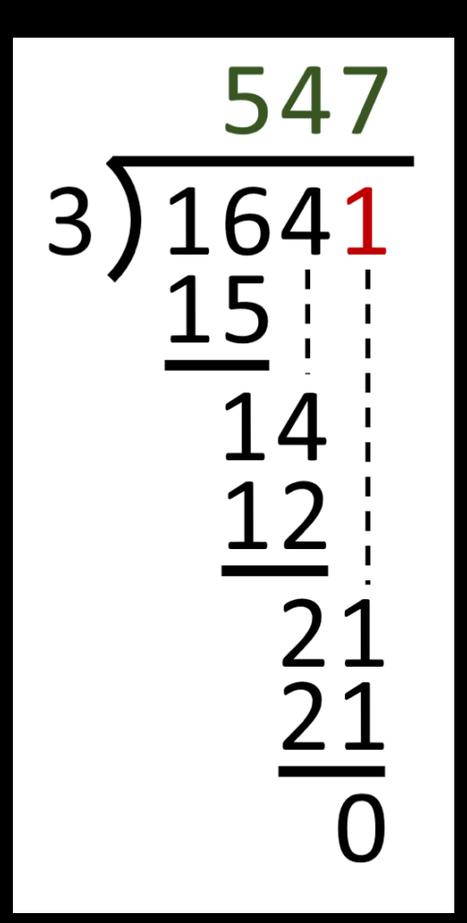
Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- We use algorithms (recipes) to solve problems.
  - Each *instance* of a problem is a unique set of inputs.
- For example, long division (on paper) is one algorithm for dividing two numbers
- Some instances of this problem are:
  - 100 ÷ 5, 8 ÷ 2, 33 ÷ 11



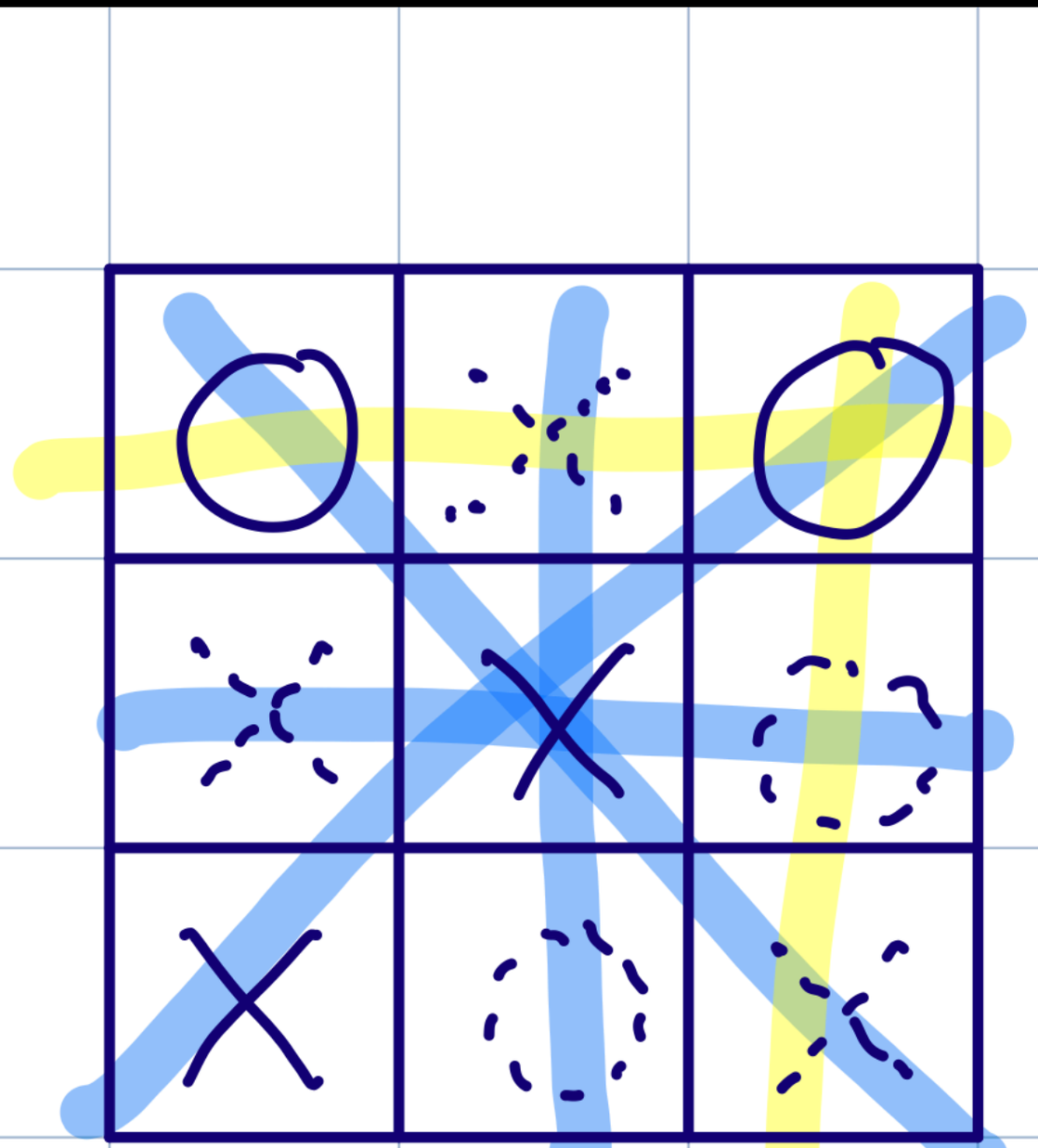
Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- Have you ever played Tic Tac Toe?
    - Play a game with a friend right now.
    - As you play, think out loud please.
    - What are you considering as you decide where to play next?

Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"



1

2

3

4

9 · 8 · 7 · ... · 2 · 1

9!

362,880

Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- Tic Tac Toe is an example of *factorial time* problem.
  - For the first move, there are 9 possible locations.
  - For the second move, there are 8 possible locations, and so on.
  - So, very roughly, O(n!)
  - It's beyond the scope of the course to formally analyze the complexity of Tic Tac Toe, but it's an example of a *superpolynomial* time problem.

Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- There are different categories of problems, too.
  - Delivery company scheduling drivers to deliver packages to final destinations on the most efficient route (like UPS or Fedex) ... these are *optimization* problems
- Some problems, like finding whether a given number is even, is easy – that is a *decision* problem.
  - An example of a hard decision problem is deciding whether two numbers have exactly two prime factors.
    - e.g.: 39 is a "yes" (3×13)
    - 32,616,811 is a "yes" (1321×24691)

Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- As numbers get larger, determining whether there are exactly two prime factors takes a long time, but it *is* something that will results in a yes or no answer.
- Some problems are *undecidable*.
  - The best example of this is analyzing a program – code verification.
    - For all possible inputs, will the program result in an infinite loop?
    - Or will it halt?

Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- You will read more about categories of problems in the suggested exercises, with a *lot* of detail.
  - It's *not required* to be able to identify whether a given problem is undecidable or not.
  - You simply need to be *aware of* the different categories of problems identified just now.

Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- Summary:
  - A *decidable* problem is a decision problem for which it is possible to obtain a yes or no answer.
  - An *undecidable* problem is one for which no algorithm can be constructed that is *always capable* of providing a yes or no answer.
    - An undecidable problem may have some instances that have a solution, but there is no algorithmic solution that could solve *all* instances of the problem.

Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- Now consider games like checkers, or chess.
- Analyzing all possible game outcomes from a single move (a brute-force approach) is way too much work to complete in a *reasonable* amount of time – even for a computer.
- A more common approach is to choose a move that *most likely* results in an advantageous position, but *might not* be the absolutely best move.
    - This is known as applying a *heuristic*.

