#295. 8.9 We can ensure that this string is valid by counting the number of left and right parens. It is always valid to add a left paren, up until the total number of pairs of parens. We can add a right paren as long as `count(left parens) <= count(right parens)`.

#296. 6.4 You can think about this either as the probability(3 ants walking clockwise) + probability(3 ants walking counter-clockwise). Or, you can think about it as: The first ant picks a direction. What's the probability of the other ants picking the same direction?

#297. 5.2 Think about what happens for values that can't be represented accurately in binary.

#298. 10.3 Can you modify binary search for this purpose?

#299. 11.1 What will happen to the `unsigned int`?

#300. 8.11 Try breaking it down into subproblems. If you were making change, what is the first choice you would make?

#301. 10.10 The problem with using an array is that it will be slow to insert a number. What other data structures could we use?

#302. 5.5 If `(n & (n-1)) == 0`, then this means that n and `n - 1` never have a 1 in the same spot. Why would that happen?

#303. 10.9 Another way to think about this is that if you drew a rectangle around a cell extending to the bottom, right coordinate of the matrix, the cell would be bigger than all the items in this square.

#304. 9.2 Is there any way to search from both the source and destination? For what reason or in what case might this be faster?

#305. 8.14 If your code looks really lengthy, with a lot of if's (for each possible operator, "target" boolean result, and left/right side), think about the relationship between the different parts. Try to simplify your code. It should not need a ton of complicated if-statements. For example, consider expressions of the form <LEFT>OR<RIGHT> versus <LEFT>AND<RIGHT>. Both may need to know the number of ways that the <LEFT> evaluates to true. See what code you can reuse.

#306. 6.9 The number 3 has an even number of factors (1 and 3). The number 12 has an even number of factors (1, 2, 3, 4, 6, 12). What numbers do not? What does this tell you about the doors?

#307. 7.12 Think carefully about what information the linked list node needs to contain.

#308. 8.12 We know that each row must have a queen. Can you try all possibilities?

#309. 8.7 Approach 2: To generate a permutation of abcd, you need to pick an initial character. It can be a, b, c, or d. You can then permute the remaining characters. How can you use this approach to generate all permutations of the full string?

#310. 10.3 What is the runtime of your algorithm? What will happen if the array has duplicates?

#311. 9.5 How would you scale this to a larger system?

#312. 5.4 Get Next: Can you flip a 0 to a 1 to create the next biggest number?

#313. 11.4 Think about what load testing is designed to test. What are the factors in the load of a webpage? What criteria would be used to judge if a webpage performs satisfactorily under heavy load?

#314. 5.3 Each sequence can be lengthened by merging it with an adjacent sequence (if any) or just flipping the immediate neighboring zero. You just need to find the best choice.

#315. 10.8 Consider implementing your own bit vector class. It's a good exercise and an important part of this problem.

#316. 10.11 You should be able to design an O(n) algorithm.

#317. 10.9 A cell will be larger than all the items below it and to the right. It will be smaller than all cells above it and to the left. If we wanted to eliminate the most elements first, which element should we compare the value x to?

#318. 8.6 If you're having trouble with recursion, then try trusting the recursive process more. Once you've figured out how to move the top two disks from tower 0 to tower 2, trust that you have this working. When you need to move three disks, trust that you can move two disks from one tower to another. Now, two disks have been moved. What do you do about the third?

#319. 6.1 Imagine there were just three bottles and one had heavier pills. Suppose you put different numbers of pills from each bottle on the scale (for example, bottle 1 has 5 pills, bottle 2 has 2 pills, and bottle 3 has 9 pills). What would the scale show?

#320. 10.4 Think about how binary search works. What will be the issue with just implementing binary search?

#321. 9.2 Discuss how you might implement these algorithms and this system in the real world. What sort of optimizations might you make?

#322. 8.13 Once we pick the box on the bottom, we need to pick the second box. Then the third box.

