Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 2

University of Southern California

Review

Amortized Cost

Reading: chapters 1 & 2

Ch1: review questions

2. (T/F) Any function which is $\Omega(\log n)$ is also $\Omega(\log(\log n))$.

$\exists c,\ f(n) \geq c \cdot \log n \geq c_2 \cdot \log\log n$

$\Rightarrow f(n) \in \Omega(\log\log n)$

3. (T/F) If $f(n) = \Theta(g(n))$ then $g(n) = \Theta(f(n))$.

$\exists c_1, c_2 \quad \underbrace{c_1 \cdot g(n) \leq f(n)} \leq c_2 \cdot g(n)$

$g(n) \leq \frac{1}{c_1} f(n)$ $\quad$ $g(n) \geq \frac{1}{c_2} f(n)$

$\frac{1}{c_2} f(n) \leq g(n) \leq \frac{1}{c_1} f(n)$

$n! = n\cdot(n-1)\cdots 2\cdot 1 \le n\cdot n\cdots n\cdot n = n^n$

Ch1: exercises

$\boxed{\frac{1}{\log n} = \log_n 2}$

4. Arrange the following functions

$$\underset{\approx n^2}{\overset{5}{4^{\log n}}},\ \overset{2}{\sqrt{\log n}},\ \overset{7}{n^{\log\log n}},\ \underset{\approx\sqrt{n}}{\overset{4}{(\sqrt{2})^{\log n}}},\ \overset{3}{2^{\sqrt{2\log n}}},\ \underset{\approx 2}{\overset{1}{n^{1/\log n}}},\ \overset{6}{(\log n)!}$$

in increasing order of growth rate with $g(n)$ following $f(n)$ in your list if and only if $f(n) = O(g(n))$.

$2^{\sqrt{2\log n}} \overset{?}{<} \sqrt{n}$

$\sqrt{2\log n} < \frac{1}{2}\log n$

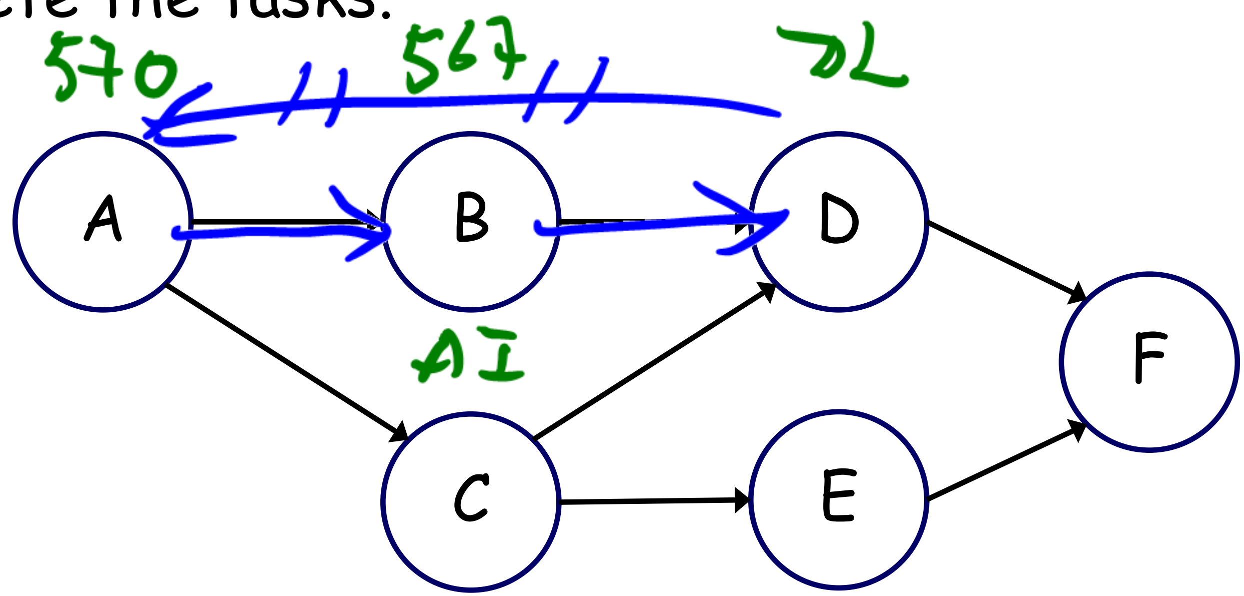
$n^{\log\log n}$ $(\log n = t)$ $= n^{\log t} = n^{\frac{\log_n t}{\log_n 2}} = t^{1/\log_n 2} = t^{\log_2 n} = (\log n)^{\log n}$

