# CSC263 Tutorial #11

## MSTs

March 31, 2023

# Things covered in this tutorial

- ⋆ What's the Disjoint Set ADT?
- ⋆ How is it implemented using trees?

# Disjoint Sets

The Disjoint Set ADT stores a collection of nonempty sets with disjoint elements $S_1, \ldots, S_k$. Each set has a representative.

**Question:** What operations are supported by the Disjoint Set ADT?

# Disjoint Sets

The Disjoint Set ADT stores a collection of nonempty sets with disjoint elements $S_1, \ldots, S_k$. Each set has a representative.

**Question:** What operations are supported by the Disjoint Set ADT?

**Answer:**

# Disjoint Sets

The Disjoint Set ADT stores a collection of nonempty sets with disjoint elements $S_1, \ldots, S_k$. Each set has a representative.

**Question:** What operations are supported by the Disjoint Set ADT?

**Answer:**

- ⋆ `MakeSet(`$x$`)`: Add a singleton set $\{x\}$ to the collection of disjoint sets.

# Disjoint Sets

The Disjoint Set ADT stores a collection of nonempty sets with disjoint elements $S_1, \ldots, S_k$. Each set has a representative.

**Question:** What operations are supported by the Disjoint Set ADT?

**Answer:**

- $\star$ `MakeSet(`$x$`)`: Add a singleton set $\{x\}$ to the collection of disjoint sets.
- $\star$ `FindSet(`$x$`)`: Return the representative of the set containing $x$.

# Disjoint Sets

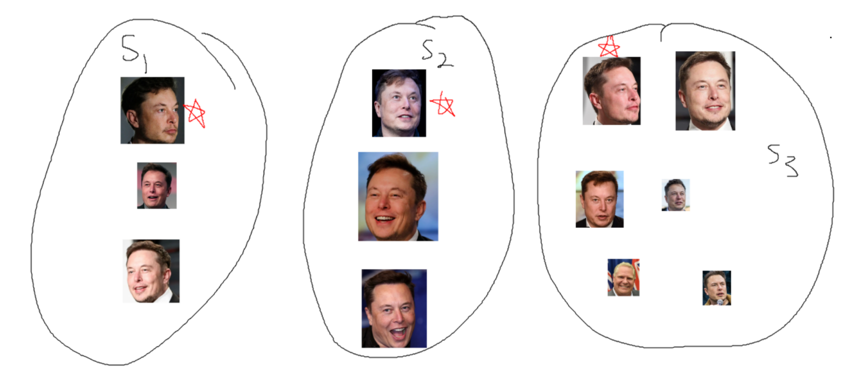
The Disjoint Set ADT stores a collection of nonempty sets with disjoint elements $S_1, \ldots, S_k$. Each set has a representative.

**Question:** What operations are supported by the Disjoint Set ADT?

**Answer:**

- ⋆ `MakeSet(`$x$`)`: Add a singleton set $\{x\}$ to the collection of disjoint sets.
- ⋆ `FindSet(`$x$`)`: Return the representative of the set containing $x$.
- ⋆ `Union(`$x$`, `$y$`)`: Combine the two disjoint sets $S_i$ containing $x$ and $S_j$ containing $y$ respectively (and ignore if $S_i = S_j$).

# Disjoint Sets



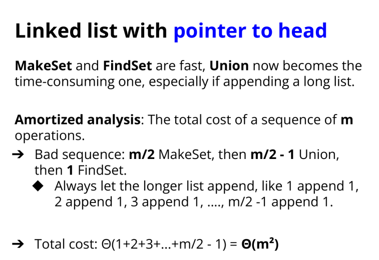
# Disjoint Sets

**Task:** Apply the following disjoint set operations, and draw the result as shown in the previous slide.

- `MakeSet(Chungi)`
- `MakeSet(Winni)`
- `MakeSet(Bob)`
- `MakeSet(727)`
- `MakeSet(Elong)`
- `MakeSet(420)`
- `MakeSet(Mogus)`
- `Union(420, Mogus)`
- `Union(Mogus, Bob)`
- `Union(Winni, 727)`
- `Union(Winni, Bob)`
- `Union(Winni, Chungi)`

# Disjoint Sets with Trees!

Recall how we tried to implement disjoint sets with linked lists, but that wasn't very efficient...