#323. 6.2 The probability of making two out of three shots is probability(make shot 1, make shot 2, miss shot 3) + probability(make shot 1, miss shot 2, make shot 3) + probability(miss shot 1, make shot 2, make shot 3) + probability(make shot 1, make shot 2, make shot 3).

#324. 8.11 If you were making change, the first choice you might make is how many quarters you need to use.

#325. 11.2 Think about issues both within the program and outside of the program (the rest of the system).

#326. 9.4 Estimate how much space is needed for this.

#327. 8.14 Look at your recursion. Do you have repeated calls anywhere? Can you memoize it?

#328. 5.7 The value 1010 in binary is 10 in decimal or 0xA in hex. What will a sequence of 101010... be in hex? That is, how do you represent an alternating sequence of 1s and 0s with 1s in the odd places? How do you do this for the reverse (1s in the even spots)?

#329. 11.3 Consider both extreme cases and more general cases.

#330. 10.9 If we compare x to the center element in the matrix, we can eliminate roughly one quarter of the elements in the matrix.

#331. 8.2 For the robot to reach the last cell, it must find a path to the second-to-last cells. For it to find a path to the second-to-last cells, it must find a path to the third-to-last cells.

#332. 10.1 Try moving from the end of the array to the beginning.

#333. 6.8 If we drop Egg 1 at fixed intervals (e.g., every 10 floors), then the worst case is the worst case for Egg 1 + the worst case for Egg 2. The problem with our earlier solutions is that as Egg 1 does more work, Egg 2 doesn't do any less work. Ideally, we'd like to balance this a bit. As Egg 1 does more work (has survived more drops), Egg 2 should have less work to do. What might this mean?

#334. 9.3 Think about how infinite loops might occur.

#335. 8.7 Approach 2: To generate all permutations of abcd, pick each character (a, b, c, or d) as a starting character. Permute the remaining characters and prepend the starting character. How do you permute the remaining characters? With a recursive process that follows the same logic.

#336. 5.6 How would you figure out how many bits are different between two numbers?

#337. 10.4 Binary search requires comparing an element to the midpoint. Getting the midpoint requires knowing the length. We don't know the length. Can we find it?

#338. 8.4 Subsets that contain c will be subsets {a, b, c} but not {a, b}. Can you build these subsets from the subsets of {a, b}?

#339. 5.4 Get Next: Flipping a 0 to a 1 will create a bigger number. The farther right the index is the smaller the bigger number is. If we have a number like 1001, we want to flip the rightmost 0 (to create 1011). But if we have a number like 1010, we should not flip the rightmost 1.

#340. 8.3 Given a specific index and value, can you identify if the magic index would be before or after it?

#341. 6.6 Now suppose there were two blue-eyed people. What would they see? What would they know? When would they leave? Remember your answer from the prior hint. Assume they know the answer to the earlier hint.

#342. 10.2 Do you even need to truly "sort"? Or is just reorganizing the list sufficient?

#343. 8.11 Once you've decided to use two quarters to make change for 98 cents, you now need to figure out how many ways to make change for 48 cents using nickels, dimes, and pennies.

#344. 7.5 Think about all the different functionality a system to read books online would have to support. You don't have to do everything, but you should think about making your assumptions explicit.

#345. 11.4 Could you build your own? What might that look like?

#346. 5.5 What is the relationship between how n looks and how n - 1 looks? Walk through a binary subtraction.

#347. 9.4 Will you need multiple passes? Multiple machines?

#348. 10.4 We can find the length by using an exponential backoff. First check index 2, then 4, then 8, then 16, and so on. What will be the runtime of this algorithm?

#349. 11.6 What can we automate?

#350. 8.12 Each row must have a queen. Start with the last row. There are eight different columns on which you can put a queen. Can you try each of these?

#351. 7.10 Should number cells, blank cells, and bomb cells be separate classes?

#352. 5.3 Try to do it in linear time, a single pass, and `O(1)` space.

#353. 9.3 How would you detect the same page? What does this mean?