$(\log n)! \le (\log n)^{\log n}$

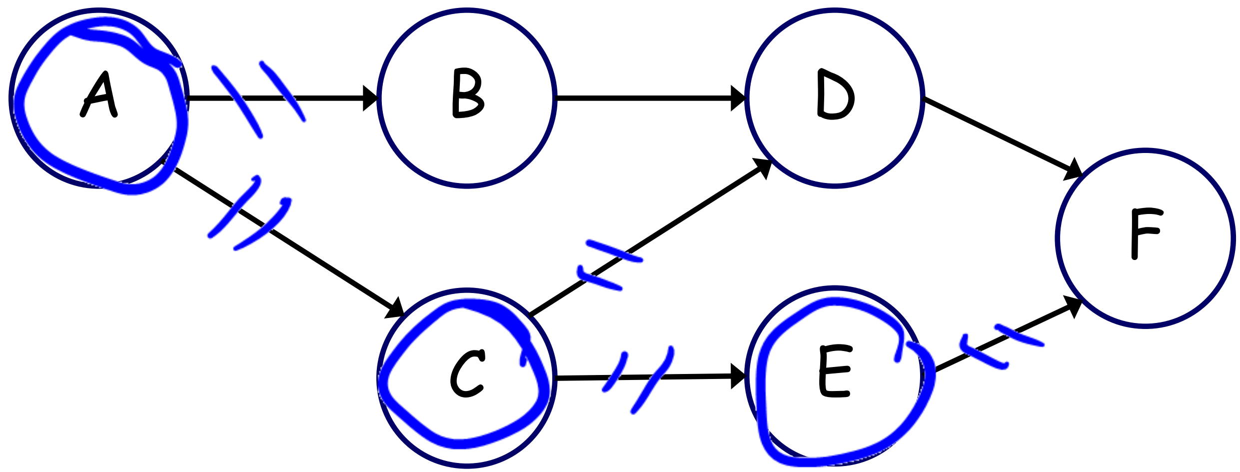
$\boxed{n! \le n^n}$

Topological Sort for $\boxed{\text{DAG}}$

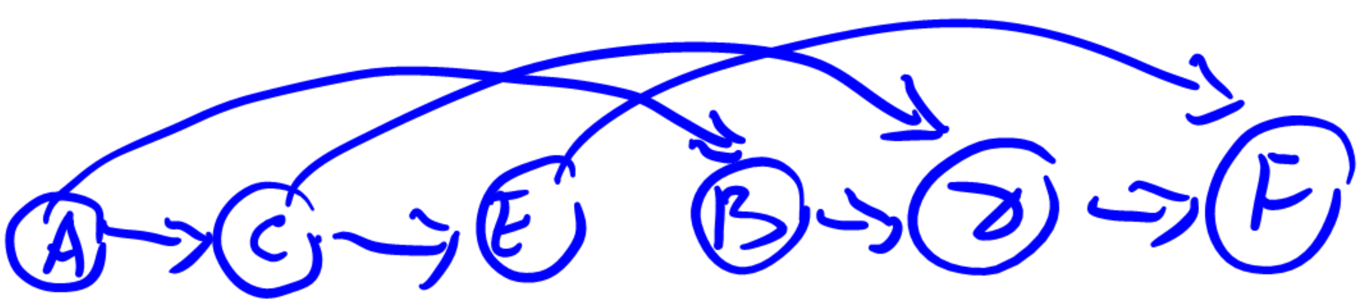
Suppose each vertex represents a task that must be completed, and a directed edge (u, v) indicates that task v depends on task u. That is task u must be completed before v. The topological ordering of the vertices is a valid order in which you can complete the tasks.



