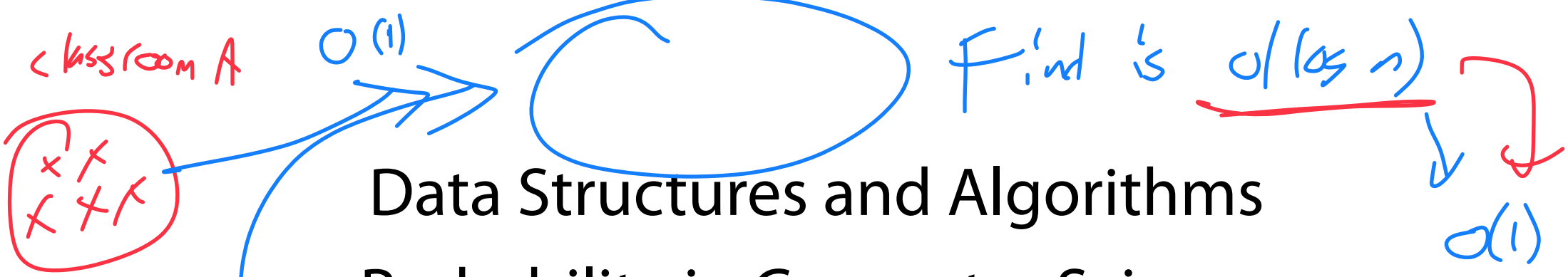


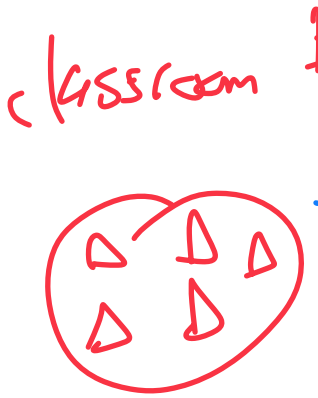
Data Structures and Algorithms

Probability in Computer Science

CS 225

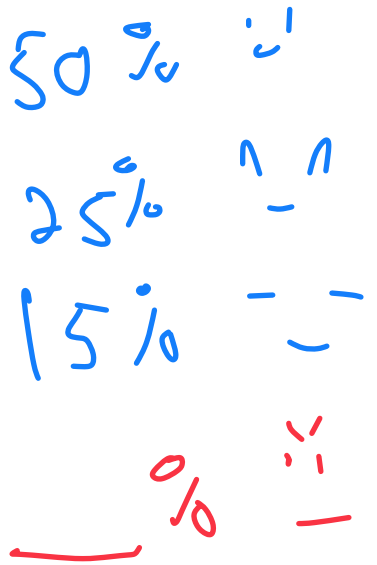
Brad Solomon

October 23, 2022



UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science