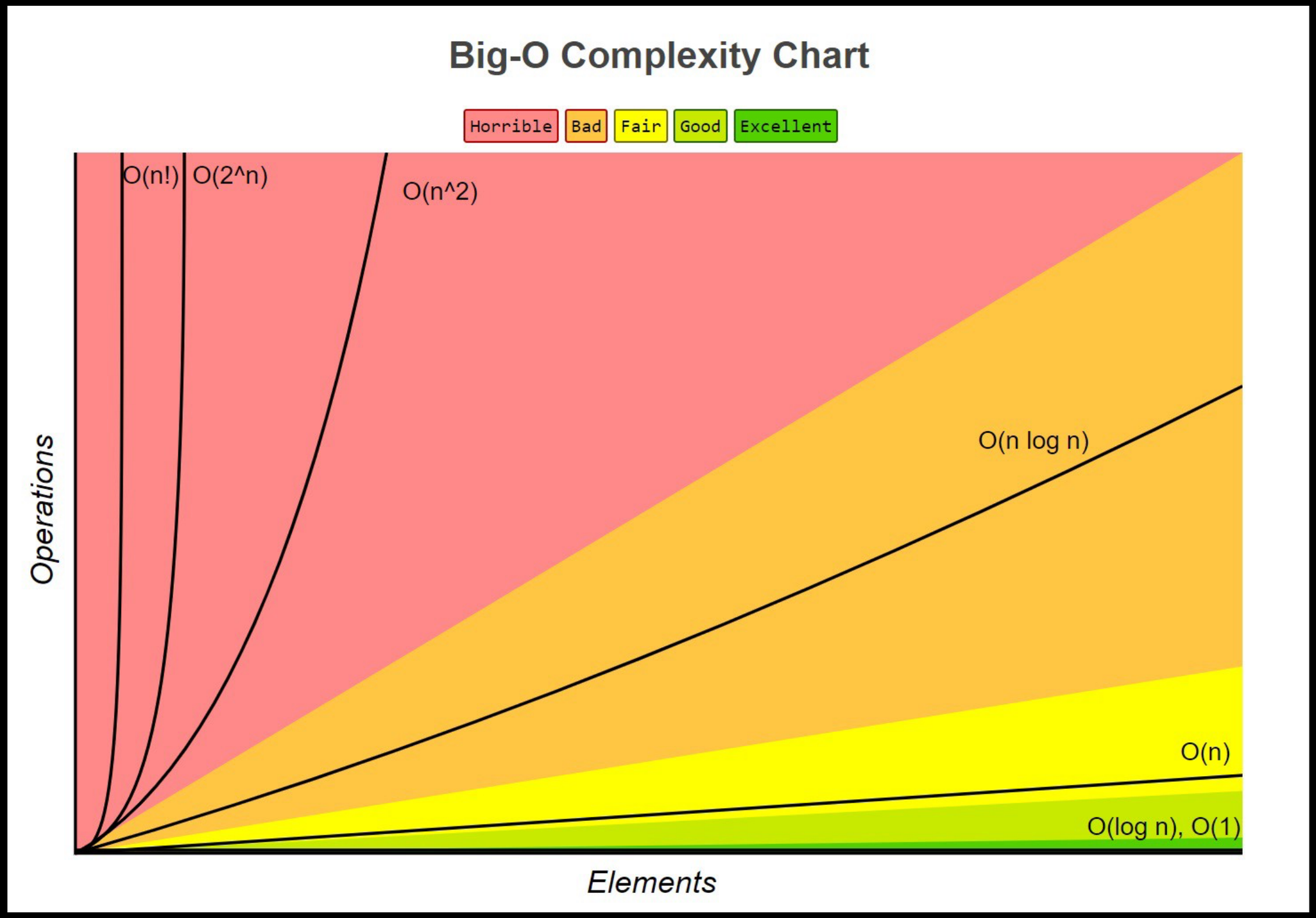
Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- Programs that can solve a problem in *reasonable* amount of time are those whose runtimes increase no more than a *polynomial* function of *n.*
  - O(1), O(log n), O(n), O(n log n), $O(n^2)$
  - $O(n^2)$ is not awesome for high values of n, but it's still classified as having *reasonable* time complexity.
- Programs that run in an *unreasonable* amount of time have *superpolynomial* time complexities.
  - $O(2^n)$, O(n!)

Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"



Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

- When all possible outcomes of a problem cannot be found in a *reasonable* amount of time, a heuristic approach can provide a "good enough" solution.
- There are many heuristics that are used for a variety of problems.
- This can be hard to imagine, so here's a nice illustration of heuristics that can be applied to a classic computer science dilemma:
    - "Travelling Salesperson Problem"

Heuristics

Don't Let "Perfect" be the Enemy of "Good Enough"

< Click to watch

Suggested Exercises

Heuristics & Undecidable problems

- From Khan Academy, finish this module and it's related quiz:
  - Solving hard problems
    - NOTE: Undecidable problems gives a great example, but with far more detail than you'll need to know for the AP CSP exam.