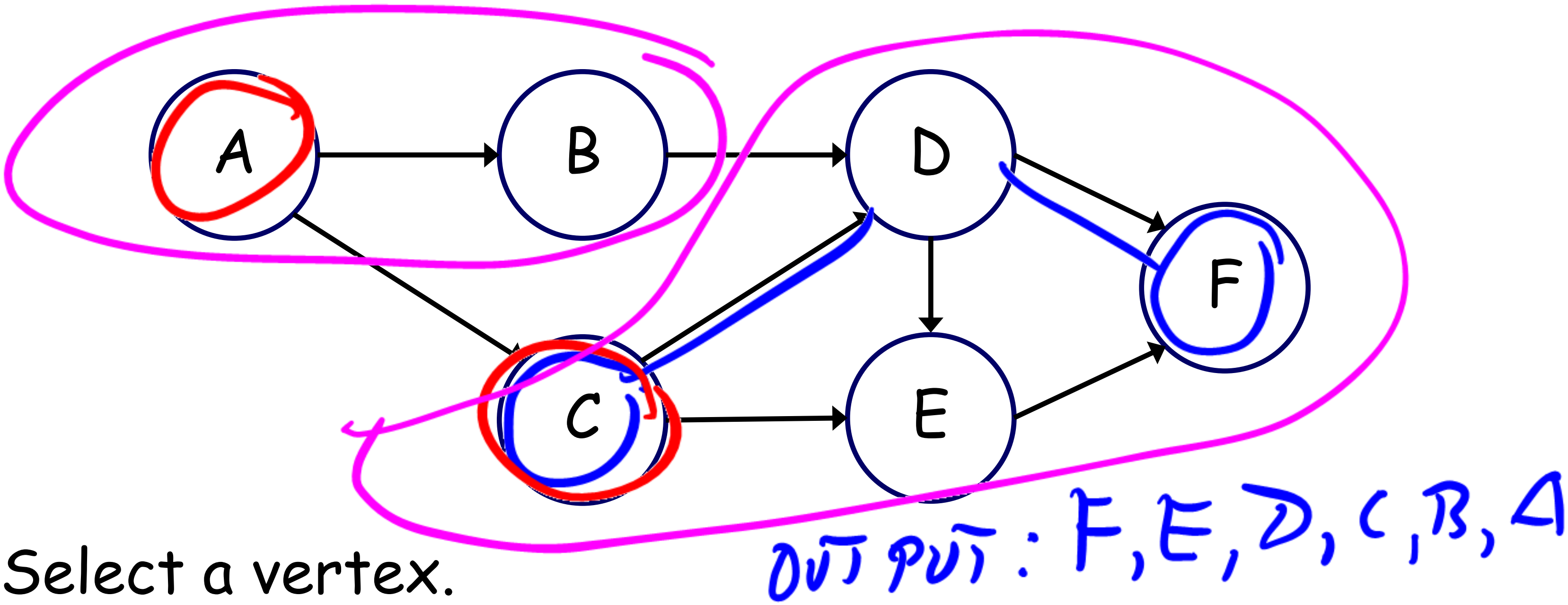
How to find a topological order?



output : A, C, E, B, D, F



Linear Time Algorithm



- Select a vertex.
- Run DFS and return vertices that has no undiscovered leaving edges
- May run DFS several times

