#651. 17.21 You can calculate the area between the tallest bar overall and the tallest bar on the left by just iterating through the histogram and subtracting out any bars in between. You can do the same thing with the right side. How do you handle the remainder of the graph?

#652. 17.18 One brute force solution is to take each starting position and move forward until you've found a subsequence which contains all the target characters.

#653. 16.18 Don't forget to handle the possibility that the first character in the `pattern` is `b`.

#654. 16.20 In the real world, we should know that some prefixes/substrings won't work. For example, consider the number 33835676368. Although 3383 does correspond to `fftf`, there are no words that start with `fftf`. Is there a way we can short-circuit in cases like this?

#655. 17.7 An alternative approach is to think of this as a graph. How would this work?

#656. 17.13 You can think about the choices the recursive algorithm makes in one of two ways: (1) At each character, should I put a space here? (2) Where should I put the next space? You can solve both of these recursively.

#657. 17.8 If you needed only one side of the pair to increase, then you would just sort all the values on that side. Your longest sequence would in fact be all of the pairs (other than any duplicates, since the longest sequence needs to strictly increase). What does this tell you about the original problem?

#658. 17.21 You can handle the remainder of the graph by just repeating this process: find the tallest bar and the second tallest bar, and subtract out the bars in between.

#659. 17.4 To find the least significant bit of the missing number, note that you know how many `0`s and `1`s to expect. For example, if you see three `0`s and three `1`s in the least significant bit, then the missing number's least significant bit must be a `1`. Think about it: in any sequence of `0`s and `1`s, you'd get a `0`, then a `1`, then a `0`, then a `1`, and so on.

#660. 17.9 Rather than checking all values in the list for the next value (by multiplying each by 3, 5, and 7), think about it this way: when you insert a value `x` into the list, you can "create" the values `3x`, `5x`, and `7x` to be used later.

#661. 17.14 Think about the previous hint some more, particularly in the context of quicksort.

#662. 17.21 How can you make the process of finding the next tallest bar on each side faster?

#663. 16.18 Be careful with how you analyze the runtime. If you iterate through $O(n^2)$ substrings and each one does an $O(n)$ string comparison, then the total runtime is $O(n^3)$.

#664. 17.1 Now focus on the carrying. In what cases will values carry? How do you apply the carry to the number?

#665. 16.26 Consider thinking about it as, when you get to a multiplication or division sign, jumping to a separate "process" to compute the result of this chunk.

#666. 17.8 If you sort the values based on height, then this will tell you the ordering of the final pairs. The longest sequence must be in this relative order (but not necessarily containing all of the pairs). You now just need to find the longest increasing subsequence on weight while keeping the items in the same relative order. This is essentially the same problem as having an array of integers and trying to find the longest sequence you can build (without reordering those items).

#667. 16.16 Consider the three subarrays: `LEFT`, `MIDDLE`, `RIGHT`. Focus on just this question: Can you sort middle such that the entire array becomes sorted? How would you check this?

#668. 16.23 Looking at this table again, note that the number of rows will be 5^k, where k is the max number of calls to `rand5()`. In order to make each value between 0 and 6 have equal probability, $1/7$ th of the rows must map to 0, $1/7$ th to 1, and so on. Is this possible?

#669. 17.18 Another way of thinking about the brute force is that we take each starting index and *find the next instance of each element in the target string*. The maximum of all these next instances marks the end of a subsequence which contains all the target characters. What is the runtime of this? How can we make it faster?

#670. 16.6 Think about how you would merge two sorted arrays.

#671. 17.5 When the above tables have equal values for the number of As and Bs, the entire subarray (starting from index 0) has an equal number of As and Bs. How could you use this table to find qualifying subarrays that don't start at index 0?

#672. 17.19 Part 2: Adding the numbers together will tell us the result of `a + b`. Multiplying the numbers together will tell us the result of `a * b`. How can we get the exact values for a and b?

#673. 16.24 If we sorted the array, we could do repeated binary searches for the complement of a number. What if, instead, the array is given to us sorted? Could we then solve the problem in `O(N)` time and `O(1)` space?

#674. 16.19 If you were given the row and column of a water cell, how can you find all connected spaces?

#675. 17.7 We can treat adding X, Y as synonyms as adding an edge between the X node and the Y node. How then do we figure out the groups of synonyms?

#676. 17.21 Can you do precomputation to compute the next tallest bar on each side?

#677. 17.13 Will the recursive algorithm hit the same subproblems repeatedly? Can you optimize with a hash table?

#678. 17.14 What if, when you picked an element, you swapped elements around (as you do in quicksort) so that the elements below it would be located before the elements above it? If you did this repeatedly, could you find the smallest one million numbers?