#354. 8.4 You can build the remaining subsets by adding `c` to all the subsets of `{a, b}`.

#355. 5.7 Try masks `0xaaaaaaaa` and `0x55555555` to select the even and odd bits. Then try shifting the even and odd bits around to create the right number.

#356. 8.7 Approach 2: You can implement this approach by having the recursive function pass back the list of the strings, and then you prepend the starting character to it. Or, you can push down a prefix to the recursive calls.

#357. 6.8 Try dropping Egg 1 at bigger intervals at the beginning and then at smaller and smaller intervals. The idea is to keep the sum of Egg 1 and Egg 2's drops as constant as possible. For each additional drop that Egg 1 takes, Egg 2 takes one fewer drop. What is the right interval?

#358. 5.4 Get Next: We should flip the rightmost non-trailing 0. The number 1010 would become 1110. Once we've done that, we need to flip a 1 to a 0 to make the number as small as possible, but bigger than the original number (1010). What do we do? How can we shrink the number?

#359. 8.1 Try memoization as a way to optimize an inefficient recursive program.

#360. 8.2 Simplify this problem a bit by first figuring out if there's a path. Then, modify your algorithm to track the path.

#361. 7.10 What is the algorithm to place the bombs around the board?

#362. 11.1 Look at the parameters for `printf`.

#363. 7.2 Before coding, make a list of the objects you need and walk through the common algorithms. Picture the code. Do you have everything you need?

#364. 8.10 Think about this as a graph.

#365. 9.3 How do you define if two pages are the same? Is it the URLs? Is it the content? Both of these can be flawed. Why?

#366. 5.8 First try the naive approach. Can you set a particular "pixel"?

#367. 6.3 Picture a domino laying down on the board. How many black squares does it cover? How many white squares?

#368. 8.13 Once you have a basic recursive algorithm implemented, think about if you can optimize it. Are there any repeated subproblems?

#369. 5.6 Think about what an XOR indicates. If you do a XOR b, where does the result have 1s? Where does it have 0s?

#370. 6.6 Build up from this. What if there were three blue-eyed people? What if there were four blue-eyed people?

#371. 8.12 Break this down into smaller subproblems. The queen at row 8 must be at column 1, 2, 3, 4, 5, 6, 7, or 8. Can you print all ways of placing eight queens where a queen is at row 8 and column 3? You then need to check all the ways of placing a queen on row 7.

#372. 5.5 When you do a binary subtraction, you flip the rightmost 0s to a 1, stopping when you get to a 1 (which is also flipped). Everything (all the 1s and 0s) on the left will stay put.

#373. 8.4 You can also do this by mapping each subset to a binary number. The `i`th bit could represent a "boolean" flag for whether an element is in the set.

#374. 6.8 Let X be the first drop of Egg 1. This means that Egg 2 would do `X - 1` drops if Egg 1 broke. We want to try to keep the sum of Egg 1 and Egg 2's drops as constant as possible. If Egg 1 breaks on the second drop, then we want Egg 2 to do `X - 2` drops. If Egg 1 breaks on the third drop, then we want Egg 2 to do `X - 3` drops. This keeps the sum of Egg 1 and Egg 2 fairly constant. What is X?

#375. 5.4 Get Next: We can shrink the number by moving all the 1s to the right of the flipped bit as far right as possible (removing a 1 in the process).

#376. 10.10 Would it work well to use a binary search tree?

#377. 7.10 To place the bombs randomly on the board: Think about the algorithm to shuffle a deck of cards. Can you apply a similar technique?

#378. 8.13 Alternatively, we can think about the repeated choices as: Does the first box go on the stack? Does the second box go on the stack? And so on.

#379. 6.5 If you fill the 5-quart jug and then use it to fill the 3-quart jug, you'll have two quarts left in the 5-quart jug. You can either keep those two quarts where they are, or you can dump the contents of the smaller jug and pour the two quarts in there.

#380. 8.11 Analyze your algorithm. Is there any repeated work? Can you optimize this?

