CS 577 - Data Structures & Amortized Analysis

Manolis Vlatakis

Department of Computer Sciences
University of Wisconsin – Madison

Fall 2024

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Data structures

Static problems

Given an input, produce an output.

Examples

Sorting, Compute Fourier Transform, shortest paths, …

Data structures

Static problems

Given an input, produce an output.

Examples

Sorting, Compute Fourier Transform, shortest paths, …

Dynamic problems

Given a sequence of operations (given one at a time), produce a sequence of outputs.

Examples

Dynamic Median Maintenance of a growing list,
Incremental Convex Hull of a online updated set of points, …

Data structures

Algorithm Vs Data structure

Algorithm. Step-by-step procedure to solve a problem.
Data structure. Way to store and organize data.

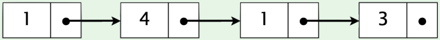
Data structures

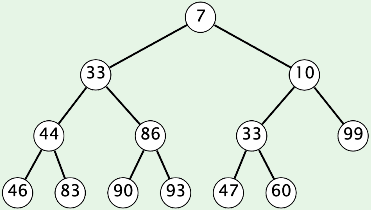
Algorithm Vs Data structure

Algorithm. Step-by-step procedure to solve a problem.
Data structure. Way to store and organize data.

🤔What data structures have you learned about in your courses so far?

Data structures

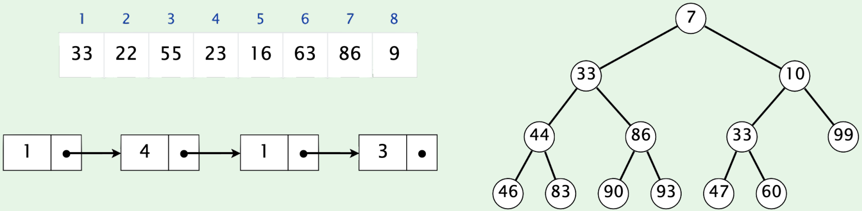




Ex. Array, linked list, binary search tree, hash table, …

What data structures have you learned about in your courses so far?

D



Ex. Array, linked list, binary search tree, hash table, …

🤔What data structures have you learned about in your courses so far?

Let's see an example of a Data Structure Problem.

Data Structure Problems

Appetizer.
Efficiently null-initialized array

Goal. Design a data structure to support **all** operations in $O(1)$ time.

- **INIT**(n)**:** create and return an initialized array (all zero) of length n.
- **READ**(A, i)**:** return element i in array.
- **WRITE**$(A, i, \textbf{value})$**:** set element i in array to value.

Assumptions.

- Can **MALLOC** an uninitialized array of length n in $O(1)$ time. true in C or C++, but not Java
- Given an array, can read or write element i in $O(1)$ time.

APPETIZER.
EFFICIENTLY NULL-INITIALIZED ARRAY

Goal. Design a data structure to support **all** operations in $O(1)$ time.

- **INIT**(n)**:** create and return an initialized array (all zero) of length n.
- **READ**(A, i)**:** return element i in array.
- **WRITE**$(A, i, \textbf{value})$**:** set element i in array to value.

Assumptions.

- Can **MALLOC** an uninitialized array of length n in $O(1)$ time.
  true in C or C++, but not Java

Remark. A standard array does $\begin{cases} \textbf{INIT} \text{ in } \Theta(n) \text{ time} \\ \textbf{READ} \text{ and } \textbf{WRITE} \text{ in } \Theta(1) \text{ time} \end{cases}$

APPETIZER.
EFFICIENTLY NULL-INITIALIZED ARRAY

Goal. Design a data structure to support **all** operations in $O(1)$ time.

- **INIT**(n)**:** create and return an initialized array (all zero) of length n.
- **READ**(A, i)**:** return element i in array.
- **WRITE**$(A, i, \textbf{value})$**:** set element i in array to value.

How can we build such a structure using standard arrays like Lego blocks?

