Example 1: Maximum Number

- **Problem.** Given a set of numbers $S$, find the largest number in $S$.
- **How can we approach such a problem?**

$1, 5, -3, 3.14, 100.1, 432, 123, 123.4$
• **Problem.** Given a set of numbers $S$, find the largest number in $S$.
  ◦ Variable $X$ - stores the largest number we found so far.

• **Algorithm.**
  ◦ Set $X$ to be the first number of $S$.
  ◦ Loop over every number of $S$:
    • If the current number $N$ is larger than $X$, replace the value of $X$ with $N$.
  ◦ Return $X$

1, 5, −3, 3.14, 100.1, 432, 123, 123.4

---

**Algorithm VS Computer Program**

• What is the difference between the algorithm and a computer program that implements the algorithm?
  ◦ The algorithm did not specify:
    • What programming language to use.
    • What type of loop is used to go over each number.
    • What structure stores the set of numbers.
    • Other similar technical details.
Algorithm “Definition”

- **Algorithm**: A sequence of instructions specifying how to solve a family of problems.
  - Similar to a **food recipe**.
  - Usually does not include technical implementation details.

Example 2: Cheapest Trip

- **Problem**: Given a list of possible flights, bus rides, and train rides, with prices.
  - Find the **cheapest trip** from city A to city B that takes **at most X hours**.
Cheapest Trip: Algorithm

- Naïve algorithm.
  - Go over all possible paths from A to B.
    - For path \( P \), check if it takes at most \( X \) hours.
    - If so, check if \( P \) is cheaper than cheapest path we found so far (as in Example 1).
    - If so, replace cheapest path we found so far with \( P \).
  - Should include a few more details:
    - How to enumerate every possible path.
    - At the end, return the cheapest path.

Issue with the Naïve Algorithm

- Issue. The number of possible paths from city A to city B is likely to be larger than the number of particles in the universe.
  - Even if we use the most efficient computer, programming language, and implementation, we will die a lot before the program ends.
  - We need a more efficient algorithm!
What is an Efficient Algorithm?

- **How do we check the efficiency of an algorithm?**
  - Different implementations of the algorithm may have different running times.
  - Different implementations may be faster for different inputs!

- **Is the concept of an “efficient algorithm” even well-defined?**
  - What if the algorithm is fast for most inputs but very slow for a few?
Example 3: Error Correcting Codes

- **Problem.** Computer A sends a message to computer B. Interference might cause a small number of errors in the message.

- **Algorithm needs to:**
  - **Detect** a small amount of errors.
  - **Correct** a small amount of errors.
  - **What else?**

  I love you
  
  I love lou
Example 4: Search Engine

• **Problem.** Design a search engine such as Google.

• **Algorithm needs to:**
  ◦ Run fast.
  ◦ **Find websites** that contain the search words.
  ◦ **Sort the sites** that were found by relevance to the user.

In this Course

• We will learn the basics of **designing efficient algorithms**.

• We will study how to **analyze the efficiency of an algorithm**.

• We will learn some well-known **real-world algorithms**.

• **Added bonus:** We will practice the type of thinking needed for various job interview riddles. ☺

• **Issue:** Too many bad jokes! 😞
Not in this Course

- We will **not do any programming**.
- Many useful algorithms require more advanced math, and we will try to avoid these.
  - Google’s PageRank algorithm requires some Linear Algebra so will not study it.
  - Error correcting codes require only the definition of vectors and matrices, so we might study these.

Course Prerequisites

- **Math 3300** or a similar programming experience.
- **Basic mathematical maturity.**
  - Designing efficient algorithms is *mathy*.
  - No advanced math knowledge is required. But you need to be comfortable with abstract thinking.
  - You will need to quickly get used to math concepts you may not be familiar with.
Course Grade

- Your grade will consist of
  - **Weekly assignments**. You are encouraged to work in groups, but **must** write solutions in your own words.
  - **Final and three midterms**. See dates on Syllabus.

Algorithmic Thinking

- Designing algorithms requires a getting used to a **different way of thinking**.
  - (Just like programming or proof writing require new ways of thinking.)
  - One might say that, the most important thing we learn in this course is algorithmic thinking.

"Science is a way of thinking much more than it is a body of knowledge"

Carl Sagan
Algorithmic Thinking

- Designing algorithms requires a getting used to a **different way of thinking**.

- **To practice algorithmic thinking:**
  - In classes we will often take **5-10 minutes to discuss in groups** how to solve a problem.
  - You need to spend time **thinking about the assignments**! Just seeing solutions is usually not enough to develop algorithmic thinking.

Any Administrative Questions?

*The secret behind Google’s search algorithm: pigeons*