I) How to prepare

- try and solve problems on your own
- code on paper ?
- test your code on paper
- type your code as is to track mistakes

II) What do you need to know

- Core Data Structures, Algorithms and Concepts

DATA STRUCTURES	ALGORITHMS

DATA STRUCTURES
- linked lists
- trees, tries & graphs
- stacks and queues
- Heaps
- vectors / array lists
- hash tables

ALGORITHMS
- Breadth - First search
- Depth - First search
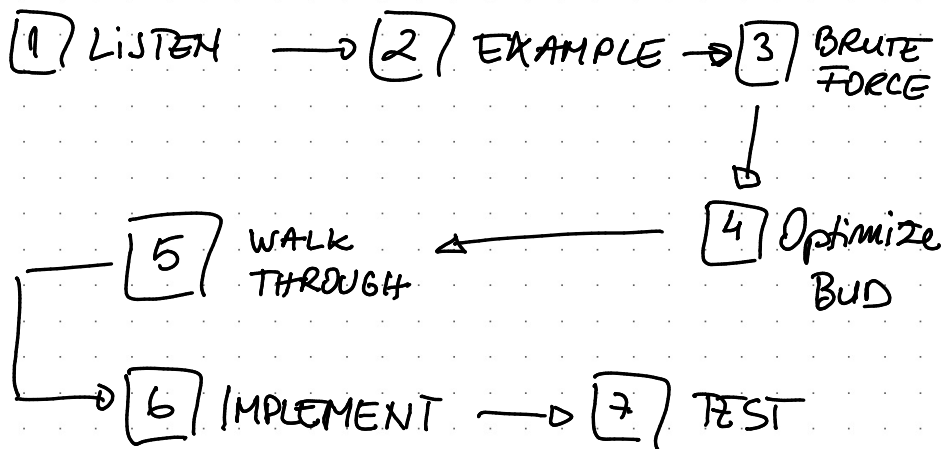- Binary search
- Merge sort
- Quick sort

CONCEPTS

- Bit Manipulation
- Memory (stack vs. Heap)
- Recursion
- Big O time & Space
- Dynamic Programming

* Hash tables are very important
* Powers of 2 Table
  III Walking Through a Problem

① LISTEN ⟶ ② EXAMPLE ⟶ ③ BRUTE FORCE
                                        ↓
⑤ WALK THROUGH ⟵⎯⎯⎯⎯⎯ ④ Optimize BUD

┌⟶ ⑥ IMPLEMENT ⟶ ⑦ TEST

BUD = ⟨ Bottlenecks
        Unnecessary Work
        Duplicated Work

* draw an example on the board
* look at unused information
* make time vs. space tradeoff
* precompute information
* hash tables
* write beautiful code (modularized)

<u>IV</u>) Optimize: Look for BUO

o Bottlenecks — slows down overall runtime

o Unnecessary work

$$a^3 + b^3 = c^3 + d^3 \qquad\qquad 1 < a, b, c, d < 1000$$

$n = 1000$

for a from 1 to n
  for b from 1 to n
    for c from 1 to n
      for d from 1 to n
        if $a^3 + b^3 = c^3 + d^3$
          print a, b, c, d
          $\boxed{break}$ out of the d loop

$O(N^4)$

---

for a
  for b
    for c
      $d = \sqrt[3]{a^3 + b^3 - c^3}$
      if (cond)
        print

$O(N^3)$

$n = 1000$

for c from 1 to n
    for d . . .
        result $= c^3 + d^3$
        append (c,d) to list (map [result])

for a . . .
    for b . . .
        result $= a^3 + b^3$
        list $-$ map. get (result)
        foreach pair in list
            print a, b, pair

$\triangle$ for each result, list in map
    for each pair 1 in list
        for each pair 2 in list
            print pair 1, pair 2

$O(N^2)$

$\underline{V)}$ Optimize: DIY

o small string s , larger string b
  find all permutations of s in b

eg: s: abbc

           b: cbabadcbbabbcbabaabccbabc

① walk through b, look at sliding windows
  of 4 characters / check if each window is
  a permutation of s

② walk through b, every time you see a character
  in s, check if the next four is a permutation
  of s

$\underline{VI \ \text{Optimize}: \text{Simplify and Generalize}}$

  - solve a simplified version first

$\underline{VII \ \text{Base Case and Build}}$

  . start with $n=1$

                $n=2$

                $n=3$

                ⋮

VIII) Optimize: Data Structure & Brainstorm

- map solution to a data structure

IX) Best Conceivable Runtime - BCR

- Brute Force
- Improved
- Optimal
- BCR

X) Handling Incorrect Answers
- you don't have to be perfect

XI) Reveal if you know the answer before hand

* The Offer & Beyond