A

- Can MALLOC an uninitialized array of length n in $O(1)$ time.
  true in C or C++, but not Java

Remark. A standard array does $\begin{cases} \textbf{INIT} \text{ in } \Theta(n) \text{ time} \\ \textbf{READ} \text{ and } \textbf{WRITE} \text{ in } \Theta(1) \text{ time} \end{cases}$

APPETIZER. EFFICIENTLY NULL-INITIALIZED ARRAY

INIT (A, n)

READ (A, i)

WRITE $(A, i, value)$

IS-INITIALIZED (A, i)

Appetizer. Efficiently null-initialized array

Init (A, n)

Read (A, i)

Write $(A, i, value)$

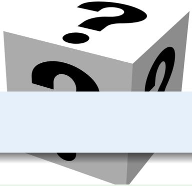
🤔 What do we obviously need?

APPETIZER.
EFFICIENTLY NULL-INITIALIZED ARRAY

INIT (A, n)

READ (A, i)

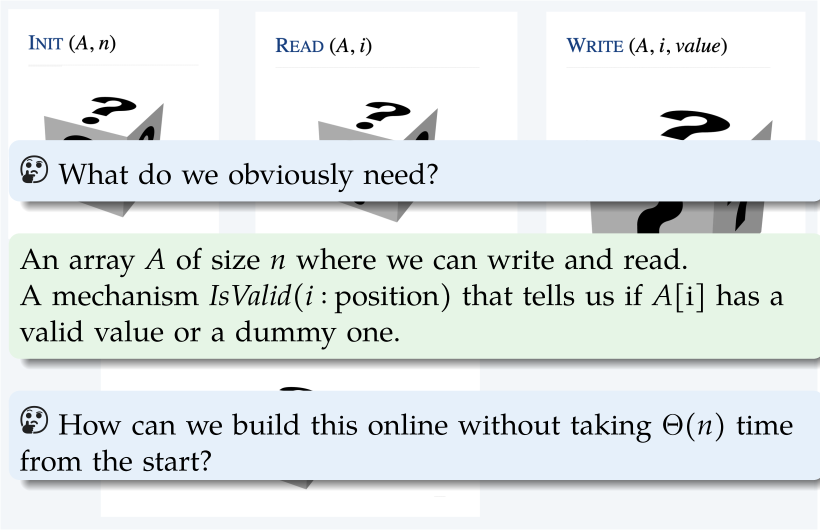
WRITE $(A, i, value)$



🤔 What do we obviously need?

An array A of size n where we can write and read.
A mechanism $IsValid(i : \text{position})$ that tells us if $A[\text{i}]$ has a valid value or a dummy one.

Appetizer.
Efficiently null-initialized array



🤔 What do we obviously need?

An array A of size n where we can write and read.
A mechanism $IsValid(i : \text{position})$ that tells us if $A[\text{i}]$ has a valid value or a dummy one.

🤔 How can we build this online without taking $\Theta(n)$ time from the start?

Efficiently Null-Initialized Array Implementation (Effort #1)

Initialization Stamp

Init(A, n)

```
k ← 0.
A ← Malloc(n).
B ← Malloc(n).
```

Read(A, i)

```
If (Is-Valid(A[i]))
  Return A[i].
Else
  Return 0.
```

Write(A, i, value)

```
If (Is-Valid(A[i]))
  A[i] ← value.
Else
  k ← k + 1.
  A[i] ← value.
  B[i] ← k.
```

Is-Valid(A, i)

```
If (1 ≤ B[i] ≤ k)
  Return true.
Else
  Return false.
```

Efficiently Null-Initialized Array

Im

Init

i. `k` keeps track of how many cells have been safely modified.
ii. `A[:]` is used to store and retrieve the values.
iii. `B[i]` indicates when `A[i]` was initially modified.

Init(A, n)

```
k ← 0.
A ← Malloc(n).
B ← Malloc(n).
```

Read(A, i)

```
If (Is-Valid(A[i]))
  Return A[i].
Else
  Return 0.
```

Write(A, i, value)

```
If (Is-Valid(A[i]))
  A[i] ← value.
Else
  k ← k + 1.
  A[i] ← value.
  B[i] ← k.
```

Is-Valid(A, i)

```
If (1 ≤ B[i] ≤ k)
  Return true.
Else
  Return false.
```

Efficiently Null-Initialized Array

i. `k` keeps track of how many cells have been safely modified.
ii. `A[:]` is used to store and retrieve the values.
iii. `B[i]` indicates when `A[i]` was initially modified.

