CTD Intro Week 6

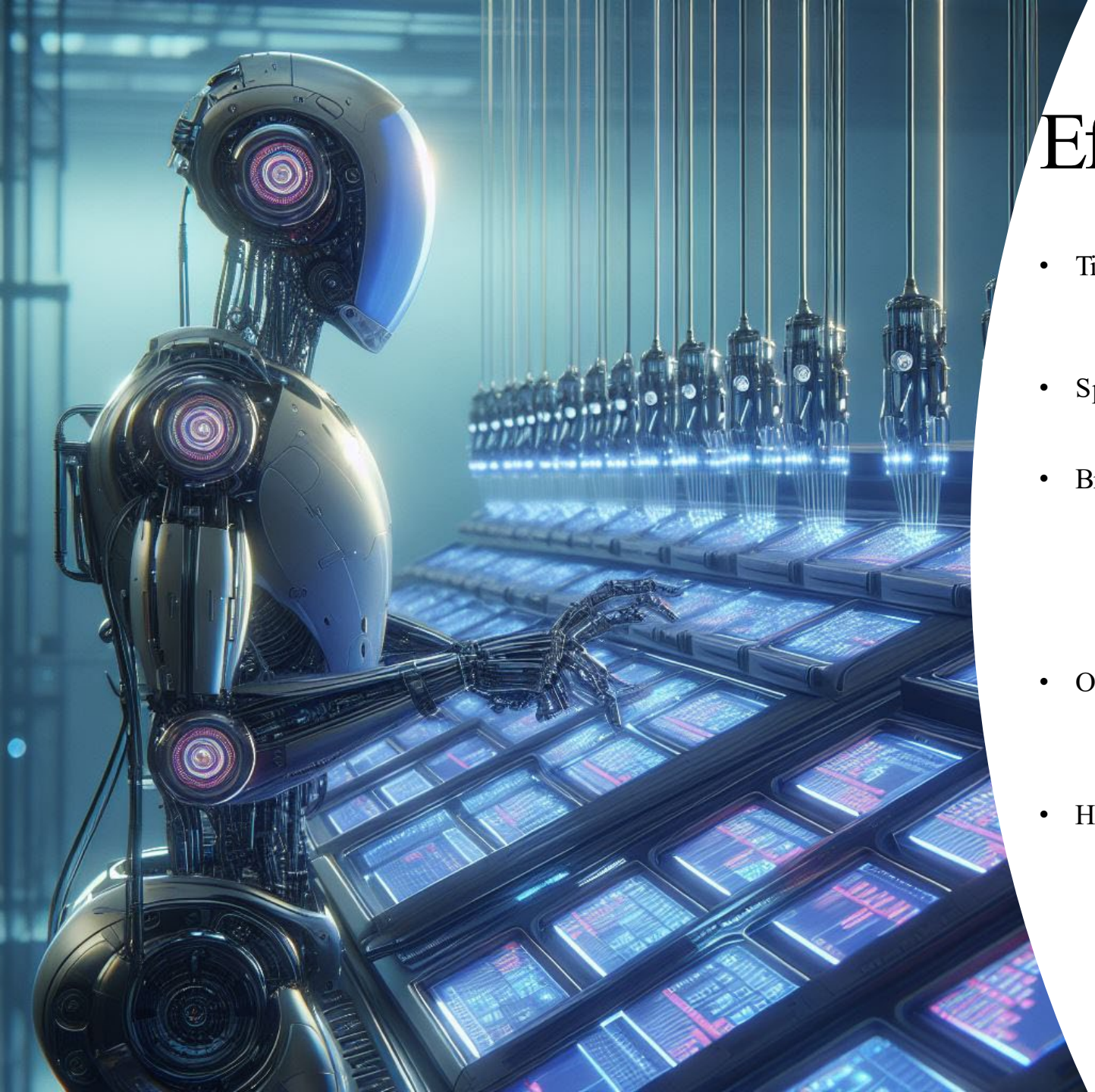
Introduction to Algorithms

What is an Algorithm?