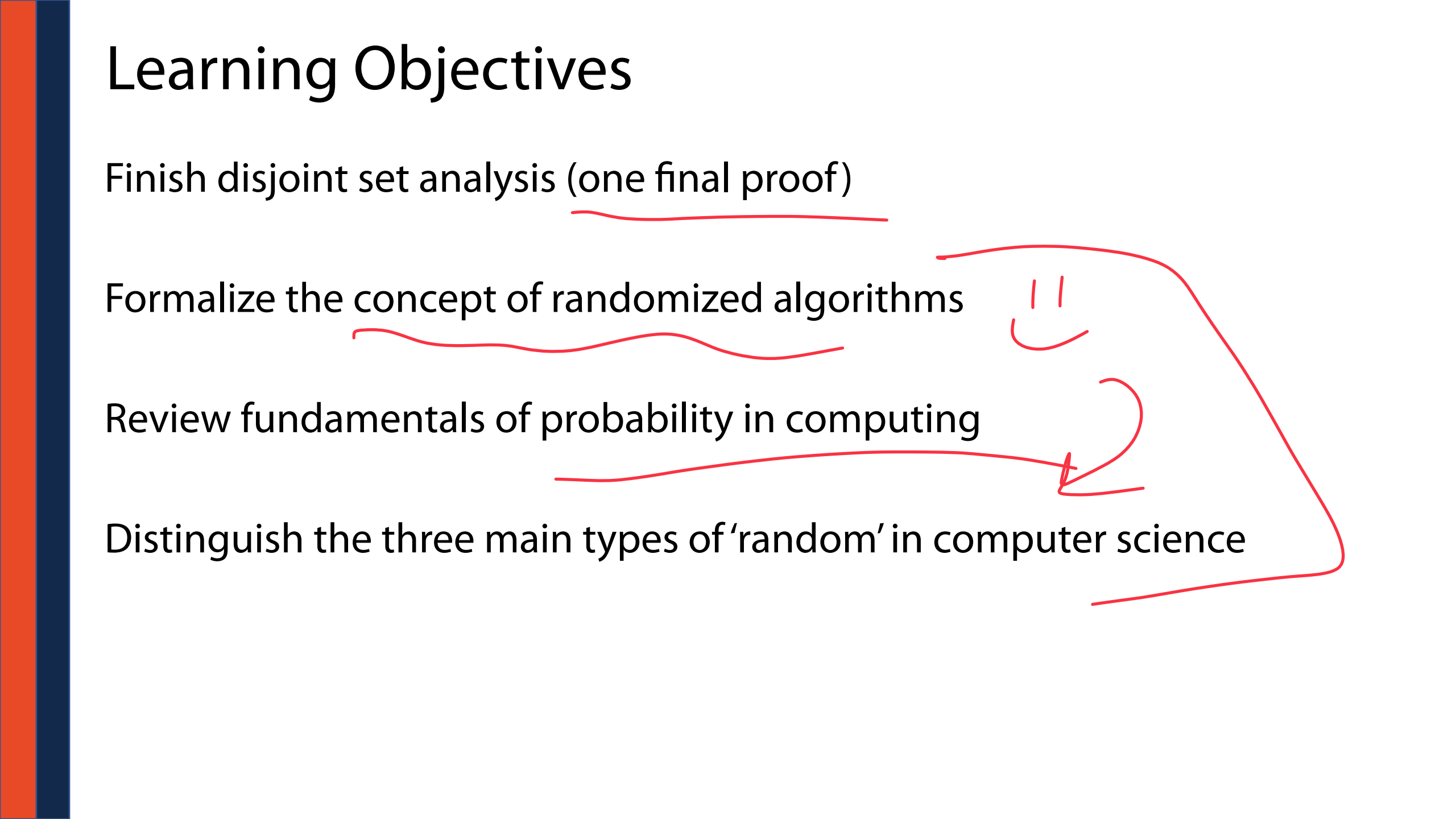
Learning Objectives

Finish disjoint set analysis (one final proof)

Formalize the concept of randomized algorithms

Review fundamentals of probability in computing

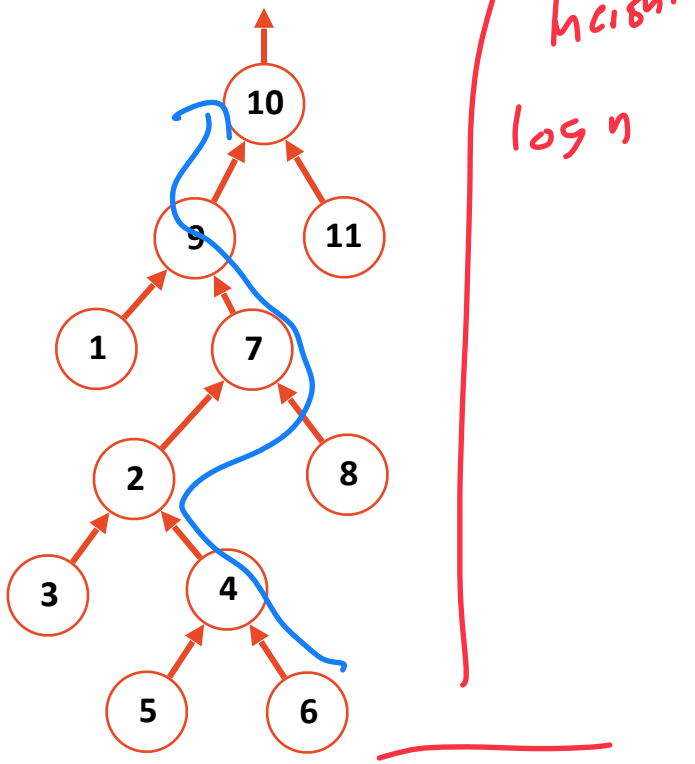
Distinguish the three main types of 'random' in computer science



