operations:

1) PUSH: push a number into stack

Cost: 1 Amortized Cost: 2

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

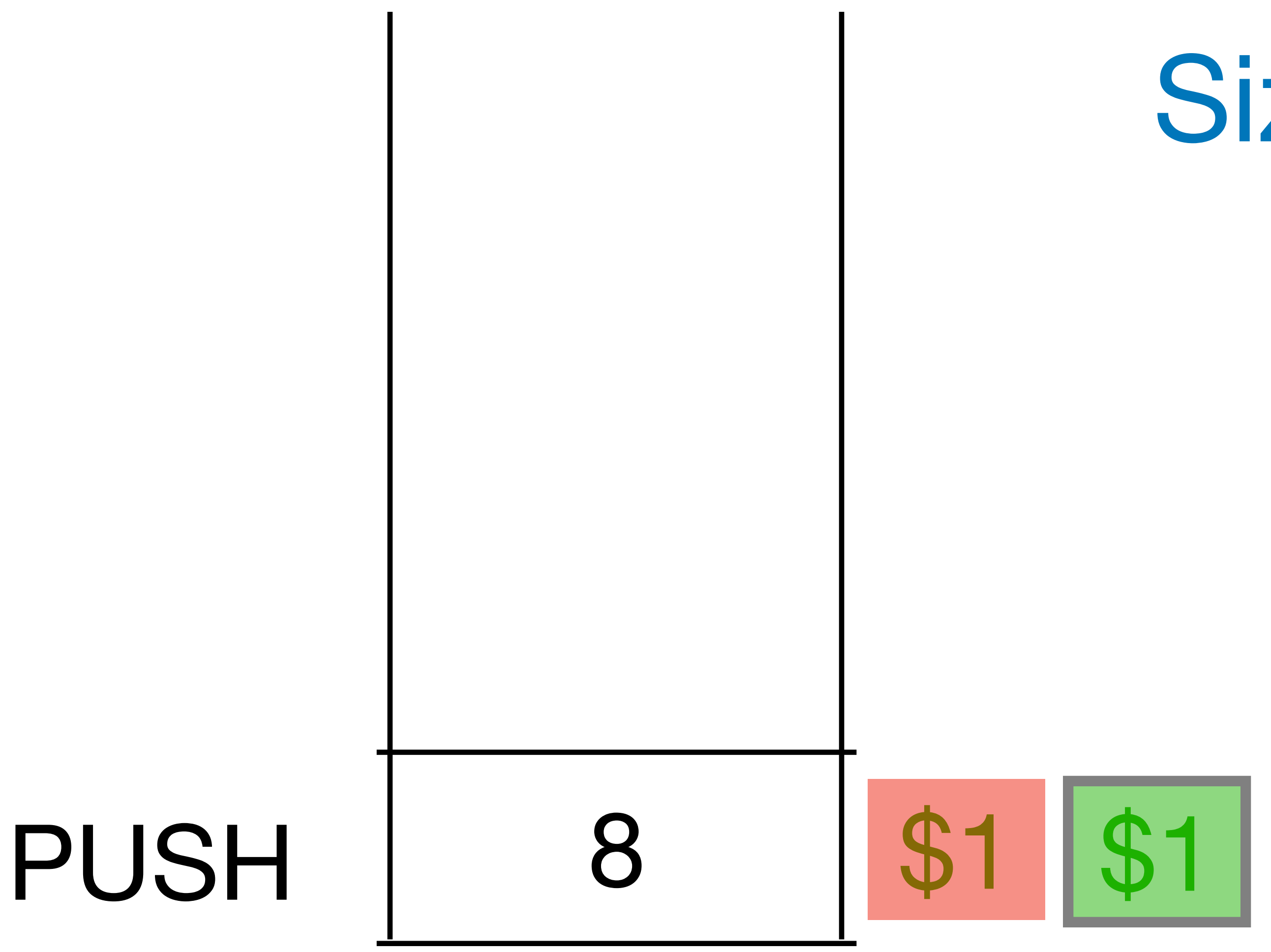
Cost: $\min\{k, |S|\}$

Amortized Cost: 0

Is it true that $\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$?

Technique 2: Accounting Method

Example: Stack Operations



Size of Stack: $|S| = 0$

Total real cost: 1
Total amortized cost: 2

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Operations:

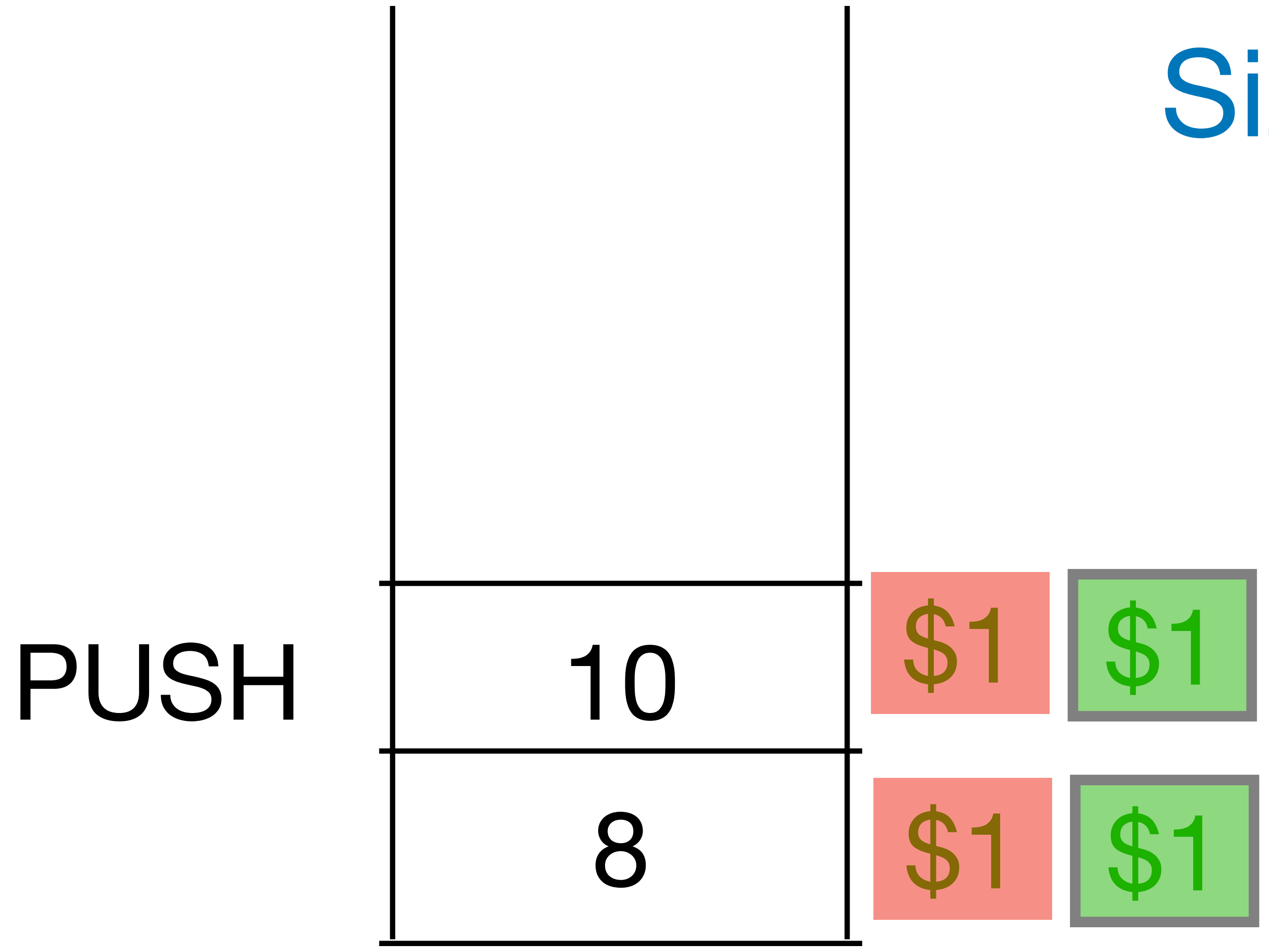
1) PUSH: push a number into stack
   Cost: 1 Amortized Cost: 2

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.
   Cost: $\min\{k, |S|\}$
   Amortized Cost: 0

Is it true that $\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$?

Technique 2: Accounting Method

Example: Stack Operations



Size of Stack: $|S| = 0$

Total real cost: 2
Total amortized cost: 4

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Operations:

1) PUSH: push a number into stack

Cost: 1 Amortized Cost: 2

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

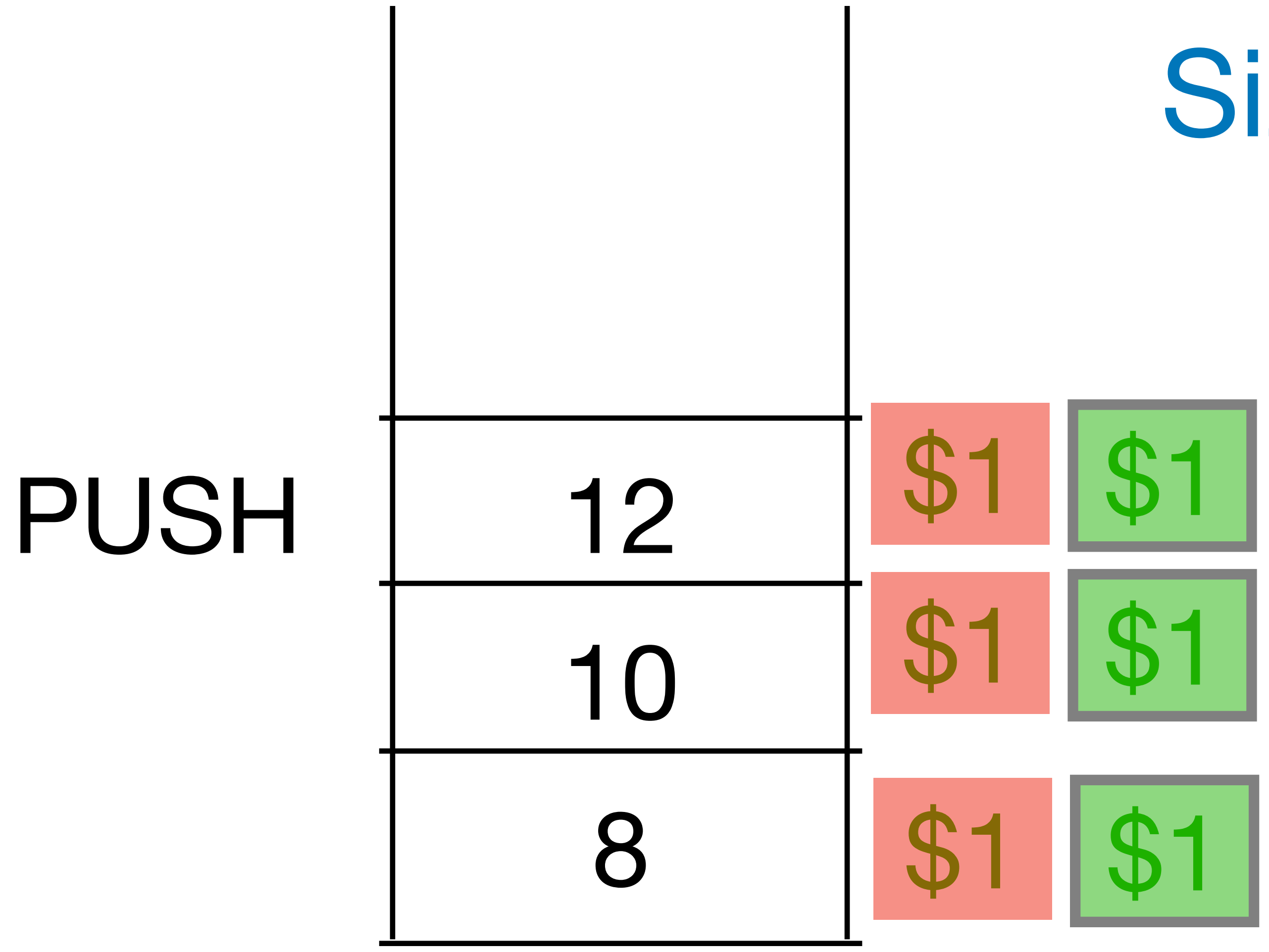
Cost: $\min\{k, |S|\}$

Amortized Cost: 0

Is it true that $\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$?

Technique 2: Accounting Method

Example: Stack Operations



Size of Stack: $|S| = 0$

Total real cost: 3

Total amortized cost: 6

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Operations:

1) PUSH: push a number into stack

Cost: 1 Amortized Cost: 2

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

Cost: $\min\{k, |S|\}$

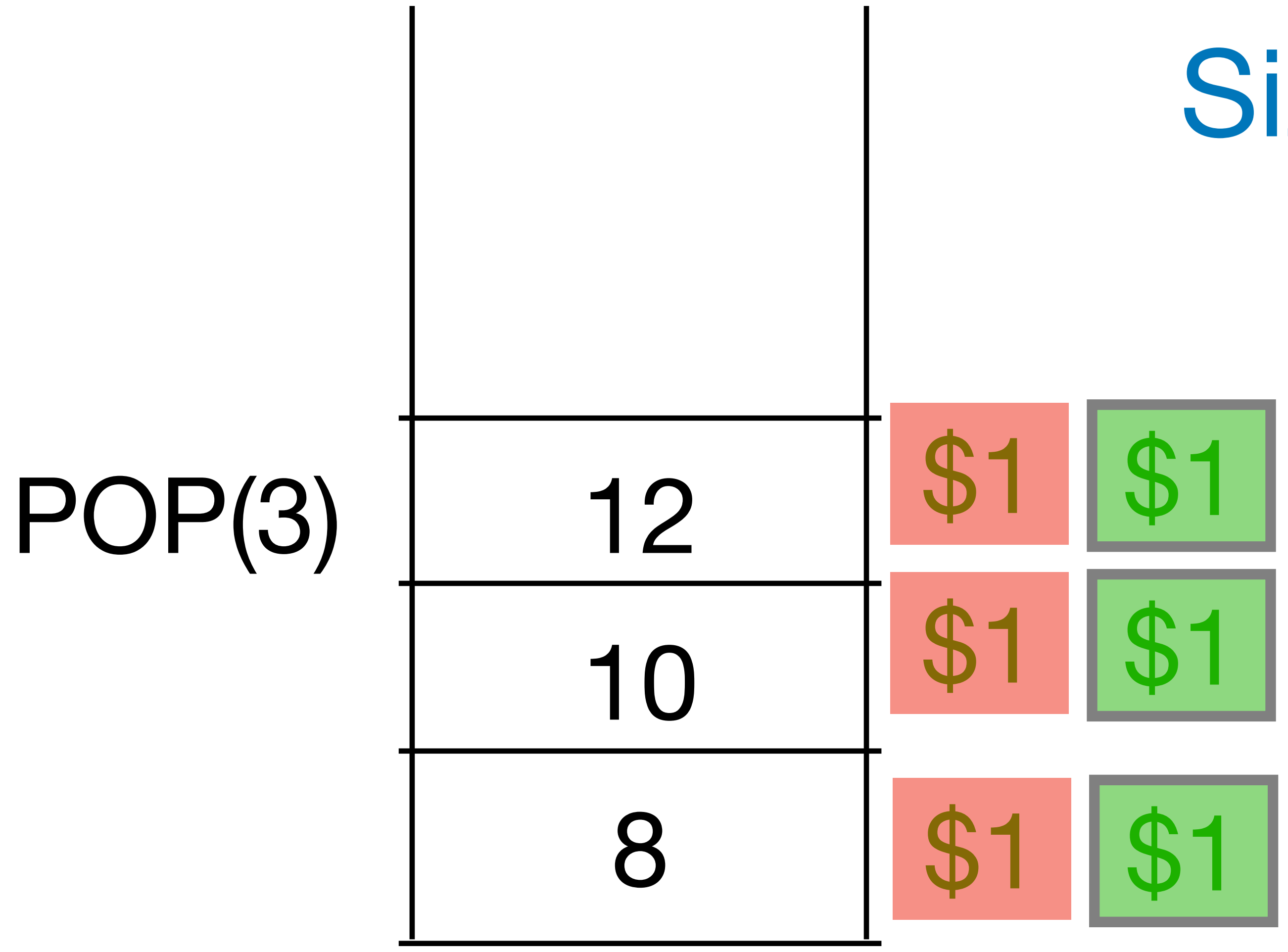
Amortized Cost: 0

Is it true that $\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$?

Technique 2: Accounting Method

Example: Stack Operations

Size of Stack: $|S| = 0$



How to cover the real cost?
Use the over-paid dollars.

Total real cost: 6
Total amortized cost: 6

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Operations:

1) PUSH: push a number into stack

Cost: 1   Amortized Cost: 2

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

Cost: $\min\{k, |S|\}$

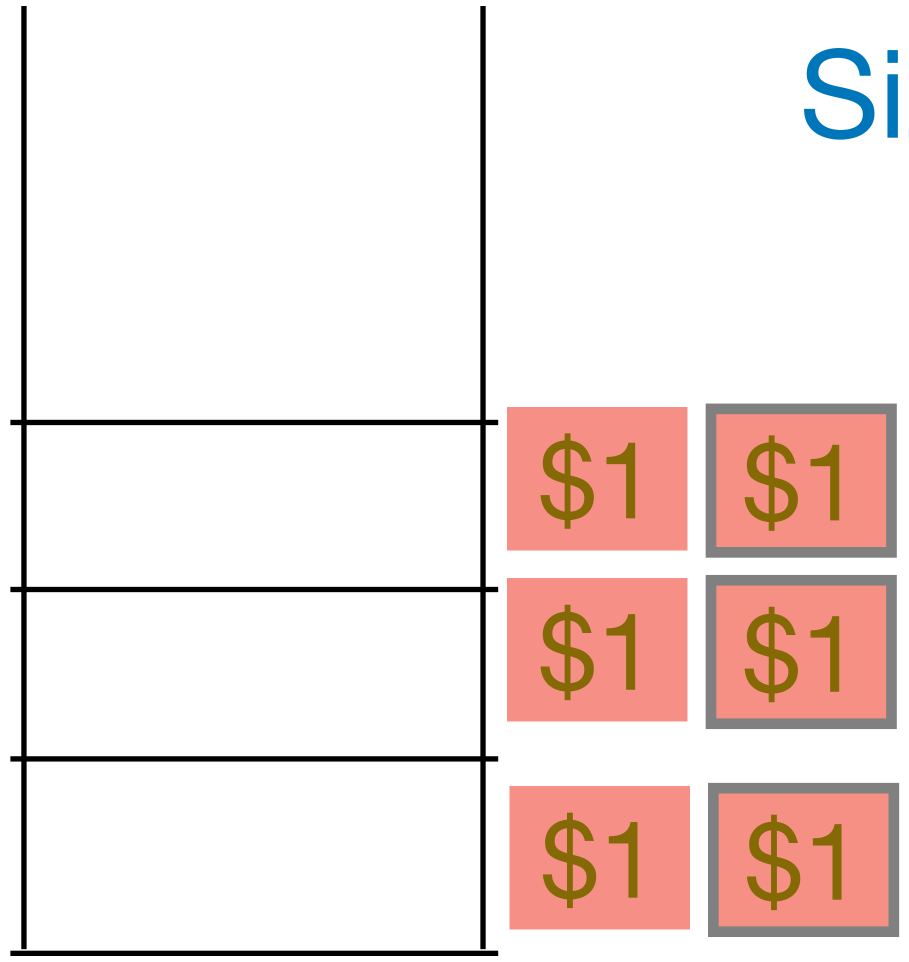
Amortized Cost: 0

Is it true that $\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$?

