CS-21 LECTURE #7

	Max	Mean	Median
Program #2:	97/100	65	70
Program #3:	120/100	82	89

- new - delete
- new[] - delete[]
- only inline appropriate functions
- Include Makefile
- Do lots of bounds checking!!!
- 'ws' = whitespace
- DONT USE STRINGS. USE INTS!!!! (~~<iomanip>~~) <iomanip>

call it that!

```
quicksort (vector<int> & A)

quicksort (A, 0, A.size()-1);
```

"which one is A?"

In C++, things are identified by things pointing to them!

NOT GOOD: → `int a[10];`

GOOD: → `int *a = new int[10]`

DISJOINT SET (NOT ON EXAM)

- Used to group n items into 1 to n sets...

initial: n items in n sets

over time: Union sets together to get fewer sets

"Stop" state: 1 set

Tasks (what we will ask of an item)

1. which set do you belong to?
2. Union 2 sets together

DISJOINT SET:

Union/find datastructure: (subsumed into the constructor "make_set" all sets at once)

public (bounds check)

- make_set(x) → A new set with x as its only member.
- Union(x,y) → Union sets containing x & y into one new set
- find(x) → (AKA find_set) return unique representative of set that contains x
- bool same_component(x,y) → return Find(x) == Find(y)
- int numSets() → return # of current sets.

How to implement?

BAD: Linked List for each set - SLOW $O(n)$ worst case

GOOD: Disjoint set forest (Just two arrays) - FAST!

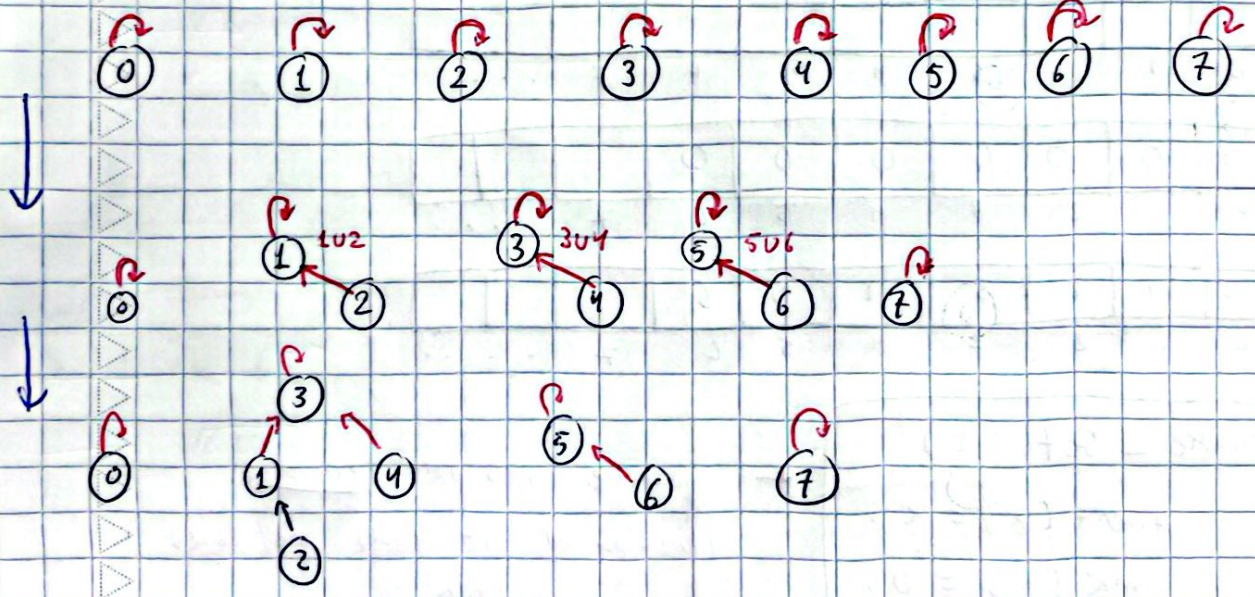
- Expect/average = $O(1)$
- worst case = $O(n)$

"amortized"

Heuristics = makes the best decision it can with limited info

- Union by rank
- Path compression

Sets are named after the member @ the root of the tree that traces that set.

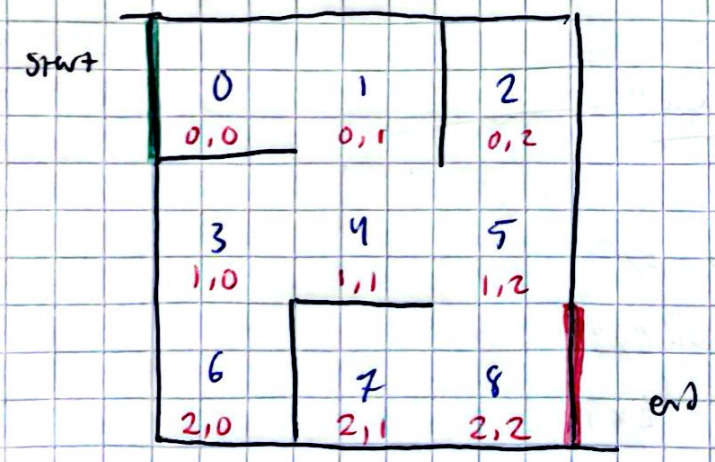


PUBLIC — Find with path compression...

```
Find(x) {
    // bounds check!!!
    if( x != parent[x] ) {
        parent[x] = find( parent[x] );
    }
    return parent[x];
}
```

We're going to generate a maze with this!!!!!

generate Maze (n x n)



numbering scheme...

you can either manage the ~~size~~ by 1 dimension or 2 dimensions

- we use the disjoint set to generate the walls of the mazes
- A wall is like a set, you union ~~removed~~ sets together to create a final wall setup

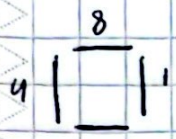
maze ← class 1 — makes disjoint set, has n x n data, keeps track of walls,

Disjoint set ← class 2

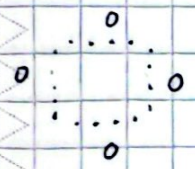
DO NOT MIX THE CLASSES

How do we represent the maze?

Each square = Hex value 0 - F



All of the walls? val = 15 so represented by "F"



No walls? val = 0 so represented by "0"

Make vector or array w/ every index in it & randomly shuffle it. To make sure you go every square once.