Disjoint Sets w/ Path Compression

(and rank)

How do we observe how the efficiency of a set changes due to PC?



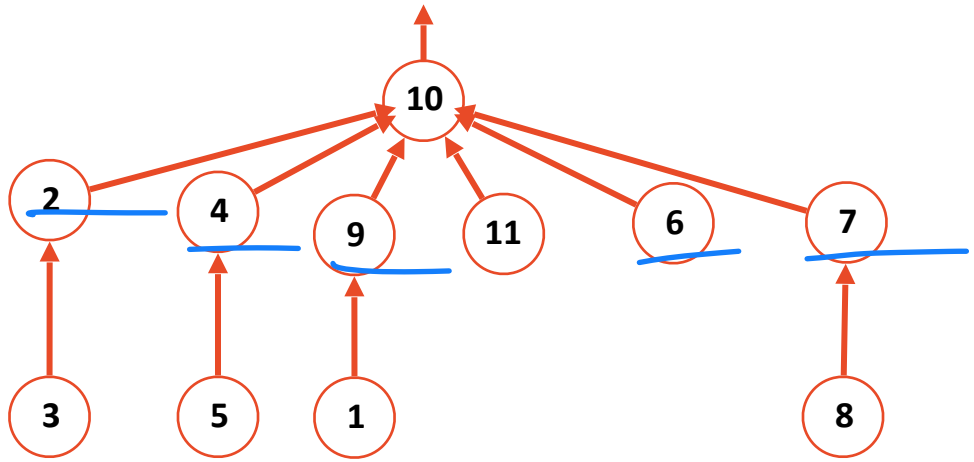
height

log n

Single op can be slow!

Amortized time!