#381. 5.8 When you're drawing a long line, you'll have entire bytes that will become a sequence of 1s. Can you set this all at once?

#382. 8.10 You can implement this using depth-first search (or breadth-first search). Each adjacent pixel of the "right" color is a connected edge.

#383. 5.5 Picture n and `n-1`. To subtract 1 from n, you flipped the rightmost 1 to a 0 and all the 0s on its right to 1s. If `n & n-1 == 0`, then there are no 1s to the left of the first 1. What does that mean about n?

#384. 5.8 What about the start and end of the line? Do you need to set those pixels individually, or can you set them all at once?

#385. 9.1 Think about this as a real-world application. What are the different factors you would need to consider?

#386. 7.10 How do you count the number of bombs neighboring a cell? Will you iterate through all cells?

#387. 6.1 You should be able to have an equation that tells you the heavy bottle based on the weight.

#388. 8.2 Think again about the efficiency of your algorithm. Can you optimize it?

#389. 7.9 The `rotate()` method should be able to run in `O(1)` time.

#390. 5.4 Get Previous: Once you've solved Get Next, try to invert the logic for Get Previous.

#391. 5.8 Does your code handle the case when `x1` and `x2` are in the same byte?

#392. 10.10 Consider a binary search tree where each node stores some additional data.

#393. 11.6 Have you thought about security and reliability?

#394. 8.11 Try using memoization.

#395. 6.8 I got 14 drops in the worst case. What did you get?

#396. 9.1 There's no one right answer here. Discuss several different technical implementations.

#397. 6.3 How many black squares are there on the board? How many white squares?

#398. 5.5 We know that n must have only one 1 if `n & (n-1) == 0`. What sorts of numbers have only one 1?

#399. 7.10 When you click on a blank cell, what is the algorithm to expand the neighboring cells?

#400. 6.5 Once you've developed a way to solve this problem, think about it more broadly. If you are given a jug of size X and another jug of size Y, can you always use it to measure Z?

#401. 11.3 Is it possible to test everything? How will you prioritize testing?

III

Hints for Knowledge-Based Questions

#402. 12.9 Focus on the concept firsts, then worry about the exact implementation. How should `SmartPointer` look?

#403. 15.2 A context switch is the time spent switching between two processes. This happens when you bring one process into execution and swap out the existing process.

#404. 13.1 Think about who can access private methods.

#405. 15.1 How do these differ in terms of memory?

#406. 12.11 Recall that a two dimensional array is essentially an array of arrays.

#407. 15.2 Ideally, we would like to record the timestamp when one process "stops" and the timestamp when another process "starts." But how do we know when this swapping will occur?

#408. 14.1 A GROUP BY clause might be useful.

#409. 13.2 When does a finally block get executed? Are there any cases where it won't get executed?

#410. 12.2 Can we do this in place?

#411. 14.2 It might be helpful to break the approach into two pieces. The first piece is to get each building ID and the number of open requests. Then, we can get the building names.

#412. 13.3 Consider that some of these might have different meanings depending on where they are applied.

#413. 12.10 Typically, `malloc` will just give us an arbitrary block of memory. If we can't override this behavior, can we work with it to do what we need?

#414. 15.7 First implement the single-threaded FizzBuzz problem.

#415. 15.2 Try setting up two processes and have them pass a small amount of data back and forth. This will encourage the system to stop one process and bring the other one in.

#416. 13.4 The purpose of these might be somewhat similar, but how does the implementation differ?

#417. 15.5 How can we ensure that `first()` has terminated before calling `second()`?

#418. 12.11 One approach is to call `malloc` for each array. How would we free the memory here?

#419. 15.3 A deadlock can happen when there's a "cycle" in the order of who is waiting for whom. How can we break or prevent this cycle?

#420. 13.5 Think about the underlying data structure.

#421. 12.7 Think about why we use virtual methods.

#422. 15.4 If every thread had to declare upfront what processes it might need, could we detect possible deadlocks in advance?