- Definitions (many ways to put it, a few more examples)
  - Problem solving through systematic steps
  - Procedure to accomplish a specific task
  - The idea behind any reasonable computer program
- A well specified algorithmic problem includes:
  - Complete set of instances it must address
  - Constraints on inputs
  - Expectations for efficiency
- Pseudocode
  - Definition of steps without language specific details

History

- The word **“algorithm”** traces its roots to the **9th-century Persian mathematician**, **abu-Jaʿfar Mohammed ibn-Mūsa al-Khuwārizmi**.
- It found its way into language via a mangled transliteration of **al-Khwarizmi.**
- Modern usage started in the early 20th century, with common usage ramping up in the 1960s.
- Lots of fundamental work on algorithms comes from **Donald Knuth’s**, **“The Art of Computer Programming”**, started in 1962.

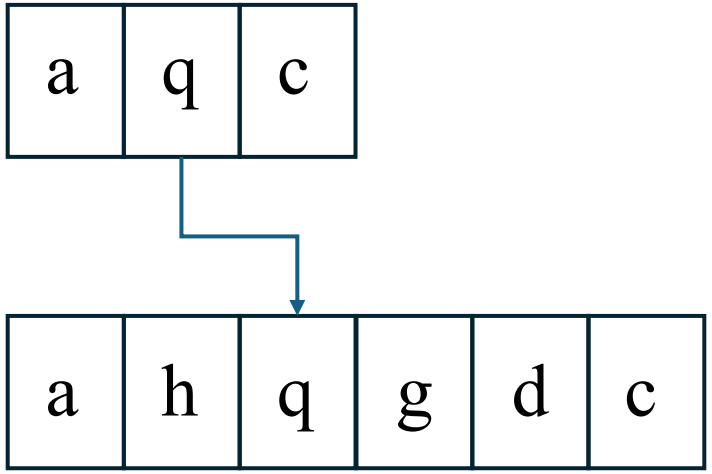


Efficiency

- Time Efficiency
  - How long does it take (how many steps) for a given problem size
- Space efficiency
  - How much memory does it take for a given problem size
- Big O notation is used to specify the efficiency of an algorithm
  - O(1) – constant space/time
  - O(n) – linear relationship to the problem size
  - $O(n^2)$ – Exponential relationship to problem size, slow for large problems
- Order of magnitude
  - Number of digits, floor of $\log_{10}$ of a number
  - Big O estimates are orders of magnitude
- How hard is the problem?
  - Potential efficiency is inherent in the problem
  - Active research to find more efficient solutions to a given problem

