


















Tackling Hard Problems

What makes hard problems hard?

#	Title	Solution	Acceptance	Difficulty	Frequency ?
146	LRU Cache		25.1%	Hard	<div><div></div></div>
4	Median of Two Sorted Arrays		26.2%	Hard	<div><div></div></div>
42	Trapping Rain Water		42.9%	Hard	<div><div></div></div>
23	Merge k Sorted Lists		34.2%	Hard	<div><div></div></div>
273	Integer to English Words		24.2%	Hard	<div><div></div></div>
301	Remove Invalid Parentheses		39.0%	Hard	<div><div></div></div>
297	Serialize and Deserialize Binary Tree		40.5%	Hard	<div><div></div></div>
269	Alien Dictionary 		30.8%	Hard	<div><div></div></div>
76	Minimum Window Substring		30.6%	Hard	<div><div></div></div>
295	Find Median from Data Stream		36.2%	Hard	<div><div></div></div>
975	Odd Even Jump		49.2%	Hard	<div><div></div></div>
924	Minimize Malware Spread		39.5%	Hard	<div><div></div></div>
10	Regular Expression Matching		25.2%	Hard	<div><div></div></div>
224	Basic Calculator		32.4%	Hard	<div><div></div></div>
642	Design Search Autocomplete System 		37.3%	Hard	<div><div></div></div>
41	First Missing Positive		28.7%	Hard	<div><div></div></div>
85	Maximal Rectangle		33.1%	Hard	<div><div></div></div>

What makes hard problems hard?

- Edge cases
- Non-obvious optimizations
- Lots of moving parts

Edge Cases

- Not catching weird or non-obvious edge cases
- Less important in actual interviews because no automated testing
- Make sure that you really understand the problem (Step 1)
- Brainstorm edge cases at the beginning and define assumptions

Non-obvious optimizations

- Sometimes the best optimizations don't follow from your existing solution
- Take a step back and look for alternate approaches
- Draw on your existing knowledge

Lots of moving parts

- This is the most common and most easily addressable issue
- The problem simply becomes overwhelming
- Strategies:
 - Break it down into smaller problems
 - Solve a simpler version of the problem
 - Build solution from given info

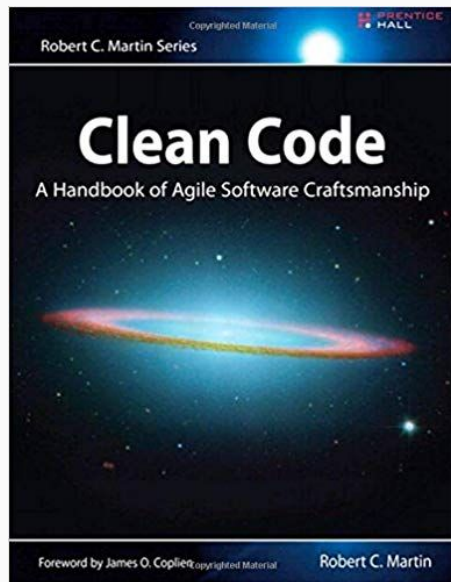
Breaking Down Problems



bytebyte

Break it into smaller problems

- Can I solve this problem by solving multiple other problems?
- Can the work I'm doing be split into a series of logical steps?
- Can I modularize my code (a la *Clean Code*)
- Examples:
 - 0-1 Knapsack
 - Longest palindromic substring
 - Almost anything



Is there a function...

Ask yourself:

"Is there a function that if I had it implemented, it would make this problem easy?"

Word Squares

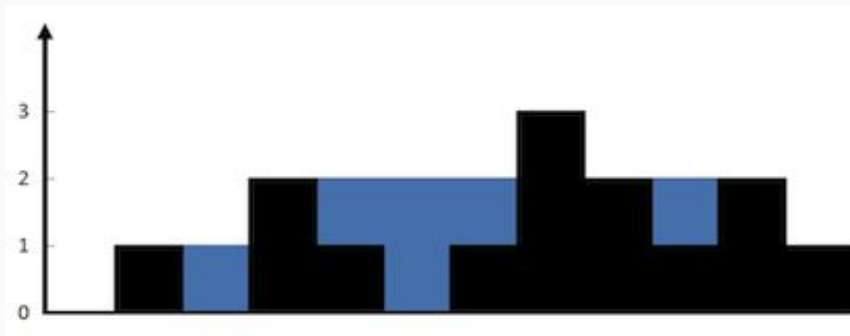
- Given a list of input strings, find all the valid word squares
- Write a function to find all potential word squares
- Write a function to validate a word square
- Write a controller that runs one then the other
- Alternatively, build and test for contradiction