Write(A,4,99), Write(A,6,33), Write(A,2,22), Write(A,3,55), Read(A,5)

	1	2	3	4	5	6	7	8
A[ ]	?	22	55	99	?	33	?	?

k = 4

	1	2	3	4	5	6	7	8
B[ ]	?	3	4	1	?	2	?	?

What if $1 \leq$ `B[5]` ≤ 4 is true by coincidence?

i. `k` keeps track of how many cells have been safely modified.
ii. `A[:]` is used to store and retrieve the values.
iii. `B[i]` indicates when `A[i]` was initially modified.
iv. `C[j]` = index of j-th initialized element

E
IM

IN

```
k ← 0.
A ← MALL
B ← MALL
C ← MALL
```

(A[i]))
lue.

.
lue.

IS-VAL

```
If (1
  Retu
Else
  Retu
```

Write(A,4,99), Write(A,6,33), Write(A,2,22), Write(A,3,55), Read(A,5)

	1	2	3	4	5	6	7	8
A[ ]	?	22	55	99	?	33	?	?
B[ ]	?	3	4	1	?	2	?	?
C[ ]	4	6	2	3	?	?	?	?

k = 4

What if $1 \le$ `B[5]` ≤ 4 is true by coincidence?

Efficiently Null-Initialized Array Implementation (Effort #2)

`If` $(1 \le$ `B[i]` $\le$ `k`), then we still can't have (`C[B[i]] = i`) because `C[1..k]` $\neq$ `i` if A[`i`] not initialized. (Induction at k ☺)

```
k ← 0.
A ← Mall
B ← Mall
C ← Mall
```

`(A[i]))`
`lue.`
`.`
`lue.`

Is-Val

```
If (1
  Retu
Else
  Retu
```

🤔Write(A,4,99), Write(A,6,33), Write(A,2,22), Write(A,3,55), Read(A,5)

	1	2	3	4	5	6	7	8
A[ ]	?	22	55	99	?	33	?	?
B[ ]	?	3	4	1	?	2	?	?
C[ ]	4	6	2	3	?	?	?	?

k = 4

What if $1 \le$ `B[5]` ≤ 4 is true by coincidence?

Efficiently Null-Initialized Array Implementation (Effort #2)

Init(A, n)

```
k ← 0.
A ← Malloc(n).
B ← Malloc(n).
C ← Malloc(n).
```

Read(A, i)

```
If (Is-Valid(A[i]))
  Return A[i].
Else
  Return 0.
```

Write(A, i, value)

```
If (Is-Valid(A[i]))
  A[i] ← value.
Else
  k ← k + 1.
  A[i] ← value.
  B[i] ← k.
  C[k] ← i.
```

Is-Valid(A, i)

```
If (1 ≤ B[i] ≤ k) and (C[B[i]] = i)
  Return true.
Else
  Return false.
```

Amortized Analysis

Definition

Worst-case analysis

Determine worst-case running time of a data structure operation as a function of the input size n.

Pessimistic if expensive operations follow many cheap ones.

Amortized analysis

Determine worst-case running time of a sequence of n data structure operations.

$$Amortized - Cost = \frac{\sum_{i \in [n]} actual - cost[move_i]}{n}$$

Binary Counter

Goal

Increment n times a k-bit binary counter (mod 2^k) where k is super-large $k = \omega(2^{2^n})$.

$A[j] = j^{th}$ least significant bit of the counter.

Table: Binary Counter Table

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	1
6	0	0	0	0	0	1	1	0
7	0	0	0	0	0	1	1	1
8	0	0	0	0	1	0	0	0
9	0	0	0	0	1	0	0	1
10	0	0	0	0	1	0	1	0
11	0	0	0	0	1	0	1	1

Binary Counter

Cost model

Number of bits flipped.