Technique 2: Accounting Method

Example: Stack Operations

POP(3)



Size of Stack: $|S| = 0$

How to cover the real cost?
Use the over-paid dollars.

Idea: Pay In Advance.

Total real cost: 6
Total amortized cost: 6

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ?

Operations:

1) PUSH: push a number into stack

Cost: 1 Amortized Cost: 2

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

Cost: $\min\{k, |S|\}$

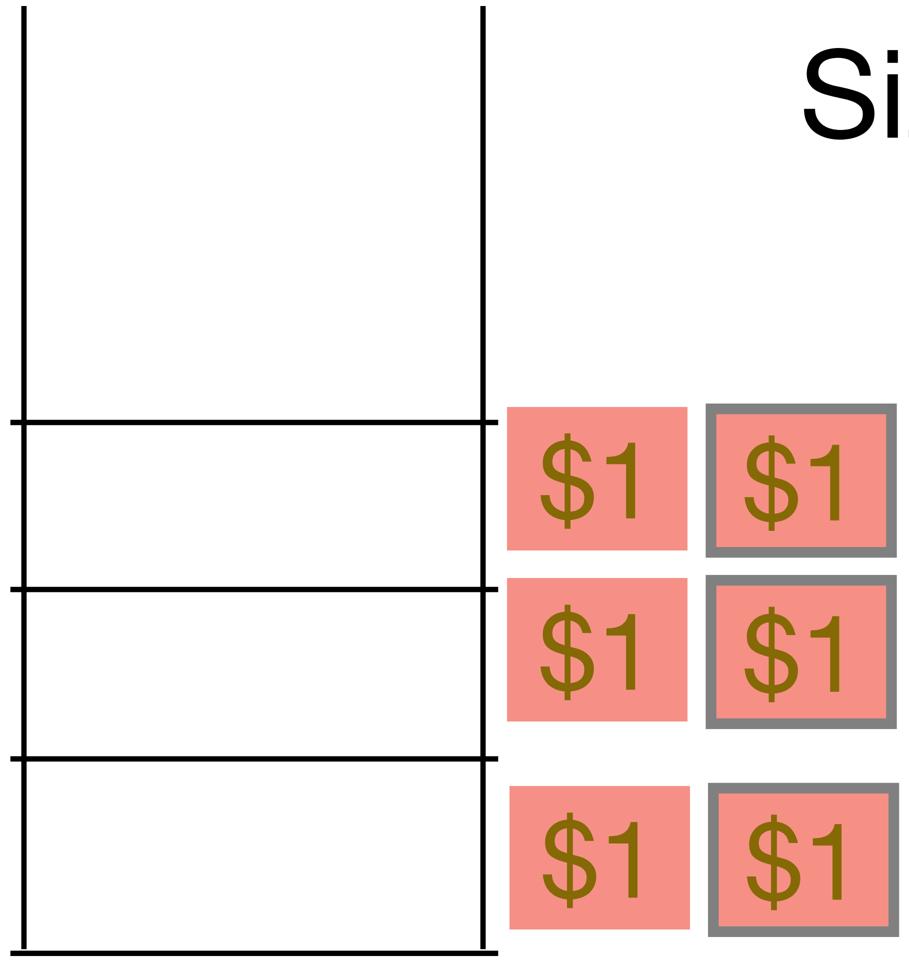
Amortized Cost: 0

Is it true that $\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$?

Technique 2: Accounting Method

Example: Stack Operations

POP(3)



Size of Stack: $|S| = 0$

How to cover the real cost?
Use the over-paid dollars.
Idea: Pay In Advance.

Total real cost: 6
Total amortized cost: 6

Consider a sequence of n stack operations.
What is a tight upper bound on its total cost ? $O(n)$

Operations:

1) PUSH: push a number into stack

   Cost: 1 Amortized Cost: 2

2) POP(k): pop out the top k numbers from stack. If the stack has fewer than k numbers, we pop out all numbers in stack.

   Cost: $\min\{k, |S|\}$

   Amortized Cost: 0

2n >= $\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$

Quiz questions:

1. Why do we want to “pay in advance” for the amortized cost of pop operations?

2. How does the above “pay in advance” method help us analyze the total cost?

Roadmap of this lecture:

1. Define "Amortized Analysis".

2. Amortized analysis by the "Aggregate Analysis" technique.

   2.1 Understand "Aggregate Analysis" through the example of "Stack Operations".

   2.2 Understand "Aggregate Analysis" through the example of "Counter Incrementation".

3. Amortized analysis by the "Accounting Method" technique.

   2.1 Understand "Accounting Method" through the example of "Stack Operations".

   2.2 Understand "Accounting Method" through the example of "Counter Incrementation".

Technique 2: Accounting Method

Large Binary Counter

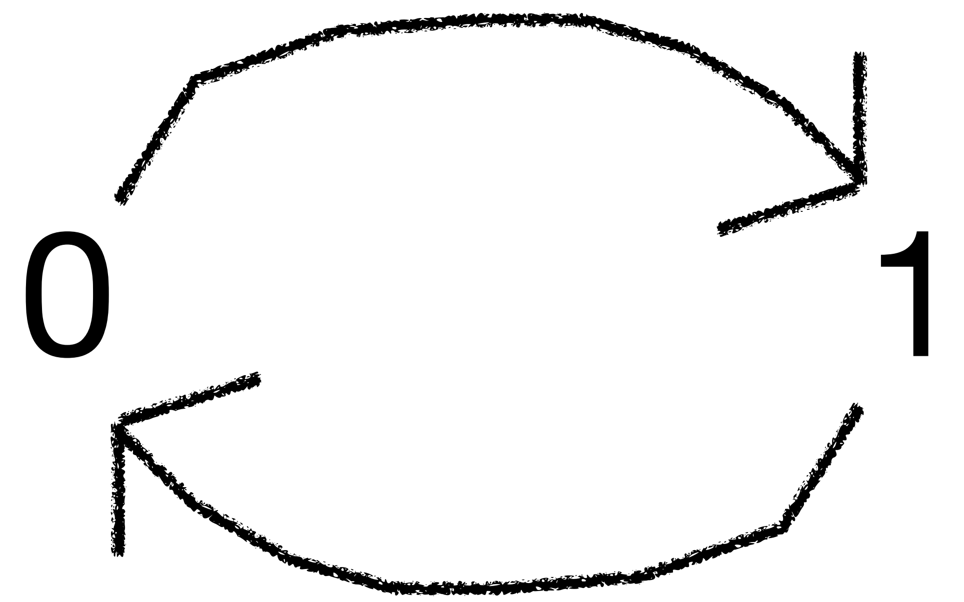
0	0 0 0 0 0 0 0 0 0 0	
		cost: 1
1	0 0 0 0 0 0 0 0 0 1	
		cost: 2
2	0 0 0 0 0 0 0 0 1 0	
		cost: 1
3	0 0 0 0 0 0 0 0 1 1	
		cost: 3
4	0 0 0 0 0 0 0 1 0 0	
		cost: 1
5	0 0 0 0 0 0 0 1 0 1	
		cost: 2
6	0 0 0 0 0 0 0 1 1 0	
		cost: 1
7	0 0 0 0 0 0 0 1 1 1	
		cost: 4
8	0 0 0 0 0 0 1 0 0 0	
		cost: 1
9	0 0 0 0 0 0 1 0 0 1	
		cost: 2
10	0 0 0 0 0 0 1 0 1 0	
		cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Cost of incrementing counter:
Number of bits that are changed.

What is the total cost of n increments?

Technique 2: Accounting Method

Lifecycle of a bit:



Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	cost: 1
9	0 0 0 0 0 0 1 0 0 1	cost: 2
10	0 0 0 0 0 0 1 0 1 0	cost: 1
11	0 0 0 0 0 0 1 0 1 1	

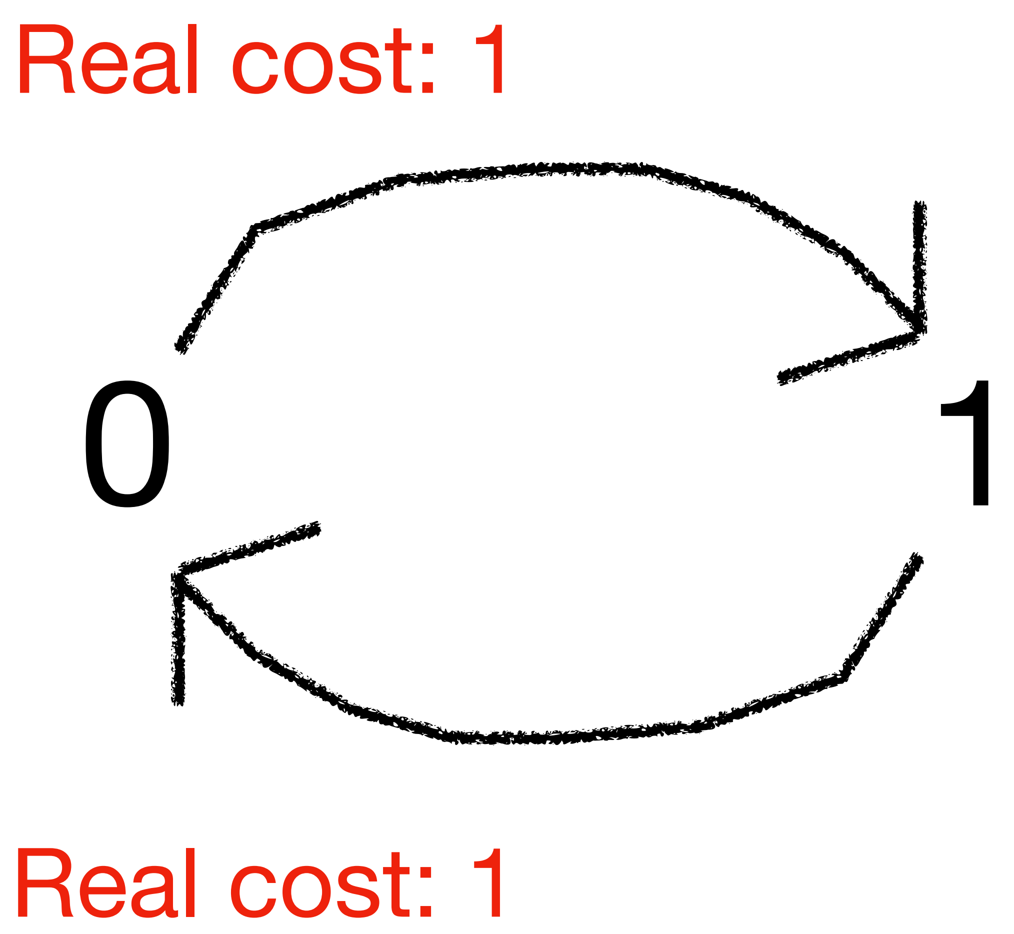
Cost of incrementing counter:

Number of bits that are changed.

What is the total cost of n increments?

Technique 2: Accounting Method

Lifecycle of a bit:



Large Binary Counter

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	cost: 1
9	0 0 0 0 0 0 1 0 0 1	cost: 2
10	0 0 0 0 0 0 1 0 1 0	cost: 1
11	0 0 0 0 0 0 1 0 1 1	

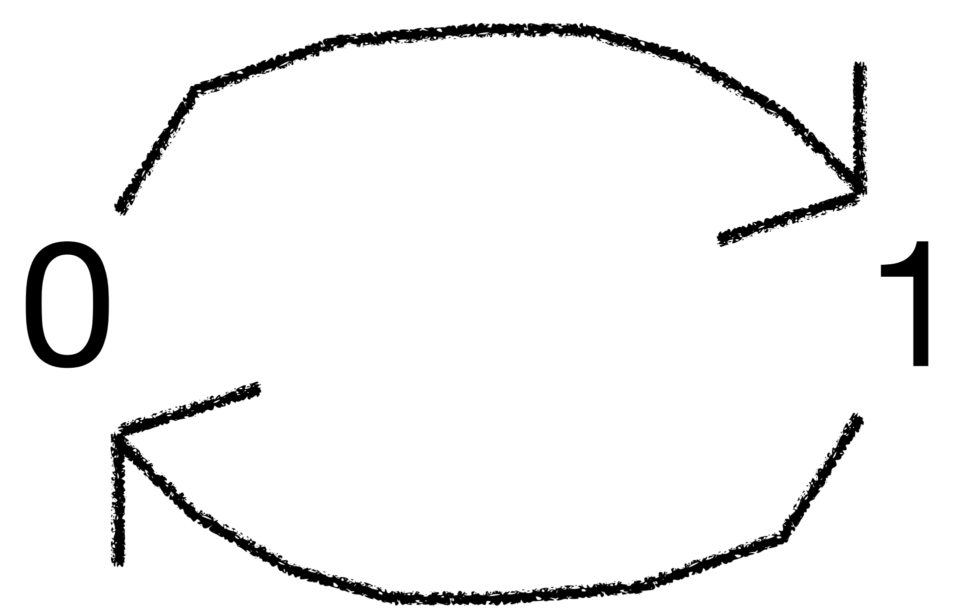
Cost of incrementing counter:
Number of bits that are changed.

What is the total cost of n increments?

Technique 2: Accounting Method

Lifecycle of a bit:

Real cost: 1 Amortized cost: 2



Real cost: 1 Amortized cost: 0

Since the amortized cost “pre-pays” the total cost of every lifecycle,

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Large Binary Counter

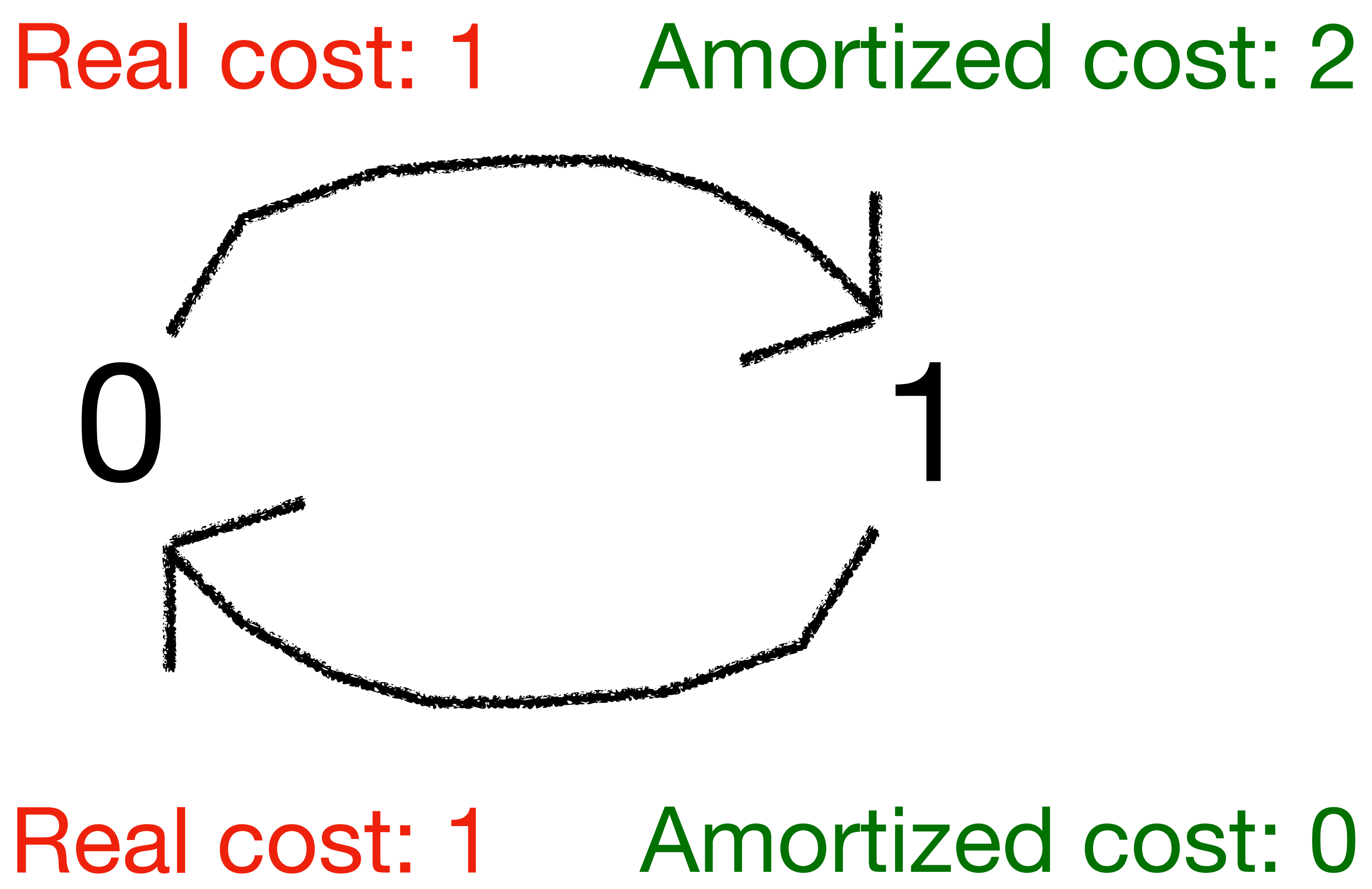
0	0 0 0 0 0 0 0 0 0 0	
		cost: 1
1	0 0 0 0 0 0 0 0 0 1	
		cost: 2
2	0 0 0 0 0 0 0 0 1 0	
		cost: 1
3	0 0 0 0 0 0 0 0 1 1	
		cost: 3
4	0 0 0 0 0 0 0 1 0 0	
		cost: 1
5	0 0 0 0 0 0 0 1 0 1	
		cost: 2
6	0 0 0 0 0 0 0 1 1 0	
		cost: 1
7	0 0 0 0 0 0 0 1 1 1	
		cost: 4
8	0 0 0 0 0 0 1 0 0 0	
		cost: 1
9	0 0 0 0 0 0 1 0 0 1	
		cost: 2
10	0 0 0 0 0 0 1 0 1 0	
		cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Cost of incrementing counter:
Number of bits that are changed.

What is the total cost of n increments?