[“ember”, “resin”, “heart”, “trend”,
“tests”, “rents”, “abuse”]

H	E	A	R	T
E	M	B	E	R
A	B	U	S	E
R	E	S	I	N
T	R	E	N	D

Trapping Rain Water ([Leetcode](#))

- Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.
- How can we simplify this problem?
- What if we just compute whether a given cell contains water?
- What if we compute the volume of 1 column?



Integer to English Words ([Leetcode](#))

- Given an integer, convert into plain English
- Write a function to compute highest-order string (eg. "One Hundred" or "Twelve Thousand")
- Then update the input and call again on the remaining string
- Could even break into further functions depending on input size
 - I.e. Special function for $i < 20$

`f(123) = "One Hundred Twenty Three"`

`f(12345) = "Twelve Thousand Three
Hundred Forty Five"`

`f(1234567) = "One Million Two Hundred
Thirty Four Thousand Five Hundred Sixty
Seven"`

`f(123456789) = "One Billion Two Hundred
Thirty Four Million Five Hundred Sixty
Seven Thousand Eight Hundred Ninety
One"`

Solve Simpler Problem

Is there a similar variant that is easier?

- Common in interviews to see a standard problem with some twist
- Can we start with the solution to the basic problem and then solve for the twist?
- Is there a more restrictive version of the problem that is easier to solve?
- Often similar to breaking into smaller problems

Some ways to simplify the problem

- Assume that the input is small
- Assume that the input is sorted
- Assume that you've handled all the edge cases
- Assume you're only solving for some specific use case

Remove Invalid Parentheses

- Given a string, remove the minimum number of invalid parentheses
eg. `removeParens("(a(b(c))")` => `"ab(c)"` or `"a(bc)"` or `"(abc)"`
- We could solve this by finding all valid ways to parenthesize the string (one of our standard recursive patterns) and then comparing these to the original string

Combination Problems

- Whenever we're looking for a specific combination
- Easiest to just compute all combinations and filter those combinations

Build Up Solution



bytebyte

Build from the info given

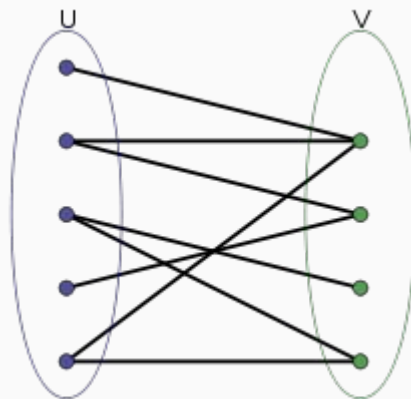
- If you're not sure where to go, start with the information you have
- Can be helpful for optimization or brute force solutions
- Use any info given as well as any info that you already know
- Can we take a greedy approach?

Can we use a greedy approach?

- Greedy solutions just take the best possible move at each step
- Simple example:
 - Making change
 - Just take the biggest possible coin each time
- Is X possible?
 - Assume it's impossible
 - Try to construct a solution and if you can create a valid solution its possible
 - Proof by contradiction

Is Graph Bipartite

- What does bipartite mean?
- Can there be multiple unique ways to divide the nodes?
 - No - causes contradiction
- Let's just try to split nodes
 - How do we know if it's valid?
 - Only even cycles
 - Node in both sets
- Proof by contradiction



Merge K Arrays

- We know all the arrays are sorted
- That means the minimum value has to be the first element of one of the arrays
- Once we select the first element, the next smallest element has to be either in the remaining set of first elements or the second in the array we picked the first one from
- Build solution from here

Median of Sorted Arrays

- Brute force: Merge or count from both ends

[1, 2, 3, 5, 6, 7, 8]
[2, 2, 2, 2, 5, 6, 7]

- Let's say the arrays are the same length

[1, 2, 3, 5]
[2, 5, 6, 7]

- If we split in half, we can eliminate half from both sides

[3, 5]
[2, 5]

- [Original version explanation](#)

[3]
[5]