

# Competitive Programming

Following a Process

# Solving Problems Systematically

- Every problem is different (at least a little bit)
- ... but, you can follow a process as you solve them.
- This will help you
  - To get started faster
  - To work better with a team
  - To know how far you are toward solving a problem
  - To switch between problems

# Your Process

- You will want to develop and refine your own process
- ... based on:
  - How you like to work
  - What operating system and development tools you like to use
  - What you believe is effective
  - What your teammates agree to do

# My Process

- Consider my process for solving a new problem:
  1. Read the problem
  2. Think about a solution and do thumbnail asymptotic analysis
  3. Create a directory for the problem
    - Copy in sample inputs and output
    - Copy in a skeleton and a Makefile
  4. Bring up the code in your editor / development tools
  5. Write code to read input and test it on the samples
  6. Write the rest of the solution
  7. Test on provided samples
  8. Think about easy, interesting test cases to create
    - Big test cases
    - Special cases
  9. Submit and hope

# My Process

- Consider my process for solving a new problem:

1. Read the problem
2. Think about a solution and do thumbnail asymptotic analysis
3. Create a directory for the problem
  - Copy in sample inputs and output
  - Copy in a skeleton and a Makefile
4. Bring up the code in your editor / development tools
5. Write code to read input and test it on the samples
6. Write the rest of the solution
7. Test on provided samples
8. Think about easy, interesting test cases to create
  - Big test cases
  - Special cases
9. Submit and hope

I have a script for this.

I don't really do these separately.

I have a code library to help with this.

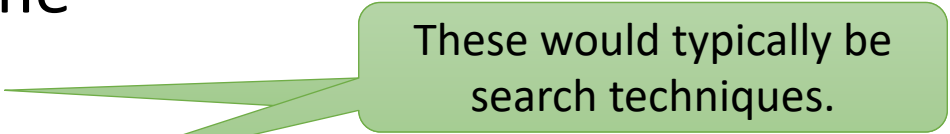
I have a script for this.

# Thumbnail Asymptotic Analysis

- Some problems require cleverness in the
  - Algorithm
  - Data structures
- Before you waste time coding, make sure your approach will be fast enough.
- Check the problem for the worst-case input size
  - Consider the running time of your approach,  $O(n \log n)$ ,  $O(n^2)$ ,  $O(n^2 \log n)$ , etc.
  - Generally, pretend 1 second of CPU time is about 1,000,000,000 operations

# Thumbnail Asymptotic Analysis

- Assume 1 second of CPU time
- For  $O(n!)$ , you need  $n \leq 12$
- For  $O(2^n)$ , you need  $n \leq 30$



These would typically be search techniques.

# Thumbnail Asymptotic Analysis

- Assume 1 second of CPU time
- For  $O(n!)$ , you need  $n \leq 12$
- For  $O(2^n)$ , you need  $n \leq 30$
- For  $O(n^3)$ , you need  $n \leq 1000$
- For  $O(n^2)$ , you need  $n \leq 30,000$
- $O(n \log n)$  is usually feasible for any reasonable input size

# Staging a new Problem

- Develop a naming scheme for your directories
- For Kattis problems, you can:
  - Use a subdirectory based on the problem ID
  - Start a directory once you think you can solve a particular problem
  - Have a directory for work-in-progress vs. solved problems
- For a contest:
  - Problems usually have a letter, a, b, c ...
  - Maybe stage each problem immediately, at the start of the contest
  - One team member can do this

# Filling in the Details

- To follow a problem-solving process, you will need a plan for:
  - What kinds of development tools you will use (e.g., editor vs IDE)?
  - How will you quickly switch problems?
  - What skeleton code you will use?
  - How will you try out the test inputs?
  - How will you check your output ?
- For a contest:
  - How will you divide up the stack of problems?
  - What will you do as you find the **easy** ones?