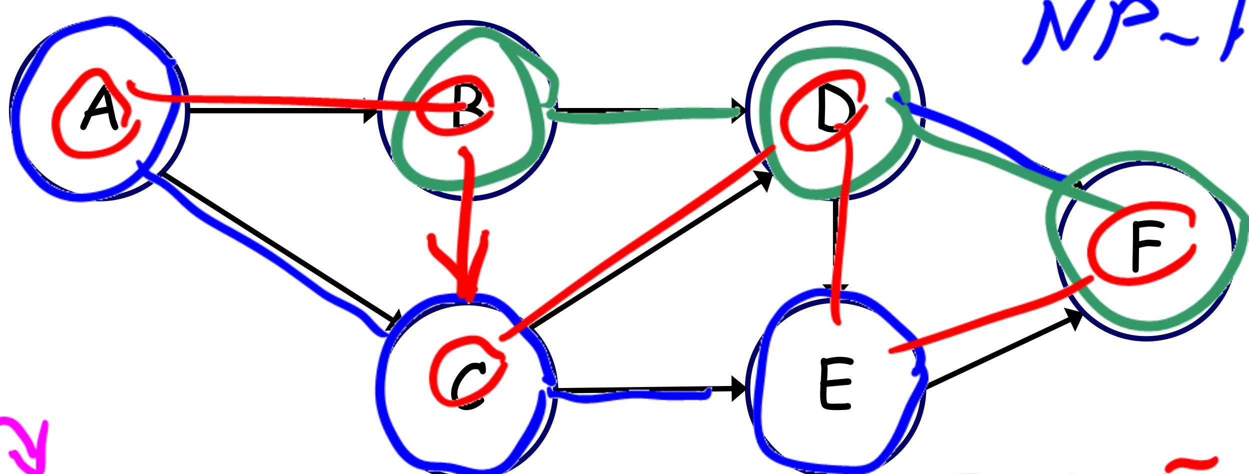
We get vertices in reverse order.

Why is it linear time?

Discussion Problem 1

We have discussed finding the shortest path between two vertices in a graph. Suppose instead we are interested in finding the *longest* path in a directed acyclic graph (DAG). In particular, we are interested in a path that visits all vertices. Give a linear-time algorithm to determine if such a path exist.

Hamiltonian Path
NP-Hard



A, C, E, B, D, F

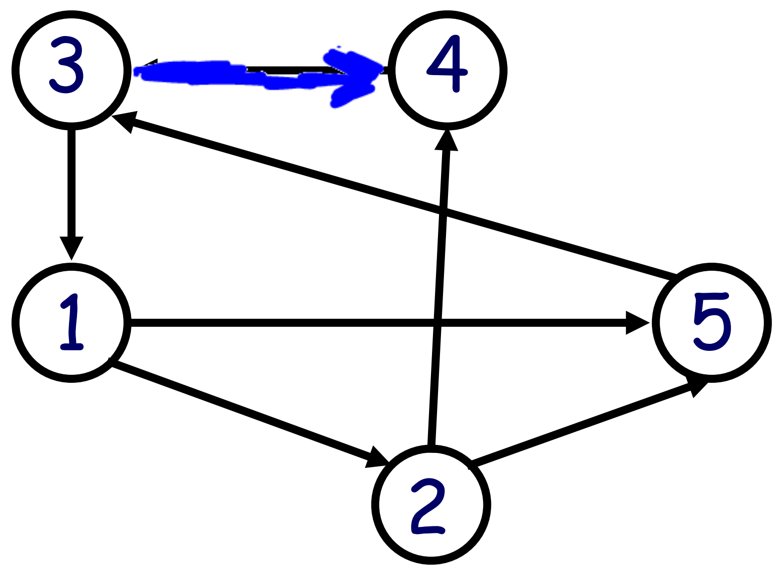
A, B, C, D, E, F

Strongly Connected Graphs

Given a directed graph. It's strongly connected if each vertex is reachable from any other vertex.

If we reverse the edge (3,4), it won't be strongly connected.

It's called a weakly connected graph.



How do you test if a graph is strongly connected?

