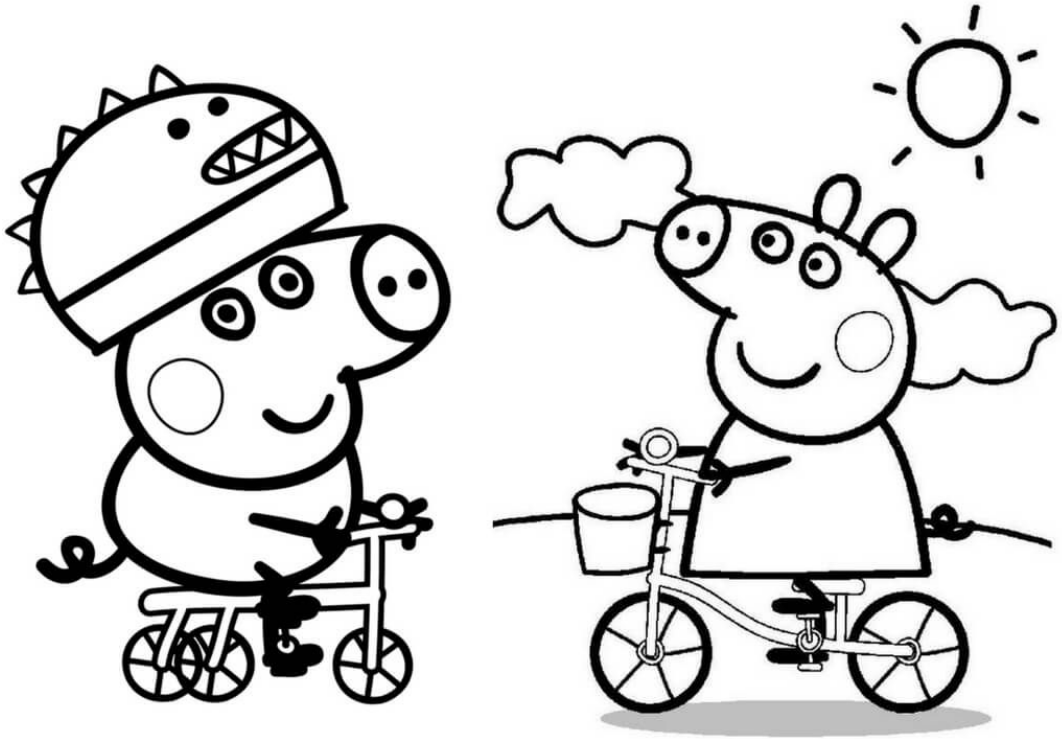
Big kids color too :)



We'll start at Berkeley time, till then feel free to color using the Zoom annotate features or ask me anything you'd like :)

Disjoint Sets, Asymptotics

Discussion 06

Announcements

Week 6

- ❏ Weekly Survey 5 - due on Monday 2/22
- ❏ Regrade requests - due on Monday 2/22
- ❏ Snaps Project 1 checkoff - due on Tuesday 2/23
- ❏ Lab 6 - due on Friday 2/26
- ❏ Start! Project! 2! ASAP!

Content Review

Disjoint Sets, also known as Union Find

```
public interface DisjointSet {
     void connect (x, y); // Connects nodes x and y (you may also see union)
     boolean isConnected(x, y); // Returns true if x and y are connected
}
```

QuickFind uses an array of integers to track which set each element belongs to.

QuickUnion stores the parent of each node rather than the set to which it belongs and merges sets by setting the parent of one root to the other.

WeightedQuickUnion does the same as QuickUnion except it decides which set is merged into which by size, reducing stringiness.

WeightedQuickUnion with Path Compression sets the parent of each node to the set's root whenever find() is called on it.