Technique 2: Accounting Method

Lifecycle of a bit:



Since the amortized cost “pre-pays” the total cost of every lifecycle,

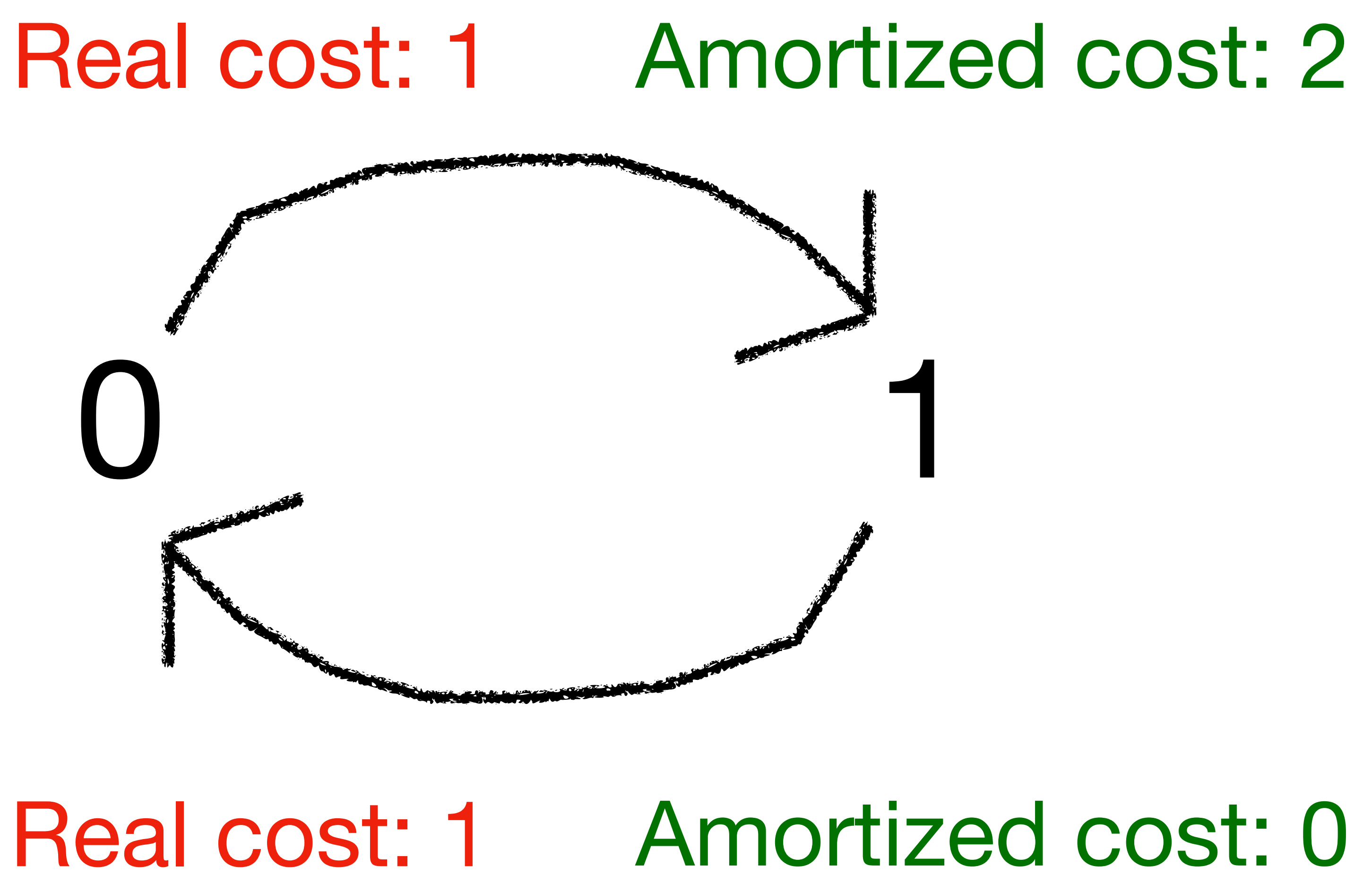
$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Amortized cost: 2

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	cost: 1
9	0 0 0 0 0 0 1 0 0 1	cost: 2
10	0 0 0 0 0 0 1 0 1 0	cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Technique 2: Accounting Method

Lifecycle of a bit:



Since the amortized cost "pre-pays" the total cost of every lifecycle,

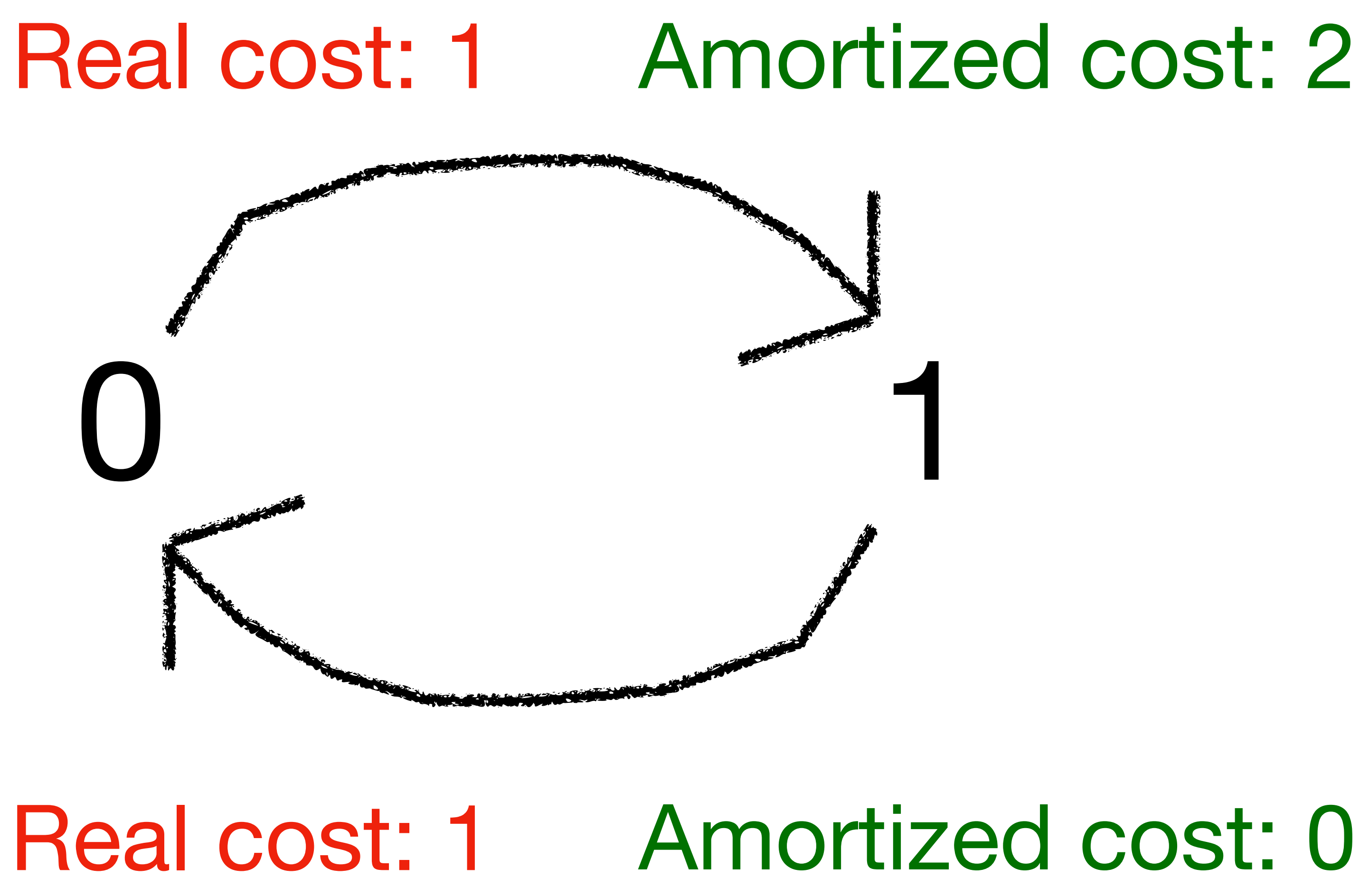
$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Amortized cost: 2

0	0 0 0 0 0 0 0 0 0 0	
		cost: 1
1	0 0 0 0 0 0 0 0 0 1	
		cost: 2
2	0 0 0 0 0 0 0 0 1 0	
		cost: 1
3	0 0 0 0 0 0 0 0 1 1	
		cost: 3
4	0 0 0 0 0 0 0 1 0 0	
		cost: 1
5	0 0 0 0 0 0 0 1 0 1	
		cost: 2
6	0 0 0 0 0 0 0 1 1 0	
		cost: 1
7	0 0 0 0 0 0 0 1 1 1	
		cost: 4
8	0 0 0 0 0 0 1 0 0 0	
		cost: 1
9	0 0 0 0 0 0 1 0 0 1	
		cost: 2
10	0 0 0 0 0 0 1 0 1 0	
		cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Technique 2: Accounting Method

Lifecycle of a bit:



Since the amortized cost "pre-pays" the total cost of every lifecycle,

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Amortized cost: 2

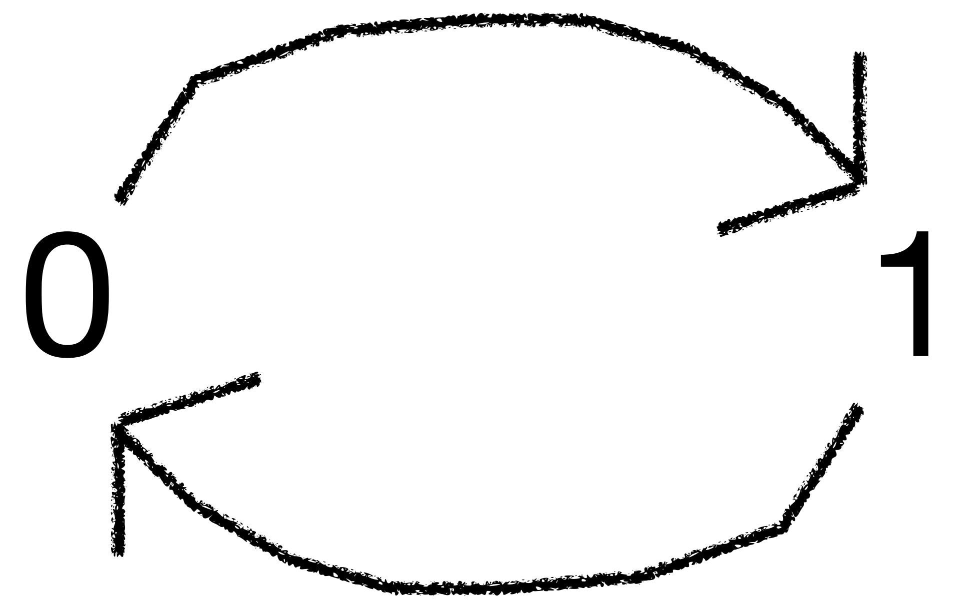
0	0 0 0 0 0 0 0 0 0 0	
		cost: 1
1	0 0 0 0 0 0 0 0 0 1	
		cost: 2
2	0 0 0 0 0 0 0 0 1 0	
		cost: 1
3	0 0 0 0 0 0 0 0 1 1	
		cost: 3
4	0 0 0 0 0 0 0 1 0 0	
		cost: 1
5	0 0 0 0 0 0 0 1 0 1	
		cost: 2
6	0 0 0 0 0 0 0 1 1 0	
		cost: 1
7	0 0 0 0 0 0 0 1 1 1	
		cost: 4
8	0 0 0 0 0 0 1 0 0 0	
		cost: 1
9	0 0 0 0 0 0 1 0 0 1	
		cost: 2
10	0 0 0 0 0 0 1 0 1 0	
		cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Technique 2: Accounting Method