Strongly Connected Graphs

Brute Force: Run DFS from every vertex

Complexity: $O(V \cdot (V+E))$

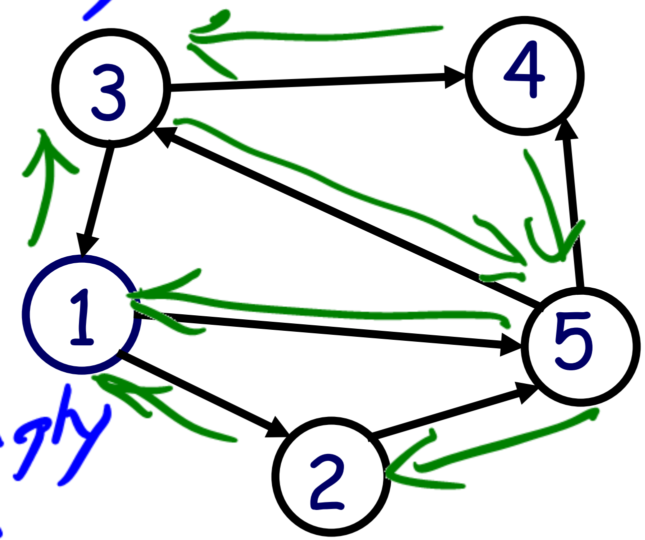
$M[2,1]=0 \quad M[1,2]=1$

Linear-time Algorithm:

1. Pick a vertex, run DFS
2. if DFS does not visit all vertices → not strongly conn.

??

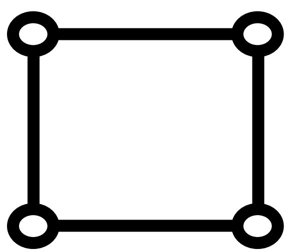
3. Reverse edge directions.
4. Run DFS again from the same vertex
5. if DFS does not visit all vertices → not



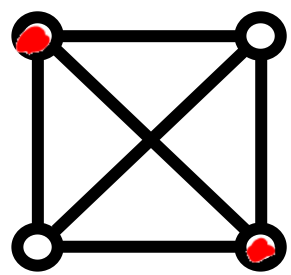
$G = M^T$

Planar Graphs

A graph is planar if it can be drawn in the plane without crossing edges

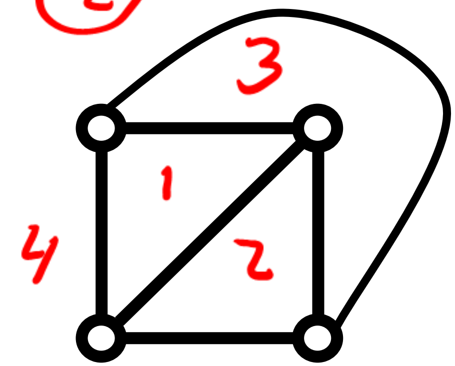


1

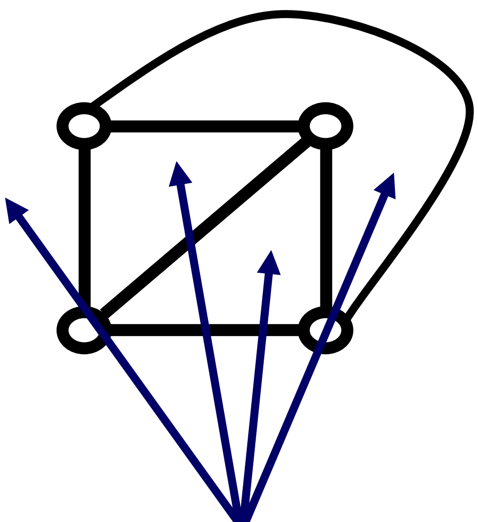


=

2



F = 4, V = 4, E = 6



4 faces

A planar graph when drawn in the plane, splits the plane into disjoint faces.

cube

$$\boxed{V - E + F = 2}$$

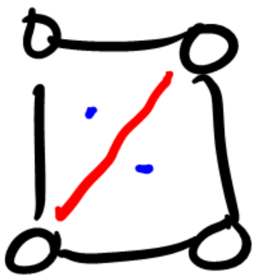
Euler's Formula

Theorem. If G is a connected planar graph with V vertices, E edges and F faces, then $V - E + F = 2$.

