Amortized Analysis

- Sometimes it is more useful to think about the average run time of an operation as part of a sequence of operations

- Different from average-case analysis
    - Amortized analysis guarantees the average performance of each operation in the worst case

- Adding multipop to a stack

- If we perform n operations on a stack (including push, pop, and multipop) what is the worst case run time?

- Multipop pops k elements from the stack $\Theta(k)$

- Push and pop are constant time

- Upper bound is $O(nk)$

				13	
	10		12	12	
8	8	8	8	8	
push(8)	push(10)	pop()	push(12)	push(13)	multipop(3)

```
MULTIPOP(S, k)
  while S is not empty and k > 0
      POP(S)
      k = k - 1
```

- We can never pop more elements than we can push
- The sum of the costs of all operations must be $O(n)$
  because we can't push more than n times
- the average cost of an operation is $\dfrac{O(n)}{n} = O(1)$
- The amortized cost of multipop is $O(1)$ over a large number of operations

- Incrementing a binary number

$$01001100 + 1 = 01001101$$

one bit changes

$$\begin{array}{r} 1111111 \\ 01111111 \\ +\quad 1 \\ \hline 10000000 \end{array}$$

8 bits changed

- Start from the right and flip 1's to 0's until we find a 0, which we flip to 1
- $O(k)$ where k is the number of bits

INCREMENT(A, k)

$i = 0$
while $i < k$ and $A[i] == 1$
  $A[i] = 0$
  $i = i + 1$
if $i < k$
  $A[i] = 1$

A is a 0-indexed array of k bits. A[0] is the least significant bit, A[k-1] is the most significant

- n repeated increments

  $O(nk)$, but we can achieve a tighter bound

- $A[0]$ flips every time
- $A[1]$ flips every other time
- $A[i]$ flips $\left\lfloor \frac{n}{2^i} \right\rfloor$ times

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

Total flips is

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i}$$

$$= 2n = O(n)$$

Each increment is amortized $O(1)$ over n increments

- Dynamic tables
    - Commonly seen as growable arrays
        - Python lists, C++ vectors, Java ArrayLists, etc.
    - Elements occupy a certain number of slots, which may be less than the slots allocated for storage
    - If the # of elements reaches the # of slots allocated, the array must grow when appending

T.num is the number of elements, T.size is the number of slots

Best case is $O(1)$

Worst case is $\Theta(T.num)$

TABLE-INSERT(T, x)

```
if T.size == 0
    allocate T.table with 1 slot
    T.size = 1
if T.num == T.size                                  // expand?
    allocate new-table with 2 · T.size slots
    insert all items in T.table into new-table      // T.num elem insertions
    free T.table
    T.table = new-table
    T.size = 2 · T.size
insert x into T.table                               // 1 elem insertion
T.num = T.num + 1
```

- Cost of a single Table-Insert

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

- Total cost of n calls to Table-Insert

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j$$

$$< n + 2n$$

$$= 3n = O(n)$$

Amortized time for inserting is $O(1)$

Writing in Computer Science

- Papers are generally organized like this:
    - Abstract
    - Background
    - Research carried out
    - Results
    - Conclusion and future work

- Avoid informal language
- The paper is about the work, not you
    - Do not provide personal motivation for the research
    - Avoid first person singular pronouns and minimize the use of we and our

I found that algorithm A is faster than algorithm B

We found that A was faster than B

It was found that A is faster than B

Preferred: Results show that A is faster than B

- Use present tense as much as possible