Amortized cost: 2

Lifecycle of a bit:

Real cost: 1 Amortized cost: 2



Real cost: 1 Amortized cost: 0

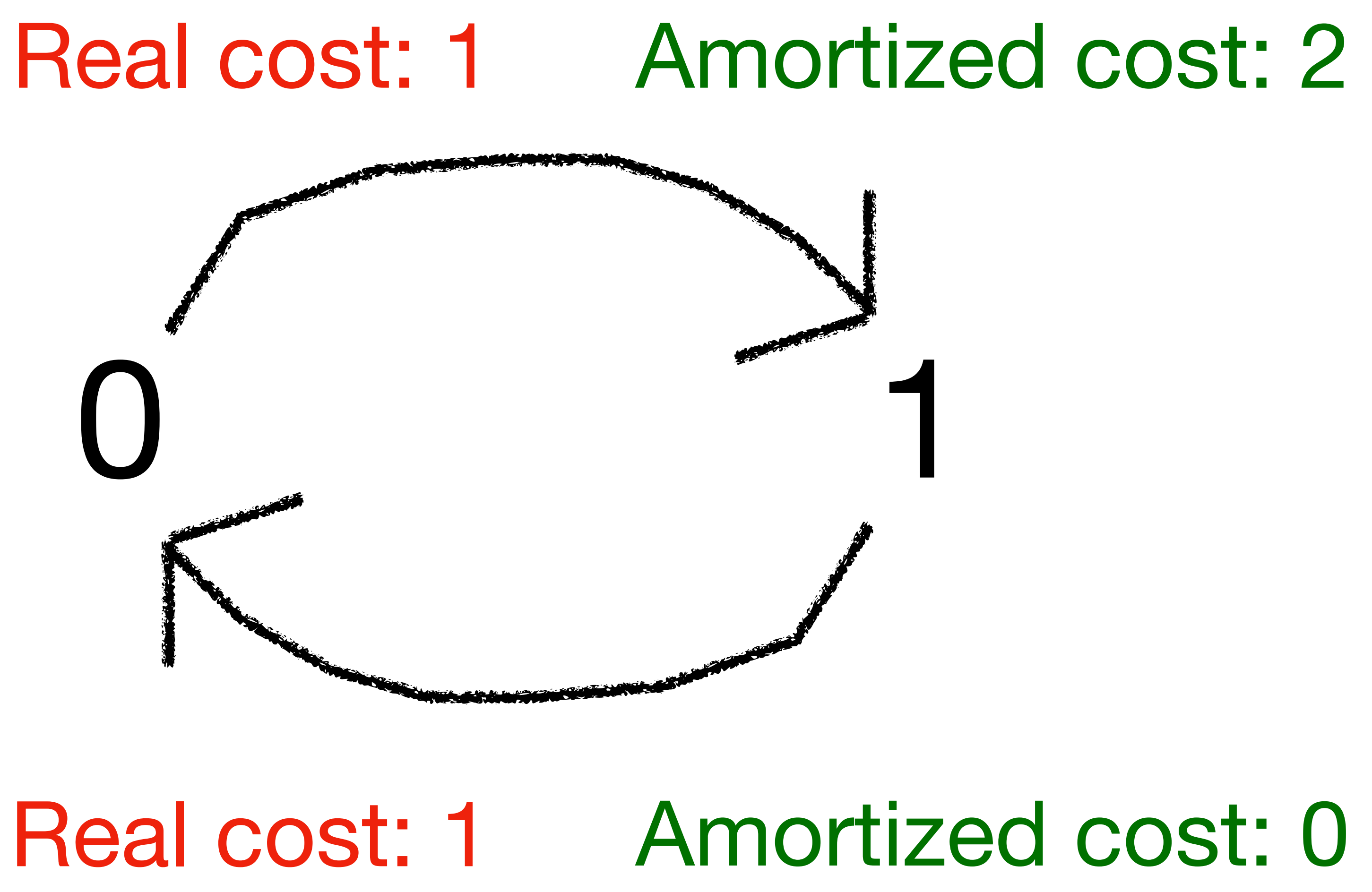
Since the amortized cost "pre-pays" the total cost of every lifecycle,

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	cost: 1
9	0 0 0 0 0 0 1 0 0 1	cost: 2
10	0 0 0 0 0 0 1 0 1 0	cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Technique 2: Accounting Method

Lifecycle of a bit:



Since the amortized cost "pre-pays" the total cost of every lifecycle,

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

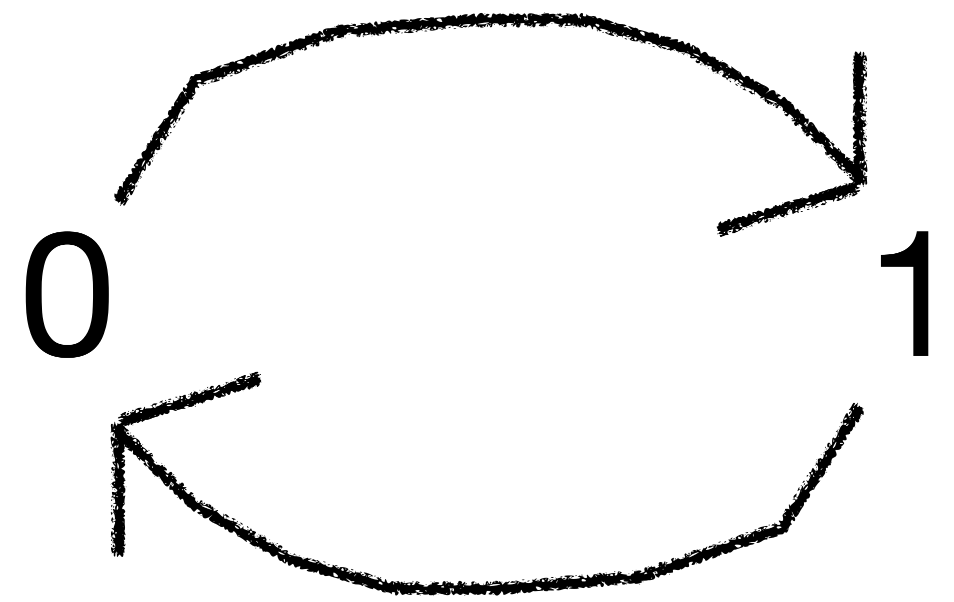
Amortized cost: 2

0	0 0 0 0 0 0 0 0 0 0	
		cost: 1
1	0 0 0 0 0 0 0 0 0 1	
		cost: 2
2	0 0 0 0 0 0 0 0 1 0	
		cost: 1
3	0 0 0 0 0 0 0 0 1 1	
		cost: 3
4	0 0 0 0 0 0 0 1 0 0	
		cost: 1
5	0 0 0 0 0 0 0 1 0 1	
		cost: 2
6	0 0 0 0 0 0 0 1 1 0	
		cost: 1
7	0 0 0 0 0 0 0 1 1 1	
		cost: 4
8	0 0 0 0 0 0 1 0 0 0	
		cost: 1
9	0 0 0 0 0 0 1 0 0 1	
		cost: 2
10	0 0 0 0 0 0 1 0 1 0	
		cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Technique 2: Accounting Method

Lifecycle of a bit:

Real cost: 1 Amortized cost: 2



Real cost: 1 Amortized cost: 0

Since the amortized cost "pre-pays" the total cost of every lifecycle,

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

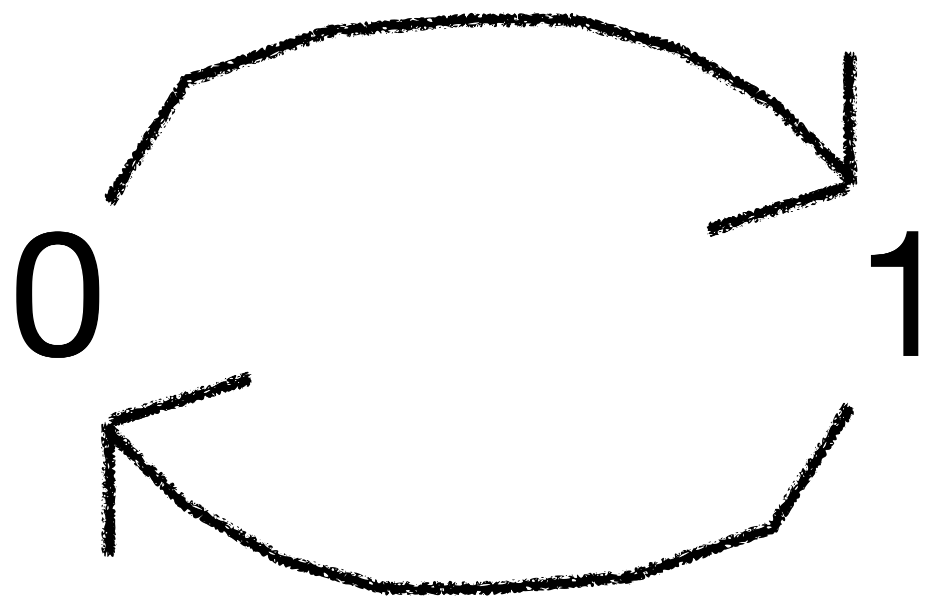
Amortized cost: 2

0	0 0 0 0 0 0 0 0 0 0	
		cost: 1
1	0 0 0 0 0 0 0 0 0 1	
		cost: 2
2	0 0 0 0 0 0 0 0 1 0	
		cost: 1
3	0 0 0 0 0 0 0 0 1 1	
		cost: 3
4	0 0 0 0 0 0 0 1 0 0	
		cost: 1
5	0 0 0 0 0 0 0 1 0 1	
		cost: 2
6	0 0 0 0 0 0 0 1 1 0	
		cost: 1
7	0 0 0 0 0 0 0 1 1 1	
		cost: 4
8	0 0 0 0 0 0 1 0 0 0	
		cost: 1
9	0 0 0 0 0 0 1 0 0 1	
		cost: 2
10	0 0 0 0 0 0 1 0 1 0	
		cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Technique 2: Accounting Method

Lifecycle of a bit:

Real cost: 1 Amortized cost: 2



Real cost: 1 Amortized cost: 0

Since the amortized cost "pre-pays" the total cost of every lifecycle,

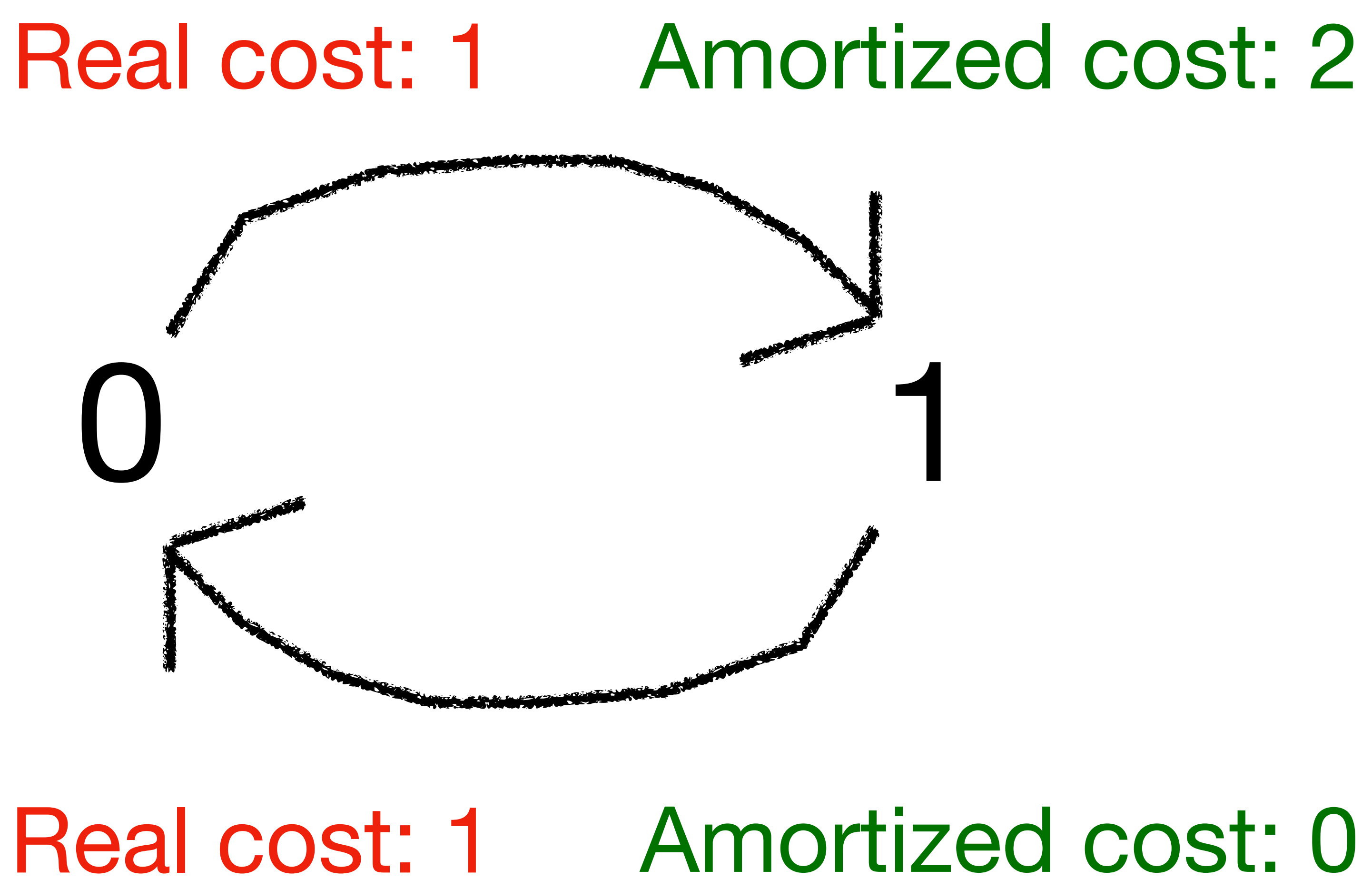
$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Amortized cost: 2

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	cost: 1
9	0 0 0 0 0 0 1 0 0 1	cost: 2
10	0 0 0 0 0 0 1 0 1 0	cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Technique 2: Accounting Method

Lifecycle of a bit:



Since the amortized cost "pre-pays" the total cost of every lifecycle,

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

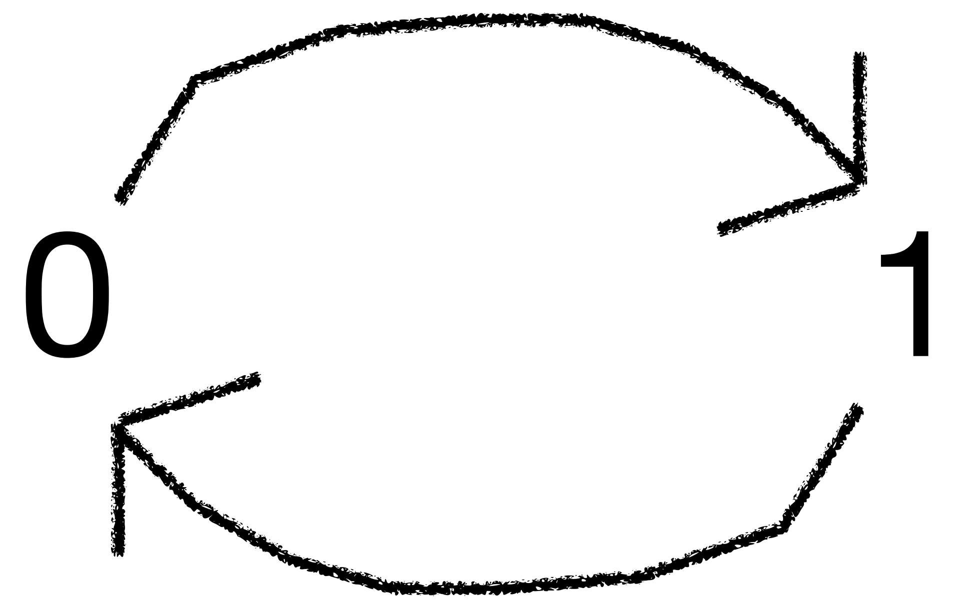
Amortized cost: 2

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	cost: 1
9	0 0 0 0 0 0 1 0 0 1	cost: 2
10	0 0 0 0 0 0 1 0 1 0	cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Technique 2: Accounting Method

Lifecycle of a bit:

Real cost: 1 Amortized cost: 2



Real cost: 1 Amortized cost: 0

Since the amortized cost "pre-pays" the total cost of every lifecycle,

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Amortized cost: 2

0	0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 0 0 1 1
4	0 0 0 0 0 0 0 1 0 0
5	0 0 0 0 0 0 0 1 0 1
6	0 0 0 0 0 0 0 1 1 0
7	0 0 0 0 0 0 0 1 1 1
8	0 0 0 0 0 0 1 0 0 0
9	0 0 0 0 0 0 1 0 0 1
10	0 0 0 0 0 0 1 0 1 0
11	0 0 0 0 0 0 1 0 1 1

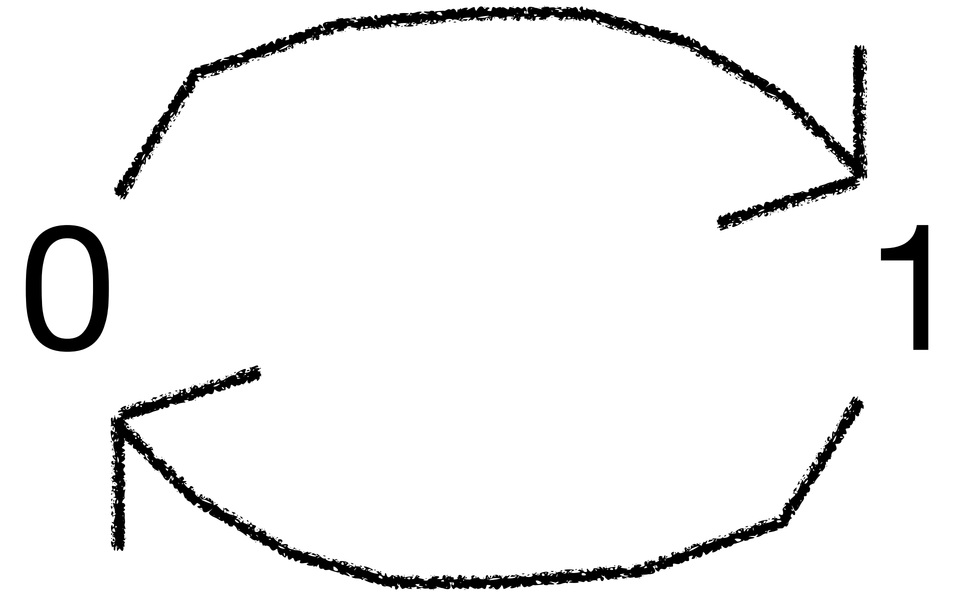
cost: 1
cost: 2
cost: 1
cost: 3
cost: 1
cost: 2
cost: 1
cost: 4
cost: 1
cost: 2
cost: 1