Proof. By induction on edges

1. Base case. , $V=2, E=1, F=1$
2. I.H. Assume it holds for graphs $E < m$ edges.
3. I.S. Prove $V-E+F=2$ for graphs with m edges.

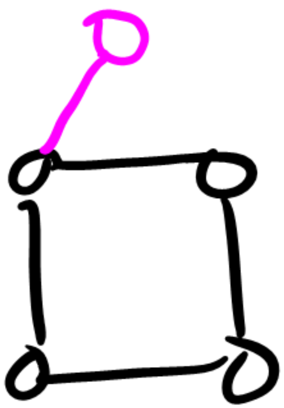
Proof of Euler's Formula



V unchanged

E++ $\qquad V-(E++)+(F++)=2$

F++



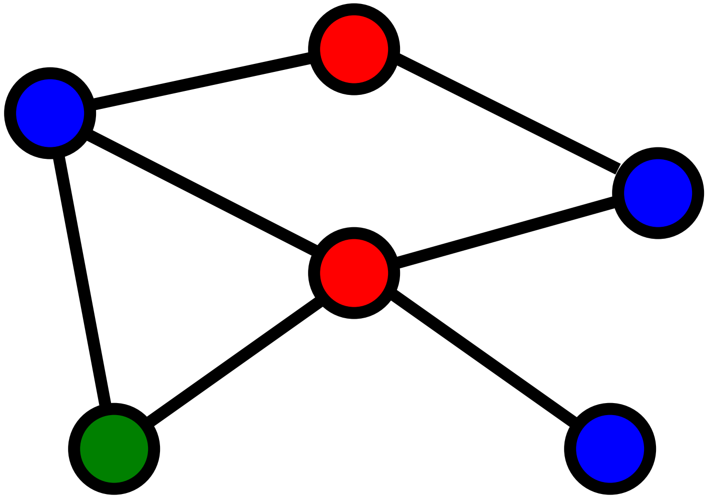
V++

E++

F unchanged

Coloring $\boxed{\text{Planar}}$ Graphs

A coloring of a graph is an assignment of a color to each vertex such that no neighboring vertices have the same color



Min number of colors?

4 Color Theorem (1976)

Theorem: Any simple planar graph can be colored with less than or equal to 4 colors.

It was proven in 1976 by K. Appel and W. Haken. They used a special-purpose computer program.

Since that time computer scientists have been working on developing a formal program proof of correctness. The idea is to write code that describes not only what the machine should do, but also why it should be doing it.

In 2005 such a proof has been developed by Gonthier, using the Coq logical proof system.

__Theorem__. Any simple planar graph can be colored with 6 colors.

Amortized Analysis

In a sequence of operations the worst case does not occur often in each operation - some operations may be cheap, some may be expensive.

Therefore, a traditional worst-case per operation analysis can give overly *pessimistic* bound.

When same operation takes different times, how can we accurately calculate the runtime complexity?

sorting n items

n! permutations

$\frac{2}{n!}$

$O(n)$ best-case

The Aggregate Method

The aggregate method computes the upper bound *T(n)* on the total cost of *n* operations.

The amortized cost of an operation is given by $\frac{T(n)}{n}$

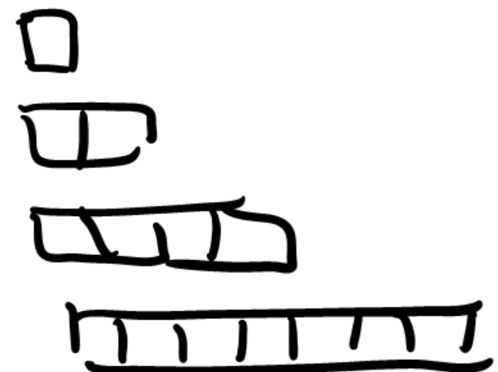
In this method each operation will get the same amortized cost, even if there are several types of operations in the sequence.