🤔How many bit flips are needed to increment the counter n times, starting from zero?

Table: Binary Counter Table

Counter value	**A[7]**	**A[6]**	**A[5]**	**A[4]**	**A[3]**	**A[2]**	**A[1]**	**A[0]**
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	1
6	0	0	0	0	0	1	1	0
7	0	0	0	0	0	1	1	1
8	0	0	0	0	1	0	0	0
9	0	0	0	0	1	0	0	1
10	0	0	0	0	1	0	1	0
11	0	0	0	0	1	0	1	1

Binary Counter

Cost model

Number of bits flipped.

How many bit flips are needed to increment the counter n times, starting from zero?

In worst-case, at most k bits flipped per increment. So $O(nk)$.

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

Binary Counter

Cost model

Number of bits flipped.

🤔How many bit flips are needed to increment the counter n times, starting from zero?

In worst-case, at most k bits flipped per increment. So $O(n\dot{k})$.

🤔But k bits together would only flip during the last increment.

6	0	0	0	0	0	1	1	0
7	0	0	0	0	0	1	1	1
8	0	0	0	0	1	0	0	0
9	0	0	0	0	1	0	0	1
10	0	0	0	0	1	0	1	0
11	0	0	0	0	1	0	1	1

Amortized Analysis: Accounting Method

Intuition

Measure running time in terms of credits (time = money).

Amortized Analysis: Accounting Method

Intuition

Measure running time in terms of credits (time = money).

Key Idea

- Expensive operations: Prepay for these by storing up credits during cheaper operations.
- Cheap operations: Use the already stored credits (no extra cost at the time).

Amortized Analysis: Accounting Method

Intuition

Measure running time in terms of credits (time = money).

Key Idea

- Expensive operations: Prepay for these by storing up credits during cheaper operations.
- Cheap operations: Use the already stored credits (no extra cost at the time).

How It Works

- For every operation, we consume one credit.
- Assign each operation a cost that includes both the actual cost and some extra "savings."
- Use these "savings" to cover the cost of future cheap operations.

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

$$\#\,\text{operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\#\,\text{operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

$$\#\text{ operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\#\text{ operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Operation	Bank	Pocket
Expensive	5\$	

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

$$\#\text{ operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\#\text{ operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Operation	**Bank**	**Pocket**
Expensive	~~5\$~~	4\$

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

$$\#\text{ operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\#\text{ operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Operation	**Bank**	**Pocket**
Expensive	~~5\$~~	4\$
Cheap		

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

$$\#\text{ operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\#\text{ operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Operation	**Bank**	**Pocket**
Expensive	~~5\$~~	4\$
Cheap		3\$

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

$$\#\text{ operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\#\text{ operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Operation	**Bank**	**Pocket**
Expensive	~~5\$~~	4\$
Cheap		3\$
Cheap		

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

$$\text{\# operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\text{\# operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Operation	**Bank**	**Pocket**
Expensive	~~5\$~~	4\$
Cheap		3\$
Cheap		2\$

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs $1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow $5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only $1 if our pocket is empty.

At the end,

$$\#\text{ operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\#\text{ operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Operation	**Bank**	**Pocket**
Expensive	~~5$~~	4$
Cheap		3$
Cheap		2$
Expensive		

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

$$\#\text{ operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\#\text{ operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Operation	**Bank**	**Pocket**
Expensive	~~5\$~~	4\$
Cheap		3\$
Cheap		2\$
Expensive	5\$	

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

$$\#\text{ operations} = \underbrace{(\text{Dollars we consumed})}_{\text{actual cost}}$$

$$\#\text{ operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}.$$

Operation	**Bank**	**Pocket**
Expensive	~~5\$~~	4\$
Cheap		3\$
Cheap		2\$
Expensive	~~5\$~~	6\$

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

- For expensive operations, we borrow \$5. The leftover money stays in our pocket for future use.
- For cheap operations, we borrow only \$1 if our pocket is empty.