Technique 2: Accounting Method

Amortized cost: 2

Lifecycle of a bit:

Real cost: 1 Amortized cost: 2



Real cost: 1 Amortized cost: 0

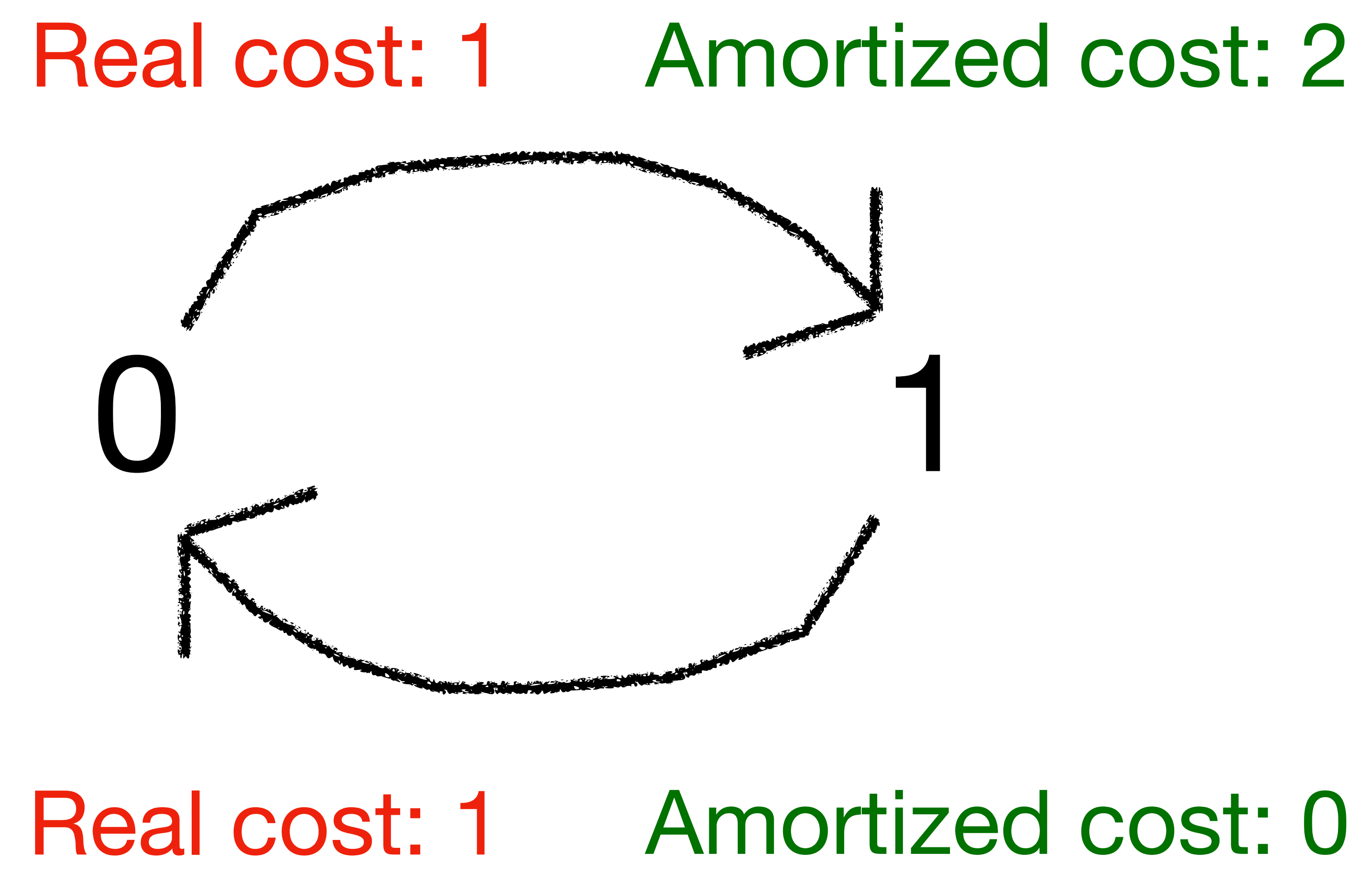
0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	cost: 1
9	0 0 0 0 0 0 1 0 0 1	cost: 2
10	0 0 0 0 0 0 1 0 1 0	cost: 1
11	0 0 0 0 0 0 1 0 1 1	

Since the amortized cost "pre-pays" the total cost of every lifecycle,

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Technique 2: Accounting Method

Lifecycle of a bit:



Since the amortized cost "pre-pays" the total cost of every lifecycle,

$$\sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Amortized cost: 2

0	0 0 0 0 0 0 0 0 0 0	cost: 1
1	0 0 0 0 0 0 0 0 0 1	cost: 2
2	0 0 0 0 0 0 0 0 1 0	cost: 1
3	0 0 0 0 0 0 0 0 1 1	cost: 3
4	0 0 0 0 0 0 0 1 0 0	cost: 1
5	0 0 0 0 0 0 0 1 0 1	cost: 2
6	0 0 0 0 0 0 0 1 1 0	cost: 1
7	0 0 0 0 0 0 0 1 1 1	cost: 4
8	0 0 0 0 0 0 1 0 0 0	cost: 1
9	0 0 0 0 0 0 1 0 0 1	cost: 2
10	0 0 0 0 0 0 1 0 1 0	cost: 1
11	0 0 0 0 0 0 1 0 1 1	

General pattern of change:

x x x x x 0 1 1 1 1

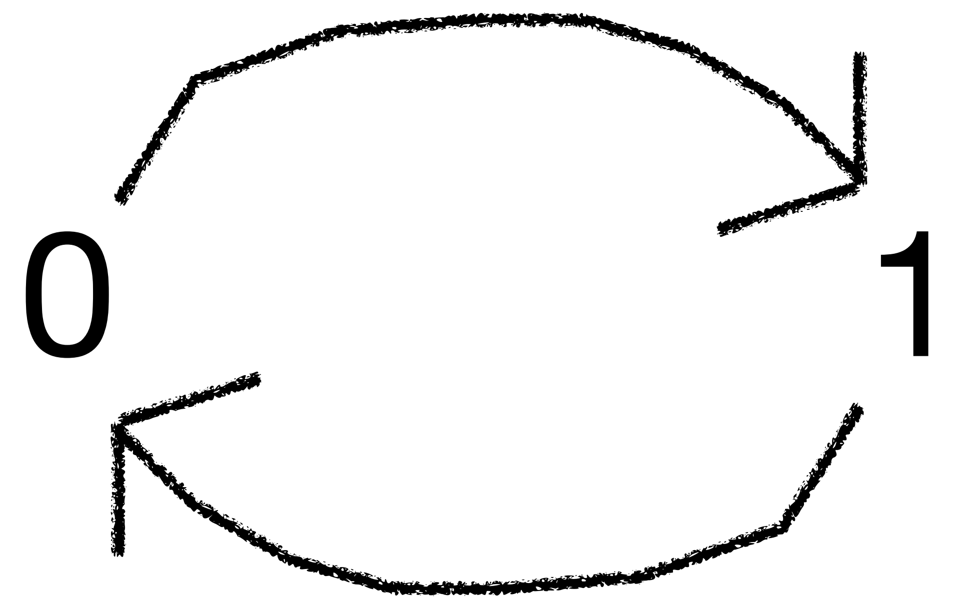
$\downarrow$

x x x x x 1 0 0 0 0

Technique 2: Accounting Method

Lifecycle of a bit:

Real cost: 1 Amortized cost: 2



Real cost: 1 Amortized cost: 0

Since the amortized cost "pre-pays" the total cost of every lifecycle,

$$2n >= \sum_{i=1}^{n} \hat{C}_i \geq \sum_{i=1}^{n} C_i$$

Amortized cost: 2

0	0 0 0 0 0 0 0 0 0 0	
1	0 0 0 0 0 0 0 0 0 1	cost: 1
2	0 0 0 0 0 0 0 0 1 0	cost: 2
3	0 0 0 0 0 0 0 0 1 1	cost: 1
4	0 0 0 0 0 0 0 1 0 0	cost: 3
5	0 0 0 0 0 0 0 1 0 1	cost: 1
6	0 0 0 0 0 0 0 1 1 0	cost: 2
7	0 0 0 0 0 0 0 1 1 1	cost: 1
8	0 0 0 0 0 0 1 0 0 0	cost: 4
9	0 0 0 0 0 0 1 0 0 1	cost: 1
10	0 0 0 0 0 0 1 0 1 0	cost: 2
11	0 0 0 0 0 0 1 0 1 1	cost: 1

total cost: $O(n)$

Quiz questions:

1. Why do we want to "pay in advance" for the amortized cost of "changing a bit from 1 to 0"?

2. How does the above "pay in advance" method help us analyze the total cost?