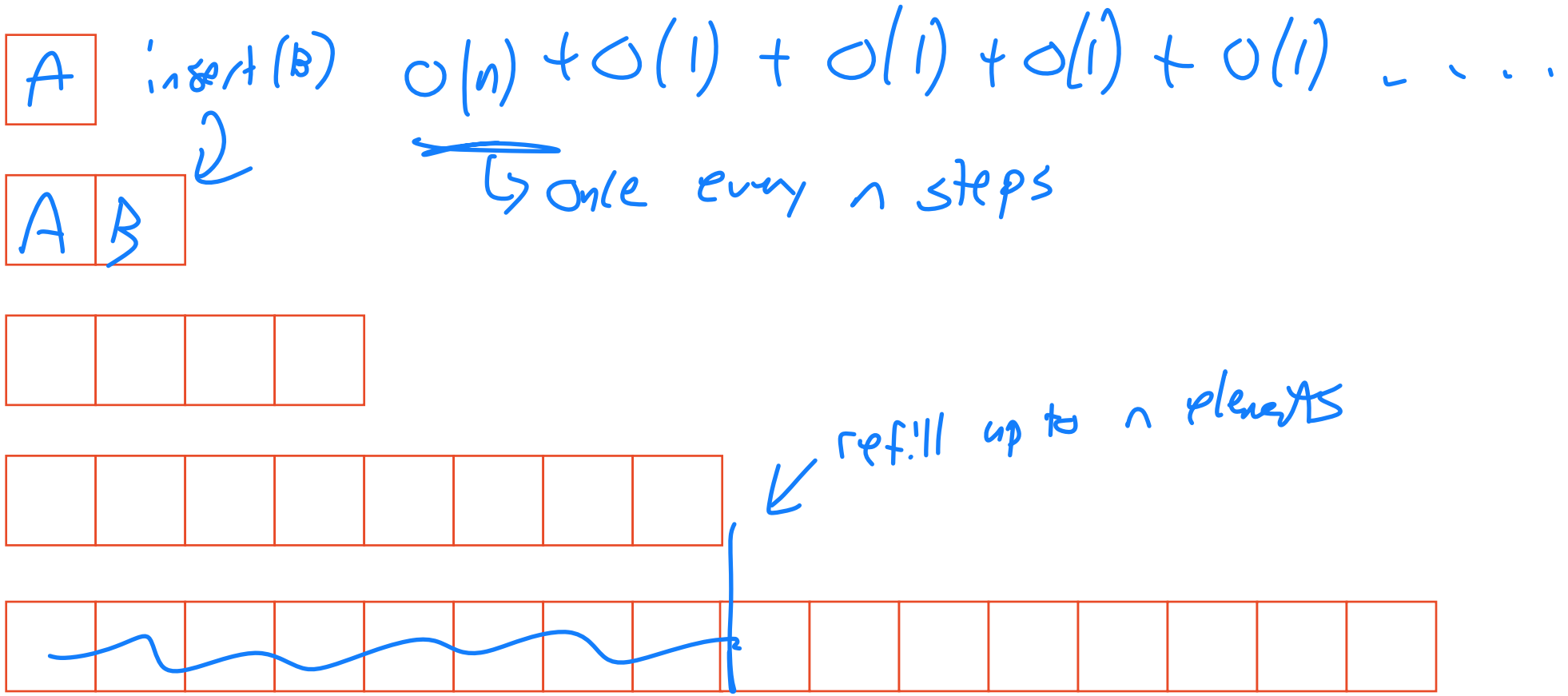
Eventually speed up



Amortized Time Review

We have **n items**. We make **n insert()** calls.

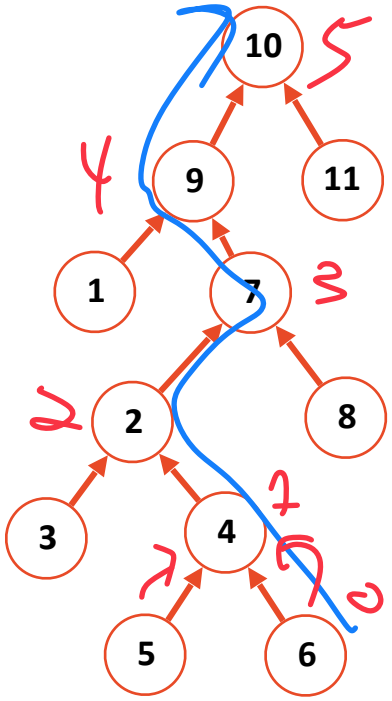
We are interested in the **worst case work** possible **over n calls**.



Amortized Time (Rank w/ Path Compression)

We have **n items** in an Uptree. We make **m find()** calls.

We are interested in the **worst case work** possible **over m calls**.

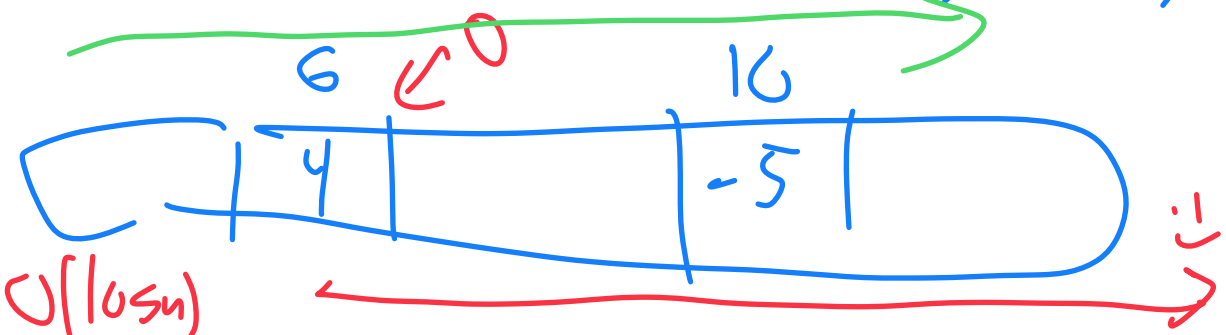


Once we find once, save work in future!

