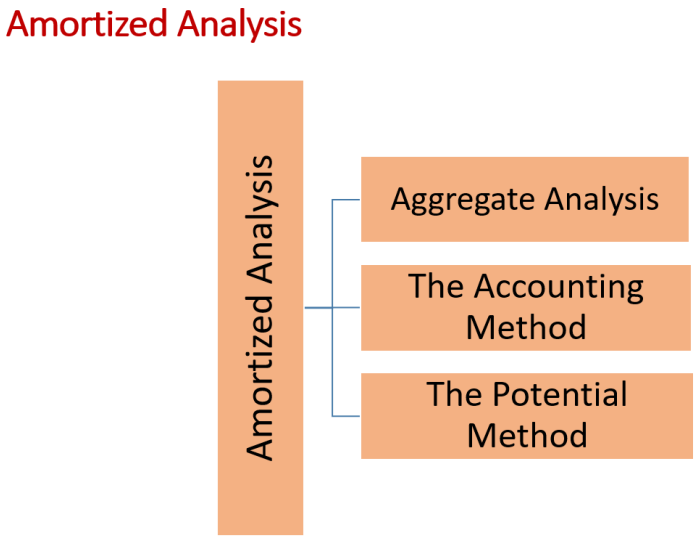
Amortized Analysis

- **Definition:** Amortized analysis is a technique used to analyze the average time or cost of a sequence of data structure operations, where we average the cost over all operations performed.

- **Objective:** The goal of amortized analysis is to demonstrate that even though some individual operations might be costly, the average cost per operation is relatively low.

- **Average in Context:** In this context, "average" doesn't involve a distribution of inputs or probabilities. It simply means calculating the cost per operation over a sequence.

- **Average Cost in the Worst Case:** Amortized analysis focuses on the worst-case scenario for the entire sequence of operations, not individual operations.



Two Problems:

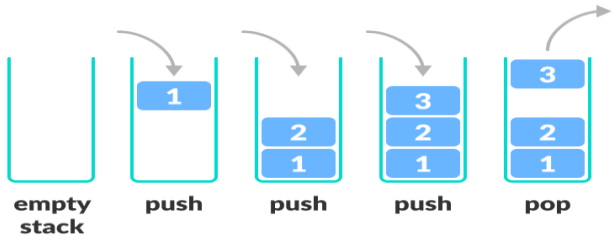
1. **Stack Operations, Including Multipop:** Analyzing the cost of a sequence of stack operations, including multipop.

2. **Binary Counter Using Increment Operation:** Analyzing the cost of incrementing a binary counter.

Simple Analysis (Stack Operations):

- Starting with an empty stack, the stack's size can be at most O(n).
- A multipop operation might appear to have a complexity of O(n) in isolation.
- If there are n operations, the upper bound on complexity appears to be O(n^2).
- However, this upper bound is an overestimate and not a realistic reflection of the actual cost.

Aggregate Analysis (Stack Operations):



- Considering a sequence of n PUSH, POP, and MULTIPOP operations.
- The worst-case cost of MULTIPOP is O(n).
- Since there are n operations in total, the worst-case cost of the sequence seems to be O(n^2).
- Observation: Each object can only be popped once for every time it's pushed.
- As there are n PUSHes, there will be n POPs, including those in MULTIPOP.
- Therefore, the total cost is O(n).
- When averaged over the n operations, the cost per operation on average is O(1).

Binary counter

k-bit binary counter $A[0..k-1]$ of bits, where $A[0]$ is the least significant bit and $A[k-1]$ is the most significant bit.

Counts upward from 0.

Value of counter is $\sum_{i=0}^{k-1} A[i] \cdot 2^i$.

Analysis

Each call could flip k bits, so n INCREMENTs takes $O(nk)$ time.

Binary counter

Observation

Not every bit flips every time.

[Show costs from above.]

bit	flips how often	times in n INCREMENTS
0	every time	n
1	1/2 the time	$\lfloor n/2 \rfloor$
2	1/4 the time	$\lfloor n/4 \rfloor$
	⋮	
i	$1/2^i$ the time	$\lfloor n/2^i \rfloor$
	⋮	
$i \geq k$	never	0

Binary counter

$$\begin{aligned}\text{Therefore, total \# of flips} &= \sum_{i=0}^{k-1} \lfloor n/2^i \rfloor \\ &< n \sum_{i=0}^{\infty} 1/2^i \\ &= n\left(\frac{1}{1-1/2}\right) \\ &= 2n\ .\end{aligned}$$

Therefore, n INCREMENTs costs $O(n)$.
Average cost per operation $= O(1)$.

Incrementing Binary Numbers

- Assume bits A[k-1] to A[0]

$$x = \sum_{i=0}^{k-1} A[i] * 2^i$$

- Incrementing starting from 0

```
INCREMENT(A)
1  i ← 0
2  while i < length[A] and A[i] = 1
3       do A[i] ← 0
4          i ← i + 1
5  if i < length[A]
6     then A[i] ← 1
```

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

- The diagram to the right counts the number of bit flips; for the first 16 numbers the total cost is 31;

Simple Analysis for Binary Counter:

In this analysis, we are measuring the cost of operations in terms of the number of bit flips in a binary counter. Some operations flip only one bit, while others can ripple through and flip multiple bits. The goal is to find the average cost per operation.

- **Cursory Analysis:**
  - The worst-case cost for the increment operation is O(k), where k is the number of bits in the binary counter.
  - If we have a sequence of n increment operations, it might seem that the worst-case behavior would be O(n * k).

- However, this upper bound is not very tight and might overestimate the actual cost.

Amortized Analysis for Binary Counter:

In an amortized analysis, we consider that not all bits are flipped in each iteration of the increment operation. Some bits flip more frequently than others, following a pattern:

- A[0] is flipped every iteration.
- A[1] is flipped every other iteration.
- A[2] is flipped every fourth iteration, and so on.
- For $i > \lfloor \log_2 n \rfloor$, the bit A[i] never flips.

By summing all the bit flips, we can determine the worst-case cost:

- The first $\lfloor \log_2 n \rfloor$ bits flip in various patterns, but their total flips are bounded by a constant.
- For the remaining bits ($i > \lfloor \log_2 n \rfloor$), they never flip.
- So, the worst-case cost is bounded by O(n).

Now, to find the average cost per operation, we divide this worst-case cost (O(n)) by the number of operations (n):

- O(n) / n = O(1).

Therefore, in the amortized analysis, the average cost per increment operation on the binary counter is O(1), which is much tighter and more accurate than the initial worst-case estimate of O(n * k).