CRACKING THE CODING SKILLS

Created By Gayle Laakmann McDowell

1 Listen

Pay very close attention to any info in the problem description. You probably need it all for an optimal algorithm.

2 Example

Most examples are too small or are special cases. **Debug your example.** Is there any way it's a special case? Is it big enough?

3 Brute Force

Get a brute-force solution as soon as possible. Don't worry about developing an efficient algorithm yet. State a naive algorithm and its runtime, then optimize from there. Don't code yet though!

BUD Optimization

Bottlenecks

Unnecessary Work

Duplicated Work

4 Optimize

Walk through your brute force with **BUD optimization** or try some of these ideas:

- Look for any unused info. You usually need all the information in a problem.
- Solve it manually on an example, then reverse engineer your thought process. How did you solve it?
- Solve it "incorrectly" and then think about why the algorithm fails. Can you fix those issues?
- Make a time vs. space tradeoff. Hash tables are especially useful!

5 Walk Through

Now that you have an optimal solution, **walk through your approach in detail**. Make sure you understand each detail before you start coding.

6 Implement

Your goal is to **write beautiful code**. Modularize your code from the beginning and refactor to clean up anything that isn't beautiful.

7 Test

Test in this order:

1. Conceptual test. Walk through your code like you would for a detailed code review.
2. Unusual or non-standard code.
3. Hot spots, like arithmetic and null nodes.
4. Small test cases. It's much faster than a big test case and just as effective.
5. Special cases and edge cases.

And when you find bugs, **fix them carefully!**

Best Conceivable Runtime (BCR)

BCR is the runtime you *know* you can't beat. For example, if asked to compute the intersection of two sets, you know you can't beat O(|A|+|B|).

5 Approaches

- **BUD:** Look for bottlenecks, unnecessary work, duplicated work.
- **DIY:** Do It Yourself
- **Simplify & Generalize:** Solve a simpler version.
- **Base Case & Build:** Solve for the base cases then build from there.
- **Data Structure Brainstorm:** Try various data structures.

What You Need To Know

1. **Data Structures:** Hash Tables, Linked Lists, Stacks, Queues, Trees, Tries, Graphs, Vectors, Heaps.
2. **Algorithms:** Quick Sort, Merge Sort, Binary Search, Breadth-First Search, Depth-First Search.
3. **Concepts:** Big-O Time, Big-O Space, Recursion & Memoization, Probability, Bit Manipulation.

Exercises:

- Implement data structures & algorithms from scratch.
- Prove to yourself the runtime of the major algorithms.

Books by Gayle

Do not...

- **Do not** ignore information given. Info is there for a reason.
- **Do not** try to solve problems in your head. Use an example!
- **Do not** push through code when confused. Stop and think!
- **Do not** dive into code without interviewer "sign off."