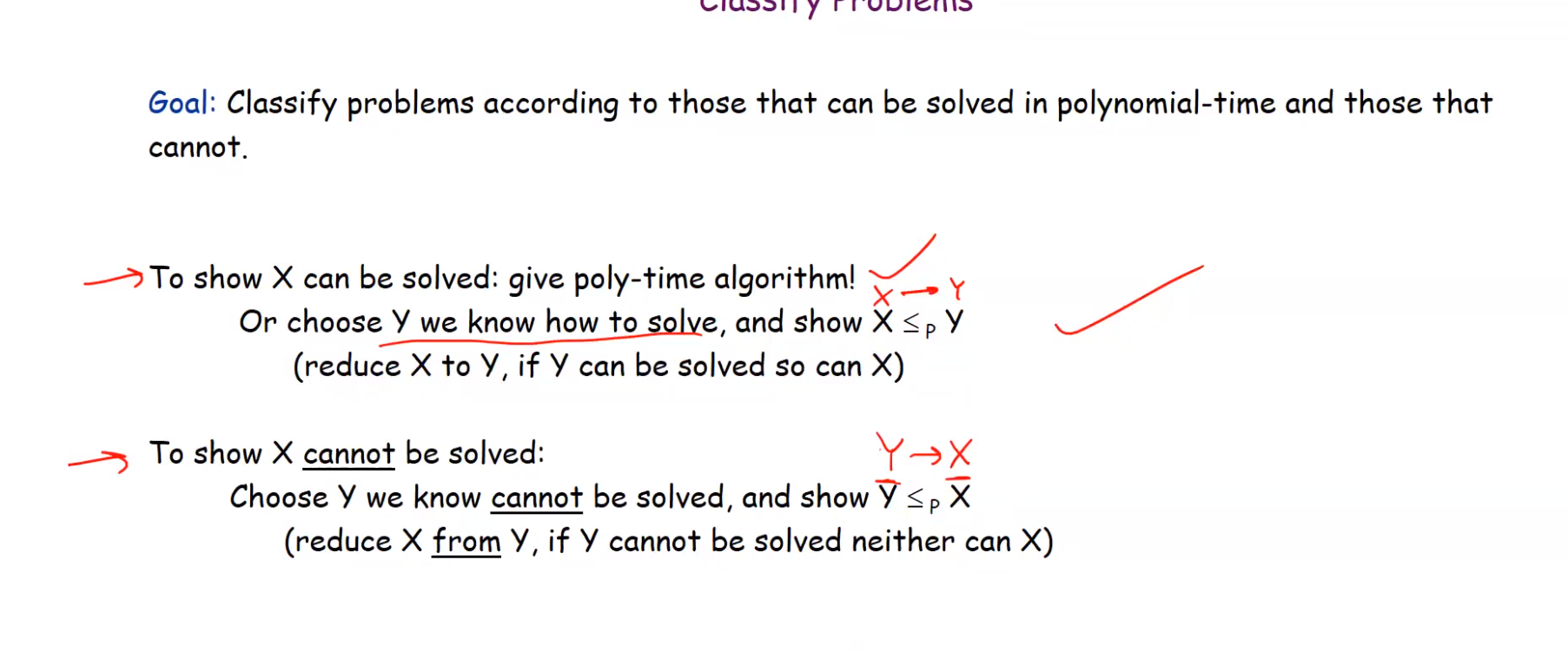
* Dijktra’s Algo
  + Runtime: Assume O(n2)
  + Use: Find the shortest distance to each node in a graph
* Kosaraju’s Algo
  + Runtime: O(n)
  + Use: Find the SCC’s of a graph
* Topological Sort
  + Runtime: O(n)
  + Use: Organizes a DAG with all edges pointing from left to right
* Kruskal’s Algorithm
  + Runtime: O(nlogn)
  + Use: Finds the minimum spanning tree in an undirected edge-weighted graph
* Prim’s Algorithm
  + Runtime: O(nlogn)
  + Use: Finds the minimum spanning tree in an undirected edge-weighted graph
* BFS
  + Runtime: O(V+E)
  + Shortest Path (Unweighted)
  + Shortest Cycle (Unweighted, Directed)
  + Can find Connected Components (A set of vertices in a graph that are linked to each other by paths)
* DFS
  + Runtime: O(V+E)
  + Shortest Path (Unweighted)
* NP-Complete problems to reduce from:
  + Independent Set
    - A set of k vertices that don’t have any edges connecting them
      * This algorithm finds maximum vertices you can select that aren’t connected to each other. You can also make a yes/no algorithm by asking “are there more than n vertices that are independent”.
  + Vertex Cover
    - A set of vertices that cover all the edges
      * This algorithm finds the minimum vertices that cover all the edges. You can make this a yes/no algorithm by asking “are there fewer than n vertices that are independent”.
    - Vertex Cover and Independent Set are closely related. (With v being the number of vertices) If you have an n size independent set, you’d have a v-n size vertex cover. And it’d just be all other vertices. And if you have an n size vertex cover, you’d have a v-n size independent set.
  + Clique
    - Literally just “is there a complete subgraph of size k?”
    - Set of vertices all connected
  + Rudrata Path (directed or undirected)
    - A path that visits every vertex once
  + Rudrata Cycle (directed or undirected)
    - A cycle that visits every vertex once
* Find median in linear time
  + K-selection
  + Find x-th element in linear time
* Merge Sort (Divide and conquer)
  + O(nlogn)
  + Split array into smaller arrays of size 2, sort and combine till back to original size
* Quick Sort
  + O(nlogn)
  + Worst Case: O(n^2)
  + Choose pivot and sort around the pivot
* Masters Theorem
  + <https://www.nayuki.io/page/master-theorem-solver-javascript>



* + Example:
    - The algorithm solves the problem by dividing them into four subproblems of size n/3 using 4nlog\_4(n) time, solving them recursively, and then combining the solutions in log\_4(n) time.
    - a = 4, b = 3, d < 1 + \e since nlogn is faster than n but slower than n^{1+\e}. Thus, T(n)=O(n^{log\_3(4)})
  + 
  + 