Every problem looks like a graph problem if you stare at it hard enough. Consider sorting an array of length \( n \). We can treat each input as an element of \( S_n \). Then we can make a graph \( G = (S_n, E) \) where \( E = \{ u \rightarrow v | u, v \in S_n \text{ and } u \neq v \} \) swap differences.

If we are asked to sort \( A \in S_n \), then we start at vertex \( A \) and attempt to find the identity permutation. The path we take represents the swaps we make, and the length of the path is the number of swaps.

**Example**

\( n = 3 \)

\( A = (231) \)

- Swap 1, 2
- Swap 2, 3

If we take a step back, there's a more general problem here.

**Reachability**

The graph problem is to determine if there is a path from \( u \) to \( v \) in a graph. There are many algorithms which do this under different constraints. Each has a different space and time complexity. Reachability is one of the few problems that require only NL space.

**NL Reachability** \( (G, s, t) \)

Let \( u = s \)
Let \( i = 0 \)
While \( u \neq t \land i \leq |V| \)
- Pick a \( v \) such that \((u, v) \in E\)
  - Let \( u = v \)
  - \( i++ \)
Return \( i \neq |V| \)

The maximum simple path length is \( |V| - 1 \) edges.
This is a sneak preview of nondeterminism. So long as any execution path of NReachability returns true, we say there is a path \( s \rightarrow t \). If no execution path returns true, we say there is no path \( s \rightarrow t \). In essence nondeterminism checks if \( \exists p \) or if \( \forall p, \neg p \).

We can turn this into a clever deterministic algorithm.

Reachability \((G, s, t, k)\)

- If \( k = 0 \)
  - Return \( s = t \)
- If \( k = 1 \)
  - Return \((s, t) \in E\)
- For \( u \in V \)
  - If Reachability \((G, s, u, \lfloor \frac{k}{2} \rfloor) \land \text{Reachability} (G, u, t, \lceil \frac{k}{2} \rceil)\)
    - Return True
- Return False

Notice that we are only storing \( u \) and \( k \) memory at any recursive call, and the recursive depth is \( \log_2 k \), so we only need \( \log_2 k \) of them. We make \( k = |V| \) initially, b/c the maximum simple path length in \( G \) has \( |V| \) vertices. Thus we need \( \log_2 |V| \) pieces of memory of \( \log_2 |V| \) size (2 integers of value at most \(|V|\)) for \( \log^2 |V| \) total memory.

How long does this algorithm take?

\[ T(n) = 2nT(\frac{n}{2}) + \Theta(n) \]

The MIT does not want \( a = 2n \) as a function of \( n \), so we need a work tree.
So Reachability is $O(n \log^2 n)$, which is very bad. But on the upside, its space complexity is $O(\log^2 n)$. However, maybe we should instead trade some extra memory usage for a vast speedup.