At the end,

Operation	**Bank**	**Pocket**
Expensive	~~5\$~~	4\$
Cheap		3\$
Cheap		2\$
Expensive	~~5\$~~	6\$

$$\#\text{ operations} = \underbrace{(\text{Dollars we borrowed})}_{10\$} - \underbrace{(\text{Dollars left in our pocket})}_{6\$} = 4.$$

Amortized Analysis: Accounting Method

An Example is Worth a Thousand Words

Every operation costs \$1 to execute. If we don't have money in our pocket, we borrow from the bank.

Aesop's Moral

- Bank: Our worst-case outlook on expensive operations.
- Pocket: Our amortized hope that costly operations are rare.

At the end,

Operation	Bank	Pocket
Expensive	~~5\$~~	4\$
Cheap		3\$
Cheap		2\$
Expensive	~~5\$~~	6\$

$$\#\,\text{operations} = \underbrace{(\text{Dollars we borrowed})}_{10\$} - \underbrace{(\text{Dollars left in our pocket})}_{6\$} = 4.$$

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is \$1. We always borrow \$2 from the bank.

- If we don't have enough money in our pocket, we borrow \$1 per necessary flip.
- Any leftover money stays in our pocket for future use.

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is $1. We always borrow $2 from the bank.

- If we don't have enough money in our pocket, we borrow $1 per necessary flip.
- Any leftover money stays in our pocket for future use.

Increment	⋯	A[4]	A[3]	A[2]	A[1]	A[0]	Bank	Pocket
0	⋯	0	0	0	0	0		
↓							2$	1$
1	⋯	0	0	0	0	1		

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is \$1. We always borrow \$2 from the bank.

- If we don't have enough money in our pocket, we borrow \$1 per necessary flip.
- Any leftover money stays in our pocket for future use.

Increment	⋯	A[4]	A[3]	A[2]	A[1]	A[0]	Bank	Pocket
0	⋯	0	0	0	0	0		
↓							2\$	1\$
1	⋯	0	0	0	0	1		
↓							2\$	1\$
2	⋯	0	0	0	1	0		

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is \$1. We always borrow \$2 from the bank.

- If we don't have enough money in our pocket, we borrow \$1 per necessary flip.
- Any leftover money stays in our pocket for future use.

Increment	⋯	A[4]	A[3]	A[2]	A[1]	A[0]	Bank	Pocket
0	⋯	0	0	0	0	0		
↓							2\$	1\$
1	⋯	0	0	0	0	1		
↓							2\$	1\$
2	⋯	0	0	0	1	0		
↓							2\$	2\$
3	⋯	0	0	0	1	1		

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is \$1. We always borrow \$2 from the bank.

- If we don't have enough money in our pocket, we borrow \$1 per necessary flip.
- Any leftover money stays in our pocket for future use.

Increment	⋯	A[4]	A[3]	A[2]	A[1]	A[0]	Bank	Pocket
0	⋯	0	0	0	0	0		
↓							2\$	1\$
1	⋯	0	0	0	0	1		
↓							2\$	1\$
2	⋯	0	0	0	1	0		
↓							2\$	2\$
3	⋯	0	0	0	1	1		
↓							2\$	1\$
4	⋯	0	0	1	0	0		

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is \$1. We always borrow \$2 from the bank.

- If we don't have enough money in our pocket, we borrow \$1 per necessary flip.
- Any leftover money stays in our pocket for future use.

Increment	⋯	**A[4]**	**A[3]**	**A[2]**	**A[1]**	**A[0]**	**Bank**	**Pocket**
0	⋯	0	0	0	0	0		
↓							2\$	1\$
1	⋯	0	0	0	0	1		
↓							2\$	1\$
2	⋯	0	0	0	1	0		
↓							2\$	2\$
3	⋯	0	0	0	1	1		
↓							2\$	1\$
4	⋯	0	0	1	0	0		
↓							2\$	2\$
5	⋯	0	0	1	0	1		

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is \$1. We always borrow \$2 from the bank.