Disjoint Sets

```
public interface DisjointSet {
     void connect (x, y); // Connects nodes x and y (you may also see union)
     boolean isConnected(x, y); // Returns true if x and y are connected
}
```

Implementation	Constructor	connect()	isConnected()
QuickUnion	Θ(N)	O(N)	O(N)
QuickFind	Θ(N)	O(N)	O(1)
Weighted Quick Union	Θ(N)	O(log N)	O(log N)
WQU with Path Compression	Θ(N)	O(log N) Θ(1)*	O(log N) Θ(1)*

Asymptotics

Asymptotics allow us to evaluate the performance of our programs using math. We ignore all constants and only care about the value with reference to the input (usually defined as N)

Big O - The upper bound in terms of the input. In other words, if a function has big O in f(x), we say that it could grow at most as fast as f(x), but it could grow more slowly.
Big Ω - The lower bound in terms of the input. In other words, if a function has big Ω in f(x), we say that it could grow at least as slowly as f(x), but it could grow more quickly.
Big Θ - The tightest bound, which only exists when the tightest upper bound and the tightest lower bound converge to the same value.

Fun sums:

$1 + 2 + 3 + \ldots + N = \Theta(N^2)$

$1 + 2 + 4 + \ldots + N = \Theta(N)$

Worksheet

1A Disjoint Sets What are the last 2 improvements we made to Union Find?

Improvement 1:

Improvement 2:

1B Disjoint Sets Draw the union find tree and underlying array.

```
connect(2, 3);
connect(1, 2);
connect(5, 7);
connect(8, 4);
connect(7, 2);
find(3);
connect(0, 6);
connect(6, 4);
connect(6, 3);
find(8);
find(6);
```

1C Disjoint Sets (extra) Let's do it with path compression!

```
connect(2, 3);
connect(1, 2);
connect(5, 7);
connect(8, 4);
connect(7, 2);
find(3);
connect(0, 6);
connect(6, 4);
connect(6, 3);
find(8);
find(6);
```

1D Disjoint Sets

What is the runtime for "connect" and "isConnected" operations using our Quick Find, Quick Union, and Weighted Quick Union ADTs?

Can you explain why the Weighted Quick union has better runtimes for these operations than the regular Quick Union?

2A Asymptotics Order the following from smallest to largest.

$O(\log n)$, $O(1)$, $O(n^n)$, $O(n^3)$, $O(n \log n)$, $O(n)$, $O(n!)$, $O(2^n)$, $O(n^2 \log n)$

2B Asymptotics Are these bounds accurate? Are they tight bounds?

$f(n) = 20501$	$g(n) = 1$	$f(b) \in O(g(n))$
$f(n) = n^2 + n$	$g(n) = 0.000001n^3$	$f(b) \in \Omega(g(n))$
$f(n) = 2^{2n} + 1000$	$g(n) = 4^n + n^{1000}$	$f(b) \in O(g(n))$
$f(n) = \log(n^{100})$	$g(n) = n\log n$	$f(b) \in \Theta(g(n))$
$f(n) = n\log n + 3^n + n$	$g(n) = n^2 + n + \log n$	$f(b) \in \Omega(g(n))$
$f(n) = n\log n + n^2$	$g(n) = \log n + n^2$	$f(b) \in \Theta(g(n))$
$f(n) = n\log n$	$g(n) = (\log n)^2$	$f(b) \in O(g(n))$

2C Asymptotics What are the best and worst case runtimes?

```
for (int i = N; i > 0; i--) {
     for (int j = 0; j <= M; j++) {
          if (ping(i, j) > 64) {
               break;
          }
     }
}
```

2D Asymptotics What is the runtime?

```
public static boolean noUniques(int[] array) {
     array = sort(array);
     int N = array.length;
     for (int i = 0; i < N; i+= 1) {
          boolean hasDuplicate = false;
          for (int j = 0; j < N; j += 1) {
               if (i != j && array[i] == array[j]) {
                    hasDuplicate = true;
               }
          }
          if (!hasDuplicate) {
               return false;
          }
     }
     return true;
}
```

2D Asymptotics

Can we do it in NlogN?