CSCI 270 Lecture 3: Minimum Spanning Trees

New data structure: Union-Find. Maintain one set for each connected component, consisting of all nodes in that component.

Union(x,y) combines the sets x and y into a single set.

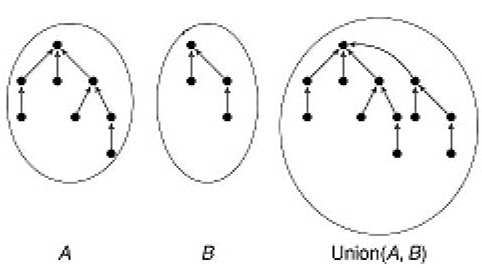
Find(x) determines which set contains node x. If Find(x) =Find(y), you don't want to add the edge (x, y).

```
Kruskal(V,E)
Sort E in ascending order of weight.
For each edge (x, y) in ascending order of weight {
    a = Find(x)
    b = Find(y)
    If a ≠ b Then {
        Union(a, b)
        Add edge (x, y)
}   }
```

Attempt 1: Each node has a variable which stores the name of the set which it is in.

- How long would Find take?
- How long would Union take?

Attempt 2: Each set is identified by a specific node, whom we'll refer to as the "captain" of the set. Each node has a pointer to another node in the set. The captain's pointer will be null.



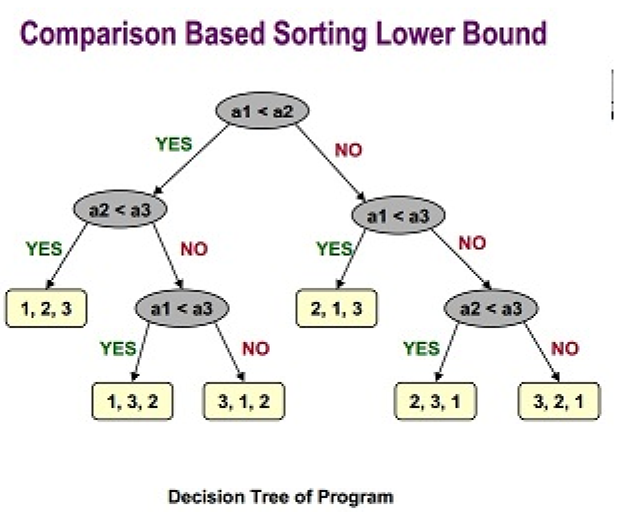
- How long would Union take, assuming we pass in the captains of sets?
- How long would Find take?
- The worst-case analysis of Find requires a rather stupid decision. How could we improve our Union-Find data structure?
- If we do a union between trees of depth x and y, where $x < y$, what is the depth of the new tree?

- When would a union produce a tree of greater depth?
- What is the depth of a 1,2, or 3 node tree?
- What is the greatest depth possible for a 4 node tree?
- How many nodes are required to produce a depth-3 tree?
- How many nodes are required to produce a depth-4 tree?
- What is the runtime of Find?
- What is the runtime of Union, assuming we pass in set captains?
- Putting it all together: what is the runtime of Kruskal's Algorithm?
- Why does the book give a different runtime analysis for Kruskal's algorithm?

To get a better runtime for Kruskal's, we would need to find a better sorting algorithm. MergeSort obtains $\theta(n \log n)$: is it possible to beat this?

Claim: Comparison-based sorting takes $\Omega(n \log n)$ time.

A sorting algorithm is **comparison-based** if we make comparisons between elements in the array-to-be-sorted, and determine the sorted order based on those comparisons.

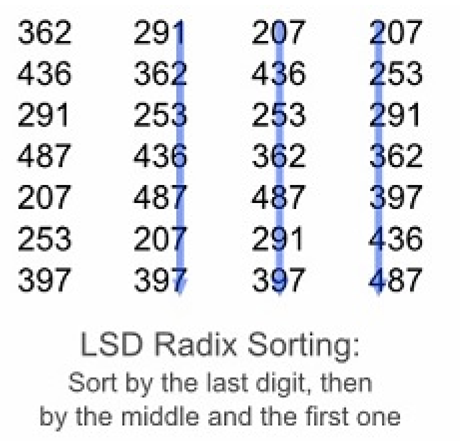


- When we reach a leaf node in a decision tree, what do we know?

- How many leaf nodes must there be when we wish to sort n numbers?

- How deep must the tree be to achieve this?

- Is there such a thing as a sorting algorithm which is **not** comparison-based?



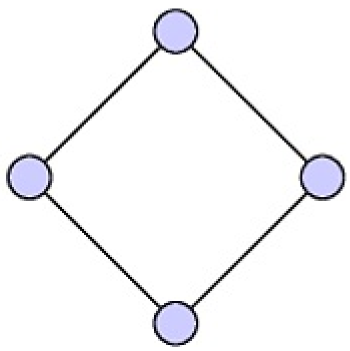
- What is the runtime of Radix Sort?

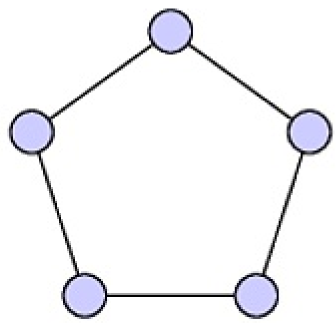
- Is this better than MergeSort?

- What is the best achieveable runtime to sort an array of size n, which contains the numbers 1 through n randomly permuted?

Graphs

A graph is **bipartite** if nodes can be colored red or blue such that all edges have one red end and one blue end.





Bipartite Graph Questions:

- What was wrong with the second example? Why was it impossible to color correctly?
- Is the converse of that statement necessarily true?
- How could one write an algorithm to test whether a graph is bipartite? This should be based on a basic search algorithm.