↳ Max work we can do

$n \to \infty$

$m \to \infty$ work $O(n)$ and $O(1)$



1 find – $O(\log n)$

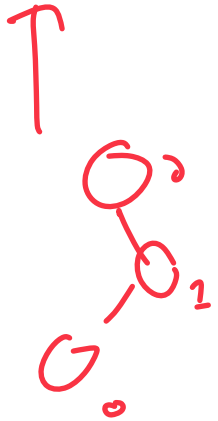
B/C this is optimal because of set 10

$O(1)$

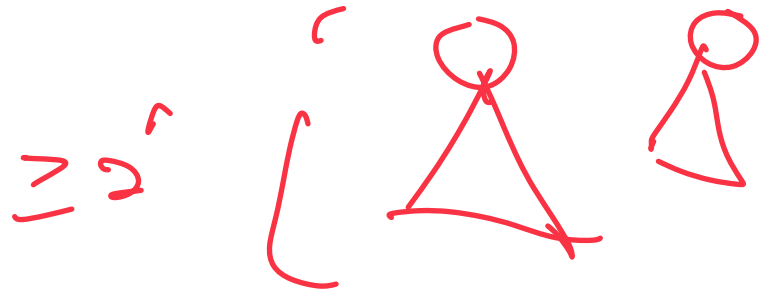
Key Properties of UpTree by rank w/ PC

The parent of a node is always higher rank than the node.

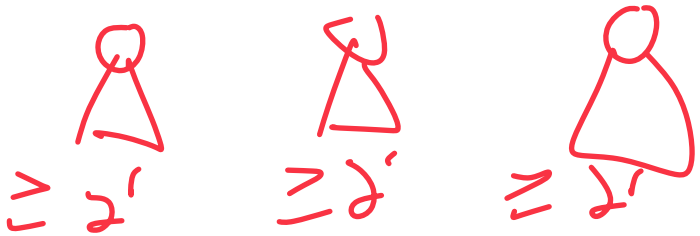
↳ we only store rank of roots but we can label all nodes



The min(nodes) in a set with a root of rank r has $\geq 2^r$ nodes.



For any integer r, there are at most $\frac{n}{2^r}$ nodes of rank r.



n total elements

Amortized Time (Rank w/ Path Compression)

Lets $0, \ldots X$ be a set of buckets

Put every non-root node in a bucket by rank!

Invent buckets

Structure buckets to store ranks $[r, 2^r - 1]$

$0, 2^0-1$

$1, 2^1-1$

$2, 2^2-1$

Ranks	Bucket
0	0
1	1
2 - 3	2
4 - 15	3
16 – 65535	4
65536 – 2^{65536}-1	5

6

Grows very very slowly $\log^*(n)$

After class question:

This partition makes $\log^*$ buckets!

(By design)

2^{65536} = larger than atoms in universe.

Iterated Logarithm Function (log^*n)

log^*n is piecewise defined as

$$0 \text{ if } n \leq 1$$

otherwise

$$1 + log^*(\log n)$$

the # of times we need to call log to get $n \leq 1$

Amortized Time (Rank w/ Path Compression)

Let $|B_r|$ be the size of the bucket with min rank r.

What is $max(|B_r|)$?