#679. 16.6 Imagine you had the two arrays sorted and you were walking through them. If the pointer in the first array points to 3 and the pointer in the second array points to 9, what effect will moving the second pointer have on the difference of the pair?

#680. 17.12 To handle whether your recursive algorithm should return the start or the end of the linked list, you could try to pass a parameter down that acts as a flag. This won't work very well, though. The problem is that when you call `convert(current.left)`, you want to get the end of `left`'s linked list. This way you can join the end of the linked list to `current`. But, if `current` is someone else's right subtree, `convert(current)` needs to pass back the start of the linked list (which is actually the start of `current.left`'s linked list). Really, you need both the start and end of the linked list.

#681. 17.18 Consider the previously explained brute force solution. A bottleneck is repeatedly asking for the next instance of a particular character. Is there a way you can optimize this? You should be able to do this in `O(1)` time.

#682. 17.8 Try a recursive approach that just evaluates all possibilities.

#683. 17.4 Once you've identified that the least significant bit is a 0 (or a 1), you can rule out all the numbers without 0 as the least significant bit. How is this problem different from the earlier part?

#684. 17.23 Start with a brute force solution. Can you try the biggest possible square first?

#685. 16.18 Suppose you decide on a specific value for the "a" part of a pattern. How many possibilities are there for b?

#686. 17.9 When you add x to the list of the first k values, you can add 3x, 5x, and 7x to some new list. How do you make this as optimal as possible? Would it make sense to keep multiple queues of values? Do you always need to insert 3x, 5x, and 7x? Or, perhaps sometimes you need to insert only 7x? You want to avoid seeing the same number twice.

#687. 16.19 Try recursion to count the number of water cells.

#688. 16.8 Consider dividing up a number into sequences of three digits.

#689. 17.19 Part 2: We could do both. If we know that `a + b = 87` and `a * b = 962`, then we can solve for a and b: `a = 13` and `b = 74`. But this will also result in having to multiply really large numbers. The product of all the numbers could be larger than 10^{157}. Is there a simpler calculation you can make?

#690. 16.11 Consider building a diving board. What are the choices you make?

#691. 17.18 Can you precompute the next instance of a particular character from each index? Try using a multi-dimensional array.

#692. 17.1 The carry will happen when you are doing 1 + 1. How do you apply the carry to the number?

#693. 17.21 As an alternative solution, think about it from the perspective of each bar. Each bar will have water on top of it. How much water will be on top of each bar?

#694. 16.25 Both a hash table and a doubly linked list would be useful. Can you combine the two?

#695. 17.23 The biggest possible square is NxN. So if you try that square first and it works, then you know that you've found the best square. Otherwise, you can try the next smallest square.

#696. 17.19 Part 2: Almost any "equation" we can come up with will work here (as long as it's not equivalent to a linear sum). It's just a matter of keeping this sum small.

#697. 16.23 It is not possible to divide 5^k evenly by 7. Does this mean that you can't implement `rand7()` with `rand5()`?

#698. 16.26 You can also maintain two stacks, one for the operators and one for the numbers. You push a number onto the stack every time you see it. What about the operators? When do you pop operators from the stack and apply them to the numbers?

#699. 17.8 Another way to think about the problem is this: if you had the longest sequence ending at each element `A[0]` through `A[n-1]`, could you use that to find the longest sequence ending at element `A[n-1]`?

#700. 16.11 Consider a recursive solution.

#701. 17.12 Many people get stuck at this point and aren't sure what to do. Sometimes they need the start of the linked list, and sometimes they need the end. A given node doesn't necessarily know what to return on its `convert` call. Sometimes the simple solution is easiest: always return both. What are some ways you could do this?

#702. 17.19 Part 2: Try a sum of squares of the values.

#703. 16.20 A trie might help us short-circuit. What if you stored the whole list of words in the trie?

#704. 17.7 Each connected subgraph represents a group of synonyms. To find each group, we can do repeated breadth-first (or depth-first) searches.

#705. 17.23 Describe the runtime of the brute force solution.

#706. 16.19 How can you make sure that you're not revisiting the same cells? Think about how breadth-first search or depth-first search on a graph works.

#707. 16.7 When `a > b`, then `a - b > 0`. Can you get the sign bit of `a - b`?

#708. 16.16 In order to be able to sort `MIDDLE` and have the whole array become sorted, you need `MAX(LEFT) <= MIN(MIDDLE and RIGHT)` and `MAX(LEFT and MIDDLE) <= MIN(RIGHT)`.

#709. 17.20 What if you used a heap? Or two heaps?

#710. 16.4 If you were calling `hasWon` multiple times, how might your solution change?

#711. 16.5 Each zero in `n!` corresponds to `n` being divisible by a factor of 10. What does that mean?