- If we don't have enough money in our pocket, we borrow \$1 per necessary flip.
- Any leftover money stays in our pocket for future use.

Increment	···	**A[4]**	**A[3]**	**A[2]**	**A[1]**	**A[0]**	**Bank**	**Pocket**
0	···	0	0	0	0	0		
↓							2\$	1\$
1	···	0	0	0	0	1		
↓							2\$	1\$
2	···	0	0	0	1	0		
↓							2\$	2\$
3	···	0	0	0	1	1		
↓							2\$	1\$
4	···	0	0	1	0	0		
↓							2\$	2\$
5	···	0	0	1	0	1		
↓							2\$	2\$
6	···	0	0	1	1	0		

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is \$1. We always borrow \$2 from the bank.

- If we don't have enough money in our pocket, we borrow \$1 per necessary flip.
- Any leftover money stays in our pocket for future use.

Increment	⋯	**A[4]**	**A[3]**	**A[2]**	**A[1]**	**A[0]**	**Bank**	**Pocket**
0	⋯	0	0	0	0	0		
↓							2$	1$
1	⋯	0	0	0	0	1		
↓							2$	1$
2	⋯	0	0	0	1	0		
↓							2$	2$
3	⋯	0	0	0	1	1		
↓							2$	1$
4	⋯	0	0	1	0	0		
↓							2$	2$
5	⋯	0	0	1	0	1		
↓							2$	2$
6	⋯	0	0	1	1	0		
↓							2$	3$
7	⋯	0	0	1	1	1		

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is $1. We always borrow $2 from the bank.

- If we don't have enough money in our pocket, we borrow $1 per necessary flip.
- Any leftover money stays in our pocket for future use.

Increment	⋯	**A[4]**	**A[3]**	**A[2]**	**A[1]**	**A[0]**	**Bank**	**Pocket**
0	⋯	0	0	0	0	0		
↓							2$	1$
1	⋯	0	0	0	0	1		
↓							2$	1$
2	⋯	0	0	0	1	0		
↓							2$	2$
3	⋯	0	0	0	1	1		
↓							2$	1$
4	⋯	0	0	1	0	0		
↓							2$	2$
5	⋯	0	0	1	0	1		
↓							2$	2$
6	⋯	0	0	1	1	0		
↓							2$	3$
7	⋯	0	0	1	1	1		
↓							2$	1$
8	⋯	0	1	0	0	0		

BACK TO BINARY COUNTER

AMORTIZED ANALYSIS

Rules: The cost of any flip is \$1. We always borrow \$2 from the bank.

- If we don't have enough money in our pocket, we borrow \$1 per necessary flip.

Observations

- #1 Pocket dollars = Aces on the counter register
- #2 Bank loan of 2\$ is always sufficient.

(Proof by induction ☺)

↓							2\$	2\$
3	⋯	0	0	0	1	1		
↓							2\$	1\$
4	⋯	0	0	1	0	0		
↓							2\$	2\$
5	⋯	0	0	1	0	1		
↓							2\$	2\$
6	⋯	0	0	1	1	0		
↓							2\$	3\$
7	⋯	0	0	1	1	1		
↓							2\$	1\$
8	⋯	0	1	0	0	0		

Back to Binary Counter

Amortized Analysis

Rules: The cost of any flip is $1. We always borrow $2 from the bank.

- If we don't have enough money in our pocket, we borrow $1 per necessary flip.

Observations

- #1 Pocket dollars = Aces on the counter register
- #2 Bank loan of 2$ is always sufficient.

(Proof by induction ☺)

Amortized Analysis

$$\#\ \text{operations} = \underbrace{(\text{Dollars we borrowed})}_{\text{bank cost}} - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}}$$

$$\#\ \text{operations} = 2n - \underbrace{(\text{Dollars left in our pocket})}_{\text{Potential overcharging}} \leq 2n$$

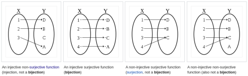
...	0	1	0	0	0	

Appendix

References

Image Sources I

https://brand.wisc.edu/web/logos/



https://en.wikipedia.org/wiki/Bijection