Ranks	Bucket
0	0
1	1
2 - 3	2
4 - 15	3
16 – 65535	4
65536 – 2^{65536}-1	5

r=0: All items w rank 0

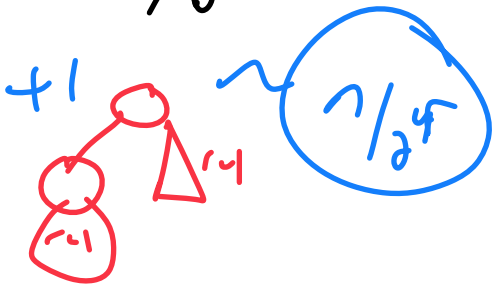
r=1: n/2

r=2: n/4

1) We can sum all possible ranks in each bucket

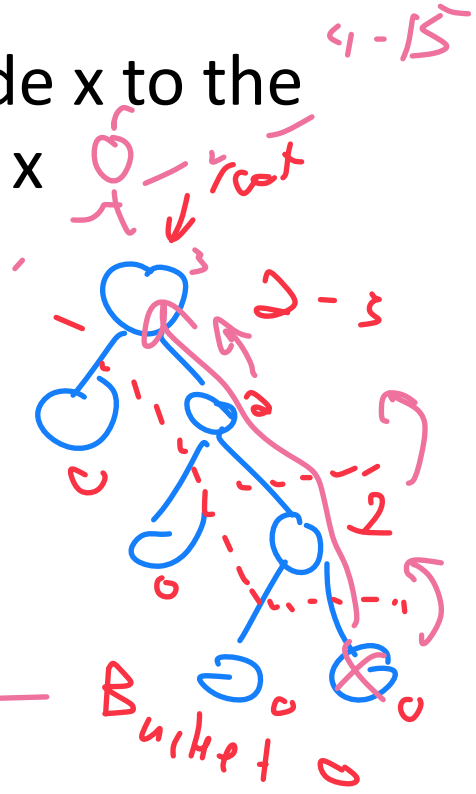
$n/2^r + n/2^{r+1} + n/2^{r+2} \dots n/2^{2^r-1} \simeq 2n/2^r$

2) To get rank r, I combined two r-1 items



Amortized Time (Rank w/ Path Compression)

The work of **find(x)** are the steps taken on the path from a node x to the root (or immediate child of the root) of the UpTree containing x



We can split this into two cases:

Case 1: We take a step from one bucket to another bucket.

↳ Easy!

At most 6

Every find will do this work

2m finds

$O(\log^* n)$

$m \log^* n$

$\Sigma O(1)$

Case 2: We take a step from one item to another inside the same bucket.

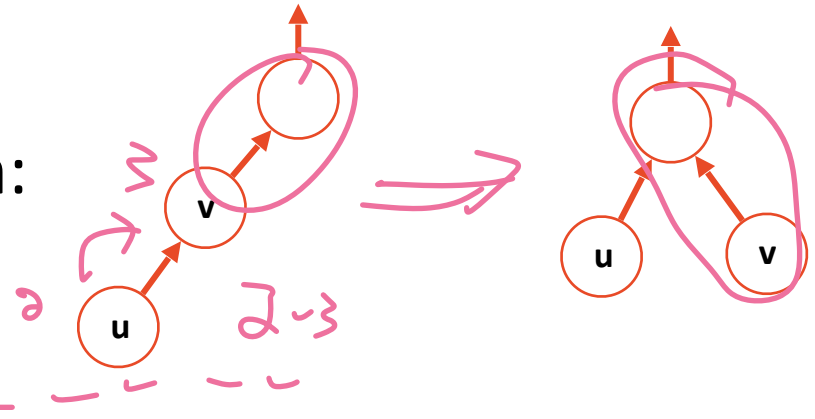
Amortized Time (Rank w/ Path Compression)

Case 2: We take a step from one item to another *inside* the same bucket.

Let's call this the step from **u** to **v**.

Every time we do this, we do path compression:

We set parent(u) a little closer to root



How many total times can I do this for each **u** in $|B_r|$?

B/c parent has higher rank

# of ranks in my bucket

Buckets

How many nodes are in $|B_r|$?