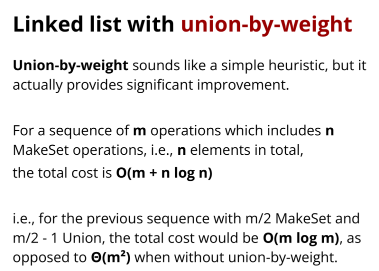


## Linked list with pointer to head

**MakeSet** and **FindSet** are fast, **Union** now becomes the time-consuming one, especially if appending a long list.

**Amortized analysis**: The total cost of a sequence of **m** operations.

- ➔ Bad sequence: **m/2** MakeSet, then **m/2 - 1** Union, then **1** FindSet.
  - ◆ Always let the longer list append, like 1 append 1, 2 append 1, 3 append 1, …., m/2 -1 append 1.
- ➔ Total cost: $\Theta(1+2+3+...+m/2 - 1) = \boldsymbol{\Theta(m^2)}$



## Linked list with union-by-weight

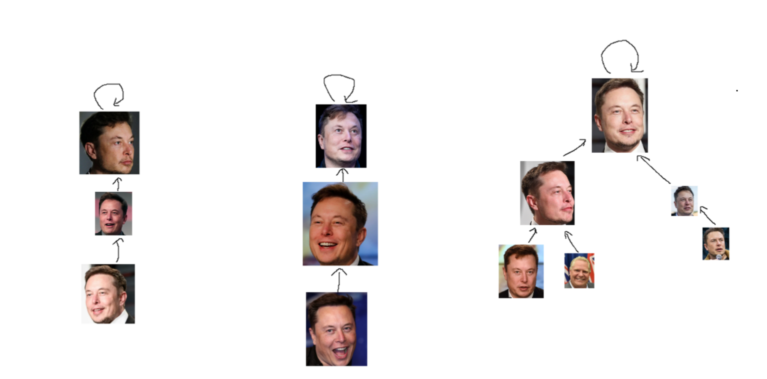
**Union-by-weight** sounds like a simple heuristic, but it actually provides significant improvement.

For a sequence of **m** operations which includes **n** MakeSet operations, i.e., **n** elements in total,
the total cost is **$O(m + n \log n)$**

i.e., for the previous sequence with m/2 MakeSet and m/2 - 1 Union, the total cost would be **$O(m \log m)$**, as opposed to **$\Theta(m^2)$** when without union-by-weight.

# Disjoint Sets with Trees!

We implement Disjoint Set using an "Inverted Tree":



We use two tricks to speed up the operations: Union By Rank and Path Compression.

# Disjoint Sets with Trees!

**Task:** Using an inverted tree implementation with Union by Rank and Path Compression, apply the following disjoint set operations.

- `MakeSet(C)`
- `MakeSet(W)`
- `MakeSet(B)`
- `MakeSet(7)`
- `MakeSet(E)`
- `MakeSet(4)`
- `MakeSet(M)`
- `Union(4, M)`
- `Union(M, B)`
- `Union(W, 7)`
- `Union(7, M)`
- `Findset(7)`
- `Findset(M)`
- `Union(W, C)`

# Disjoint Sets with Trees!

**Task:** Complete the tutorial activity! (Why can't we just traverse an inverted tree?)

# Disjoint Sets with Trees!

**Task:** Complete the tutorial activity! (Why can't we just traverse an inverted tree?)

*Hint:* The two attributes needed are named `node.next` and `node.tail` respectively.

# Goodbye!

Good luck on exams!