Amoritized Analysis

We saw best/worst/expected/average case analysis.

- We analyze a data structure over a sequence of operations
- We do not analyze a single operation
- It provides a guarantee of the average performance an worst case
- Deterministic (no probability involved)

Two methods to analyze

- Aggregate
- Accounting
- Potential

Examples:

1. Stack: we have a stack with push, pop, multipop
2. Binary counter:

```
Increment (CTR)
i = 0
while i <length(CTR) and A[i]=1 do
    CTR[i] = 0      //turn off 1-bits
    i = i+1
 if i< length (CTR) then
    CTR[i]=1;      //turn on highest bit
```

1. Aggregate: Stack

Naive way:

Pop	O(1)
Push	O(1)
Multipop	O(n)

⇒ n operations can cost up to $n \cdot O(n) = O(n^2)$

Aggregate: you cannot pop more than what you pushed. And you can push at most n items.

Total amount of cost is O(n) or $\frac{O(n)}{n} = O(1)$ /operation

2. Aggregate: Binary counter

Naive: cost comes from flipping a bit. If we call increment O(n) times on a counter that has n-bits, this may cost:

$O(n) \cdot n = O(n^2)$ in total or $\frac{O(n^2)}{n} = O(n)$ /Increment

Aggregate:

0 0 0 0 flips all the time (n-times) → $\frac{n}{2^0}$

0 0 0 1

0 0 1 0 flips every other time ($\frac{n}{2}$-times) → $\frac{n}{2^1}$

0 0 1 1

0 1 0 0 flips every $\frac{n}{4}$ times → $\frac{n}{2^2}$

0 1 0 1

0 1 1 0

0 1 1 1

1 0 0 0 flips every $\frac{n}{8}$ times → $\frac{n}{2^3}$

⋮

⇒ For n-increments, the total # flips: $\sum_{i=0}^{n} \frac{n}{2^i} \leq n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2 \cdot n = O(n)$

or $\frac{O(n)}{n} = O(1)$ amortized time

Accounting Method

- In accounting method the data structure behaves like a bank
- We charge $ for each op. This is the amortized cost of the op
- when:

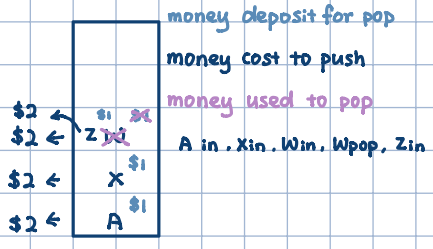
amortized cost < actual cost ⇒ deposit leftover as credit on the data structure

amortized cost > actual cost ⇒ use the credit saved in the data structure

• Your data structure cannot run on negative credit! If you end up with negative balance then your analysis is wrong

1. Accounting: Stack

	Actual	Amortized	
PUSH:	O(1)	$2	→ $1 for push, $1 deposit for pop later
POP:	O(1)	$0	
Multipop:	O(n)	$0	



⇒ For n operations, use $2n = O(n) is used in total or $\frac{O(n)}{n} = O(1)$ amortized time

2. Accounting: Binary counter

Charge $2 for every 0 → 1 flip:

$1 pays flip, and

$1 stays as credit to pay for the subsequent 1→0 flip

0 0 0 0 ⇒ For n-increments, we need $2·n or O(n)

or $\frac{O(n)}{n} = O(1)$ / increment in an amortized sense

0 0 0 1

0 0 1 0

0 0 1 1

0 1 0 0

0 1 0 1

0 1 1 0

0 1 1 1

1 0 0 0