#712. 17.1 You can use an AND operation to compute the carry. What do you do with it?

#713. 17.5 Suppose, in this table, index `i` has `count(A, 0->i) = 3` and `count(B, 0->i) = 7`. This means that there are four more Bs than As. If you find a later spot `j` with the same difference (`count(B, 0->j) - count(A, 0->j)`), then this indicates a subarray with an equal number of As and Bs.

#714. 17.23 Can you do preprocessing to optimize this solution?

#715. 16.11 Once you have a recursive algorithm, think about the runtime. Can you make this faster? How?

#716. 16.1 Let `diff` be the difference between `a` and `b`. Can you use `diff` in some way? Then can you get rid of this temporary variable?

#717. 17.19 Part 2: You might need the quadratic formula. It's not a big deal if you don't remember it. Most people won't. Remember that there is such a thing as good enough.

#718. 16.18 Since the value of `a` determines the value of `b` (and vice versa) and either `a` or `b` must start at the beginning of the value, you should have only `O(n)` possibilities for how to split up the pattern.

#719. 17.12 You could return both the start and end of a linked list in multiple ways. You could return a two-element array. You could define a new data structure to hold the start and end. You could re-use the `BiNode` data structure. If you're working in a language that supports this (like Python), you could just return multiple values. You could solve the problem as a circular linked list, with the start's previous pointer pointing to the end (and then break the circular list in a wrapper method). Explore these solutions. Which one do you like most and why?

#720. 16.23 You can implement `rand7()` with `rand5()`, you just can't do it deterministically (such that you know it will definitely terminate after a certain number of calls). Given this, write a solution that works.

#721. 17.23 You should be able to do this in $O(N^3)$ time, where N is the length of one dimension of the square.

#722. 16.11 Consider memoization to optimize the runtime. Think carefully about what exactly you cache. What is the runtime? The runtime is closely related to the max size of the table.

#723. 16.19 You should have an algorithm that's $O(N^2)$ on an NxN matrix. If your algorithm isn't, consider if you've miscomputed the runtime or if your algorithm is suboptimal.

#724. 17.1 You might need to do the add/carry operation more than once. Adding `carry` to `sum` might cause new values to carry.

#725. 17.18 Once you have the precomputation solution figured out, think about how you can reduce the space complexity. You should be able to get it down to $O(SB)$ time and $O(B)$ space (where B is the size of the larger array and S is the size of the smaller array).

#726. 16.20 We're probably going to run this algorithm many times. If we did more preprocessing, is there a way we could optimize this?

#727. 16.18 You should be able to have an $O(n^2)$ algorithm.

#728. 16.7 Have you considered how to handle integer overflow in `a - b`?

#729. 16.5 Each factor of 10 in `n!` means `n!` is divisible by 5 and 2.

#730. 16.15 For ease and clarity in implementation, you might want to use other methods and classes.

#731. 17.18 Another way to think about it is this: Imagine you had a list of the indices where each item appeared. Could you find the first possible subsequence with all the elements? Could you find the second?

#732. 16.4 If you were designing this for an NxN board, how might your solution change?

#733. 16.5 Can you count the number of factors of 5 and 2? Do you need to count both?

#734. 17.21 Each bar will have water on top of it that matches the minimum of the tallest bar on the left and the tallest bar on the right. That is, `water_on_top[i] = min(tallest_bar(0->i), tallest_bar(i, n))`.

#735. 16.16 Can you expand the middle until the earlier condition is met?

#736. 17.23 When you're checking to see if a particular square is valid (all black borders), you check how many black pixels are above (or below) a coordinate and to the left (or right) of this coordinate. Can you precompute the number of black pixels above and to the left of a given cell?

#737. 16.1 You could also try using XOR.

#738. 17.22 What if you did a breadth-first search starting from both the source word and the destination word?

#739. 17.13 In real life, we would know that some paths will not lead to a word. For example, there are no words that start with `hellothisism`. Can we terminate early when going down a path that we know won't work?

#740. 16.11 There's an alternate, clever (and very fast) solution. You can actually do this in linear time without recursion. How?

#741. 17.18 Consider using a heap.

#742. 17.21 You should be able to solve this in `O(N)` time and `O(N)` space.

#743. 17.17 Alternatively, you could insert each of the smaller strings into the trie. How would this help you solve the problem? What is the runtime?

#744. 16.20 With preprocessing, we can actually get the lookup time down to `O(1)`.

#745. 16.5 Have you considered that 25 actually accounts for two factors of 5?

#746. 16.16 You should be able to solve this in `O(N)` time.

#747. 16.11 Think about it this way. You are picking K planks and there are two different types. All choices with 10 of the first type and 4 of the second type will have the same sum. Can you just iterate through all possible choices?