Unbounded Array

resize policy: double up

insert	old size	new size	# of copy
1	1	—	—
2	1	2	1
3	2	4	2
4	4	—	—
5	4	8	4
6	8		—
7	8		—
8	8		—
9	8	16	8

2^3+1



inserts = 9

copy = 1+2+4+8 = 15

Total work: 9+15 = 24

$AC = \frac{\text{Total work}}{\text{\# inserts}} = \frac{24}{9}$

$9 = 8+1 = 2^3+1$

Unbounded Array

of inserts 2^n+1

# of copy: $1+2+4+\ldots+2^n = \sum_{k=0}^{n} 2^k = 2^{n+1}-1$

Total work: $(2^n+1)+(2^{n+1}-1) = 3 * 2^n$

$$AC(\text{per insert}) = \frac{3\cdot 2^n}{2^n+1} \quad (n \to \infty)$$

$$\lim_{n\to\infty} \frac{3\cdot 2^n}{2^n+1} = 3$$

$$\boxed{AC = O(3)}$$

constant

Binary Counter

Given a binary number n with $\log(n)$ bits, stored as an array, where each entry $A[i]$ stores the i-th bit.

$n=3$

The cost of incrementing a binary number is the number of bits flipped.

000	
001	1
010	2
011	1
100	3
101	1
110	2
111	1
000	3

best-case $O(1)$

worst-case $O(\log n)$

$$AC = \frac{\text{Total \# of flips}}{n} = \frac{14}{8}$$

Binary Counter

Number n, it has $(\log n)$ bits

Compute the total # of flips

The least significant bit flips n times

The previous $n/2$

... $n/4$

The most significant bit 2

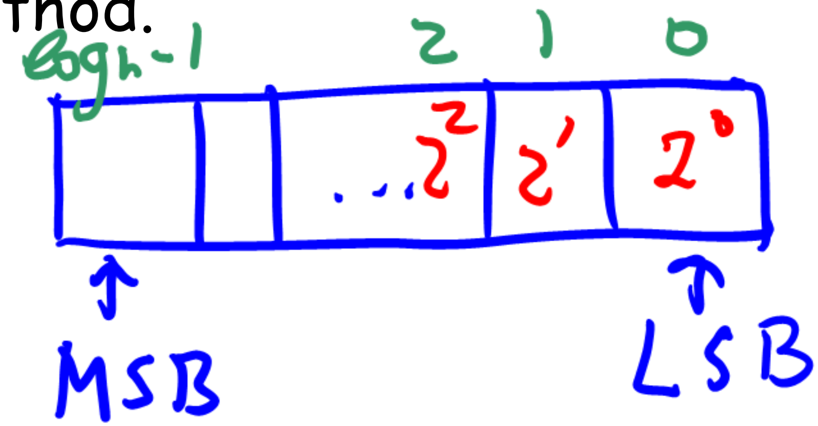
the total: $n+\frac{n}{2}+\frac{n}{4}+\dots+2 \leq n\left(1+\frac{1}{2}+\frac{1}{4}+\dots\infty\right)=2n$

geometric series

$AC=\frac{O(2n)}{n}=O(2)$

Discussion Problem 2

Another Binary Counter. Let us assume that the cost of a flip is 2^k to flip k-th bit. Flipping the lowest-order bit costs $2^0 = 1$, the next bit costs $2^1 = 2$, and so on. What is the amortized cost per increment? Use the aggregate method.



Total: $n \cdot 2^0 + \frac{n}{\cancel{2}} \cdot \cancel{2}^1 + \frac{n}{\cancel{4}} \cdot \cancel{2}^2 + \ldots + \cancel{2} \cdot 2^{\log n - \cancel{1}} =$

$= n + n + n + \ldots + n =$

$= n \cdot \log n$

$$AC = \frac{n \cdot \log n}{n} = O(\log n)$$

The Accounting Method

The accounting method (or the banker's method) computes the individual cost of each operation.