Each node (n/b) can increment at most 2^r

$\hookrightarrow$ n total work

$n \log^* n$

Final Result

We have **n items** in an Uptree. We make **m find()** calls. Total work is:

Randomized Algorithms

A **randomized algorithm** is one which uses a source of randomness somewhere in its implementation.

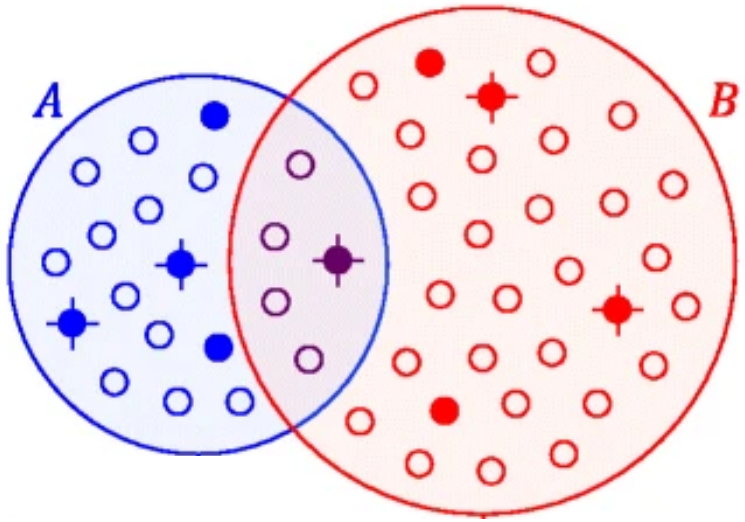
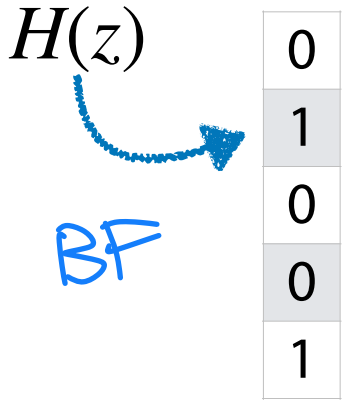
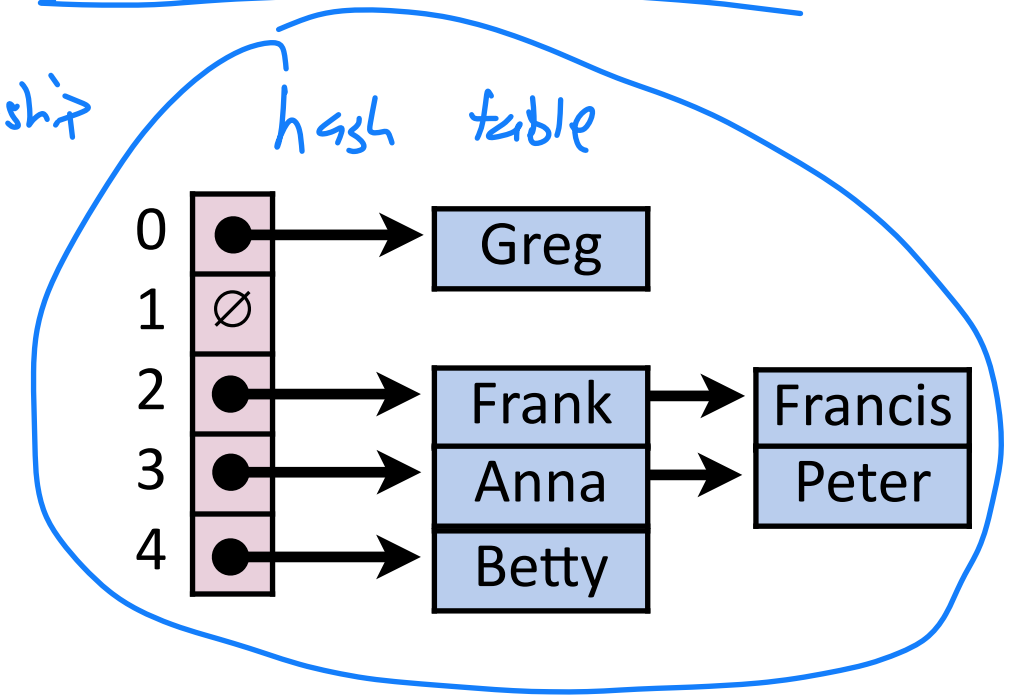