#748. 17.25 Can you use a trie to terminate early when a rectangle looks invalid?

#749. 17.13 For early termination, try a trie.

XIV

About the Author

Gayle Laakmann McDowell has a strong background in software development with extensive experience on both sides of the hiring table.

She has worked for Microsoft, Apple, and Google as a software engineer. She spent three years at Google, where she was one of the top interviewers and served on the hiring committee. She interviewed hundreds of candidates in the U.S. and abroad, assessed thousands of candidate interview packets for the hiring committee, and reviewed many more resumes.

As a candidate, she interviewed with—and received offers from—twelve tech companies, including Microsoft, Google, Amazon, IBM, and Apple.

Gayle founded CareerCup to enable candidates to perform at their best during these challenging interviews. CareerCup.com offers a database of thousands of interview questions from major companies and a forum for interview advice.

In addition to ***Cracking the Coding Interview***, Gayle has written other two books:

- ***Cracking the Tech Career: Insider Advice on Landing a Job at Google, Microsoft, Apple, or Any Top Tech Company*** provides a broader look at the interview process for major tech companies. It offers insight into how anyone, from college freshmen to marketing professionals, can position themselves for a career at one of these companies.
- ***Cracking the PM Interview: How to Land a Product Manager Job in Technology*** focuses on product management roles at startups and big tech companies. It offers strategies to break into these roles and teaches job seekers how to prepare for PM interviews.

Through her role with CareerCup, she consults with tech companies on their hiring process, leads technical interview training workshops, and coaches engineers at startups for acquisition interviews.

She holds bachelor's degree and master's degrees in computer science from the University of Pennsylvania and an MBA from the Wharton School.

She lives in Palo Alto, California, with her husband, two sons, dog, and computer science books. She still codes daily.

Amazon.com's #1 Best-Selling Interview Book

CRACKING *the* CODING INTERVIEW

I am not a recruiter. I am a software engineer. And as such, I know what it's like to be asked to whip up brilliant algorithms on the spot and then write flawless code on a whiteboard. I've been through this—as a candidate and as an interviewer.

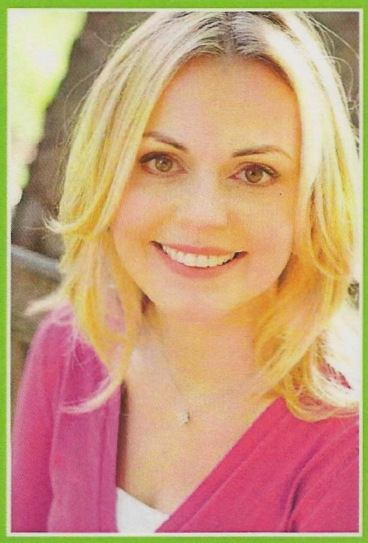
Cracking the Coding Interview, 6th Edition is here to help you through this process, teaching you what you need to know and enabling you to perform at your very best. I've coached and interviewed hundreds of software engineers. The result is this book.

Learn how to uncover the hints and hidden details in a question, discover how to break down a problem into manageable chunks, develop techniques to unstick yourself when stuck, learn (or re-learn) core computer science concepts, and practice on 189 interview questions and solutions.

These interview questions are real; they are not pulled out of computer science textbooks. They reflect what's truly being asked at the top companies, so that you can be as prepared as possible.

WHAT'S INSIDE?

- 189 programming interview questions, ranging from the basics to the trickiest algorithm problems.
- A walk-through of how to derive each solution, so that you can learn how to get there yourself.
- Hints on how to solve each of the 189 questions, just like what you would get in a real interview.
- Five proven strategies to tackle algorithm questions, so that you can solve questions you haven't seen.
- Extensive coverage of essential topics, such as big O time, data structures, and core algorithms.
- A "behind the scenes" look at how top companies, like Google and Facebook, hire developers.
- Techniques to prepare for and ace the "soft" side of the interview: behavioral questions.
- For interviewers and companies: details on what makes a good interview question and hiring process.



GAYLE LAAKMANN MCDOWELL

Gayle Laakmann McDowell is the founder and CEO of CareerCup and the author of Cracking the PM Interview and Cracking the Tech Career.

Gayle has a strong background in software development, having worked as a software engineer at Google, Microsoft, and Apple. At Google, she interviewed hundreds of software engineers and evaluated thousands of hiring packets as part of the hiring committee. She holds a B.S.E. and M.S.E. in computer science from the University of Pennsylvania and an MBA from the Wharton School.

She now consults with tech companies to improve their hiring process and with startups to prepare them for acquisition interviews.

ISBN 9780984782857

90000 >

9 780984 782857

6TH EDITION