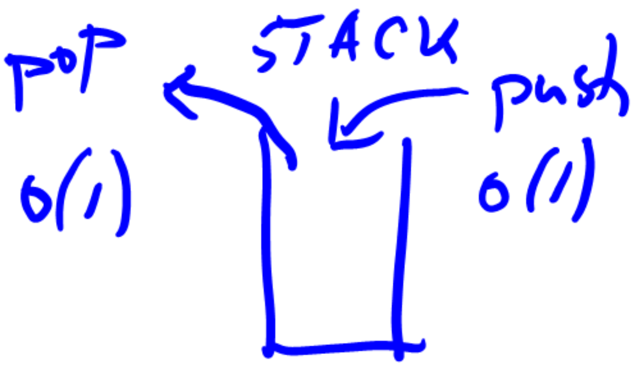
We assign different charges to each operation; some operations may charge more or less than they actually cost.

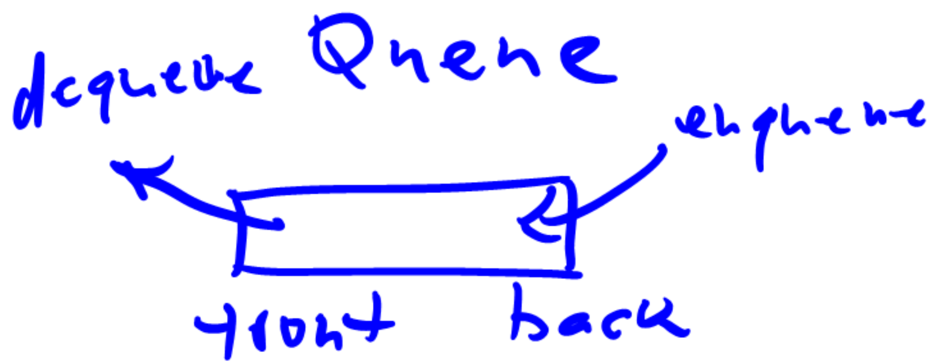
The amount we charge an operation is called its amortized cost.

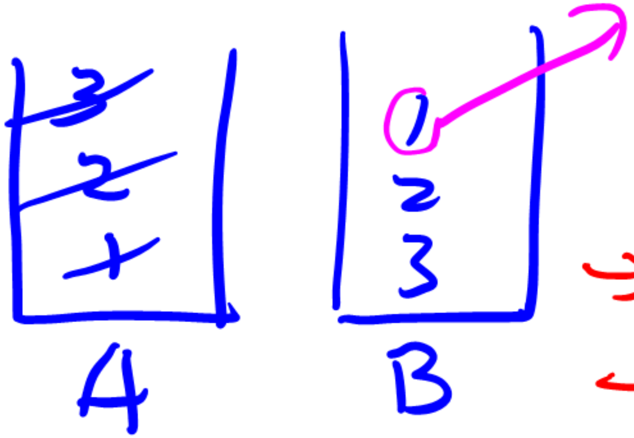
Discussion Problem 3

You have a stack data type, and you need to implement a FIFO queue. The stack has the usual POP and PUSH operations, and the cost of each operation is 1. The FIFO has two operations: ENQUEUE and DEQUEUE.

We can implement a FIFO queue using two stacks. What is the amortized cost of ENQUEUE and DEQUEUE operations?







$O(1)$

enqueue: 1, 2, 3 A.push
dequeue: move from A to B A.pop, B.push $O(n)$
dequeue: B.pop $O(1)$

1. How many tokens to enqueue? 3 tokens, 1 token, 1 token
2. How many tokens to dequeue. $O(n)$

more tokens to enqueue

3 tokens to enqueue!! 1 token for A.push

Bank: $2+2+2 - 3 \cdot 2$