Reasoning about Algorithms

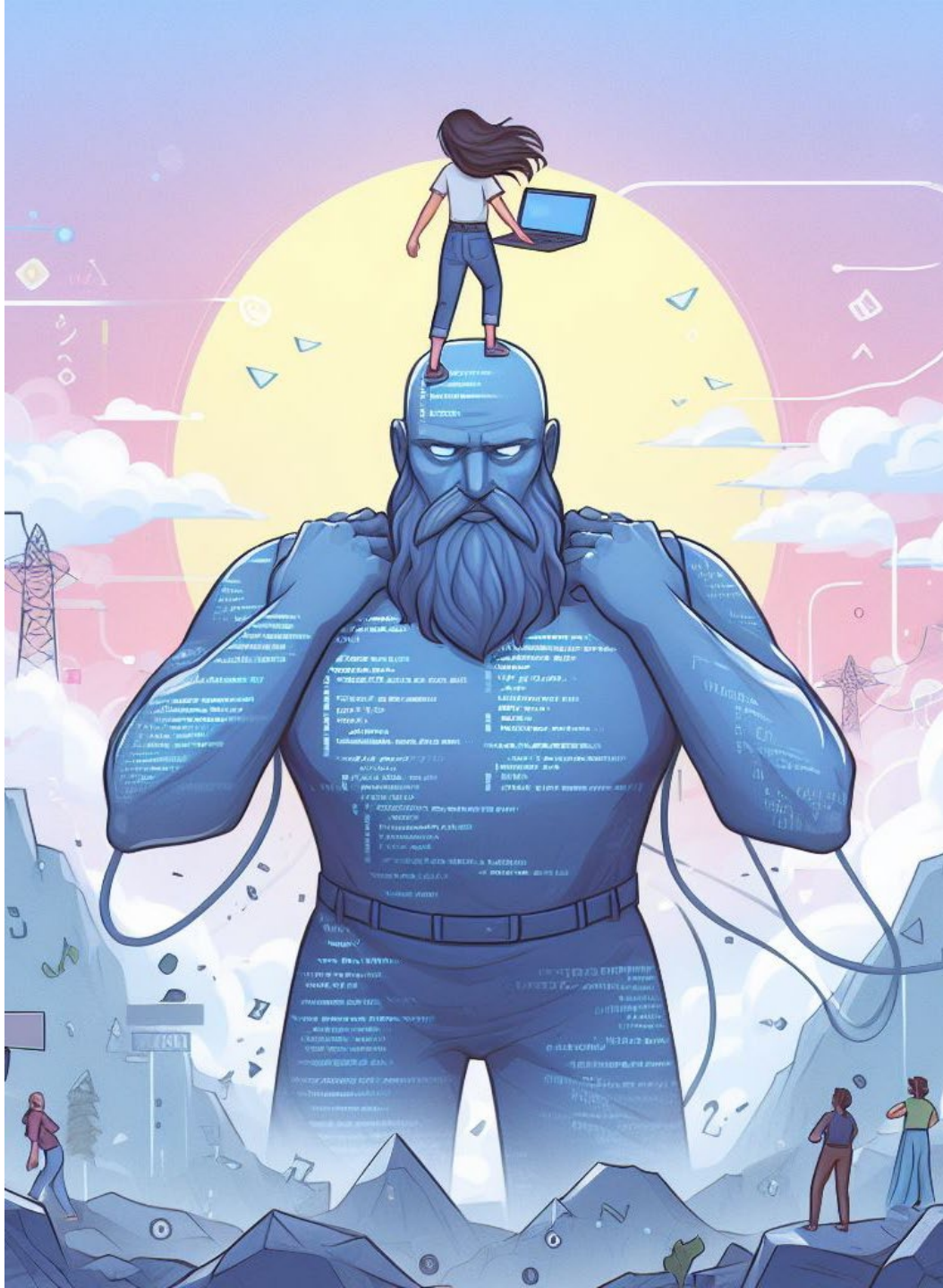
- It can be helpful to create a picture and work through some simple cases manually.
- Write pseudocode
- Get something working and then improve efficiency
- Example, subsequence detection.
  - Example 1
    - Input: s = "aqc", t = "ahqgdc“
    - Output: true
  - Example 2
    - Input: s = "azc", t = "ahbgdc“
    - Output: false




```javascript
/**
 * string s - sequence
 * string t - target
 * return boolean
 *
 * A subsequence of a string is a new string that is formed
 * from the original string by deleting some (can be none)
 * of the characters without disturbing the relative positions
 * of the remaining characters.
 * (i.e., "ace" is a subsequence of "abcde" while "aec" is not).
 *
 */
function isSubSequence(s, t) {
    prev = 0;
    for (let i = 0 ; i < s.length ; i++) {  // walk through the sequence
        let seen = false;
        // walk through the target
        for (let j = prev ; (j < t.length) && ! seen ; j++) {
            if (s[i] === t[j]) {
                prev = j + 1;
                seen = true;
            }
        }
        if (! seen) {
            return false;
        }
    }
    return true;
};
```

Standing On the Shoulders of Giants

- Isaac Newton: *"If I have seen further than others, it is by **standing upon the shoulders of giants**."*
- A massive amount of research has been done on algorithms over the last 60 years.
  - It's likely that there is already a good algorithm for the problem you need to solve.
  - Use built in algorithms (except for this week's problem set ☺)
  - If you need to build an algorithm, look at research and references first.



Further Reading and Practice

- Lots of good algorithm books and on-line resources
  - "The Algorithm Design Manual" by Steven S. Skiena.
    - Great reference with lots of examples
  - "Introduction to Algorithms" by Cormen, Leiserson, Rivest and Stein. (several editions)
    - Very popular University textbook
  - "Algorithms" by Robert Sedgwick. (several editions and languages)
  - Possible to find used versions of older editions
- Practice
  - LeetCode
    - Often used for interviews
    - Large number of practice problems
  - Codewars
    - Great place to find practice problems

Q&A and Demo