Figure from Ondov et al 2016





$H(x)$	0	2	1	0	0	4	0	2	0	6
$H(y)$	1	0	2	3	1	0	3	4	0	1
$H(z)$	2	1	0	2	0	1	0	0	7	2

A faulty list

Imagine you have a list ADT implementation ***except***…

Every time you called **insert**, it would fail 50% of the time.

add on a coin

flip

website caching

SC rambling data

Buffering, streaming out some data

Encryption → lossy! → we don't want random

A lot!

Approximate counting! ~ 1/2 real count

Quick Primes with Fermat's Primality Test

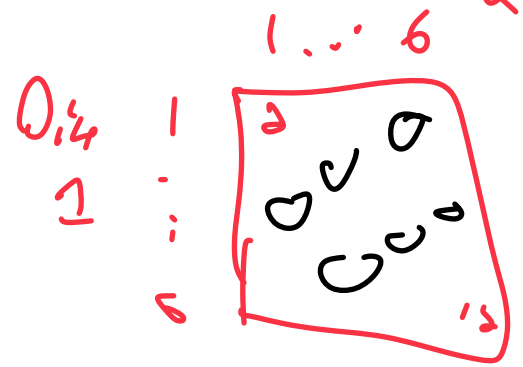
If p is prime and a is not divisible by p, then $a^{p-1} \equiv 1 \pmod{p}$

But… ***sometimes*** if n is composite and $a^{n-1} \equiv 1 \pmod{n}$

Fundamentals of Probability

Imagine you roll a pair of six-sided dice.

The **sample space** Ω is the set of all possible outcomes.



An **event** $E \subseteq \Omega$ is any subset.

Fundamentals of Probability

We can use most coop

Expectation

Imagine you roll a pair of six-sided dice. What is the expected value?

A **random variable** is a function from events to numeric values.

Let D_1 be the value of the first dice

D_2 second dice

↓ simplification?

The **expectation** of a (discrete) random variable is:

$$E[X] = \sum_{x \in \Omega} Pr\{X = x\} \cdot x$$

prob of one state

value of the state

sum of states not R.V.

sum over sample space

$$E[D_1] = \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + 4/6 + 5/6 + 6/6$$

$$= 3.5$$

$$E[D_1 + D_2] = \frac{1}{36}(1+1) + \frac{1}{36}(1+2) \ldots$$

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X+Y] = E[X] + E[Y]$$ **(Claim)**

$$= \sum_x \sum_y Pr(X=x, Y=y) \cdot (x+y)$$

prob event · value

split

$$= \sum_x x \sum_y Pr(X=x, Y=y) + \sum_y y \sum_x Pr(X=x, Y=y)$$

all prob sum to 1

$$Pr(X=x) \cdot x \sum_x \text{Prob}(Y=y)$$

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X+Y] = E[X] + E[Y]$$

$$E[X+Y] = \sum_x \sum_y Pr\{X=x, Y=y\}(x+y)$$

split in half

$$= \sum_x x \sum_y Pr\{X=x, Y=y\} + \sum_y y \sum_x Pr\{X=x, Y=y\}$$

$$= \sum_x x \cdot Pr\{X=x\} + \sum_y y \cdot Pr\{Y=y\}$$

$E[X]$ + $E[Y]$

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X + Y] = E[X] + E[Y]$$

Randomization in Algorithms

1. Assume input data is random to estimate average-case performance

2. Use randomness inside algorithm to estimate expected running time

3. Use randomness inside algorithm to approximate solution in fixed time