#423. 12.3 What is the underlying data structure behind each? What are the implications of this?

#424. 13.5 `HashMap` uses an array of linked lists. `TreeMap` uses a red-black tree. `LinkedHashMap` uses doubly-linked buckets. What is the implication of this?

#425. 13.4 Consider the usage of primitive types. How else might they differ in terms of how you can use the types?

#426. 12.11 Can we allocate this instead as a contiguous block of memory?

#427. 12.8 This data structure can be pictured as a binary tree, but it's not necessarily. What if there's a loop in the structure?

#428. 14.7 You probably need a list of students, their courses, and another table building a relationship between students and courses. Note that this is a many-to-many relationship.

#429. 15.6 The keyword `synchronized` ensures that two threads cannot execute synchronized methods on the same instance at the same time.

#430. 13.5 Consider how they might differ in terms of the order of iteration through the keys. Why might you want one option instead of the others?

#431. 14.3 First try to get a list of the IDs (just the IDs) of all the relevant apartments.

#432. 12.10 Imagine we have a sequential set of integers (3, 4, 5, ...). How big does this set need to be to ensure that one of the numbers is divisible by 16?

#433. 15.5 Why would using `boolean` flags to do this be a bad idea?

#434. 15.4 Think about the order of requests as a graph. What does a deadlock look like within this graph?

#435. 13.6 Object reflection allows you to get information about methods and fields in an object. Why might this be useful?

#436. 14.6 Be particularly careful about which relationships are one-to-one vs. one-to-many vs. many-to-many.

#437. 15.3 One idea is to just not let a philosopher hold onto a chopstick if he can't get the other one.

#438. 12.9 Think about tracking the number of references. What will this tell us?

#439. 15.7 Don't try to do anything fancy on the single-threaded problem. Just get something that is simple and easily readable.

#440. 12.10 How will we free the memory?

#441. 15.2 It's okay if your solution isn't totally perfect. That might not be possible. Discuss the tradeoffs of your approach.

#442. 14.7 Think carefully about how you handle ties when selecting the top 10%.

#443. 13.8 A naive approach is to pick a random subset size z and then iterate through the elements, putting it in the set with probability `z/list_size`. Why would this not work?

#444. 14.5 Denormalization means adding redundant data to a table. It's typically used in very large systems. Why might this be useful?

#445. 12.5 A shallow copy copies just the initial data structure. A deep copy does this, and also copies any underlying data. Given this, why might you use one versus the other?

#446. 15.5 Would semaphores be useful here?

#447. 15.7 Outline the structure for the threads without worrying about synchronizing anything.

#448. 13.7 Consider how you'd implement this first without lambda expressions.

#449. 12.1 If we already had the number of lines in the file, how would we do this?

#450. 13.8 Pick the list of all the subsets of an n-element set. For any given item x, half of the subsets contain x and half do not.

#451. 14.4 Describe INNER JOINs and OUTER JOINs. OUTER JOINs can have multiple types: left, right, and full.

#452. 12.2 Be careful about the null character.

#453. 12.9 What are all the different methods/operators we might want to override?

#454. 13.5 What would the runtime of the common operations be?

#455. 14.5 Think about the cost of joins on a large system.

#456. 12.6 The keyword `volatile` signals that a variable might be changed from outside of the program, such as by another process. Why might this be necessary?

#457. 13.8 Do not pick the length of the subset in advance. You don't need to. Instead, think about this as picking whether each element will be put into the set.

#458. 15.7 Once you get the structure of each thread done, think about what you need to synchronize.

#459. 12.1 Suppose we didn't have the number of lines in the file. Is there a way we could do this without first counting the number of lines?

#460. 12.7 What would happen if the destructor were not virtual?

#461. 13.7 Break this up into two parts: filtering the countries and then getting a sum.

#462. 12.8 Consider using a hash table.

#463. 12.4 You should discuss vtables here.

#464. 13.7 Can you do this without a `filter` operation?

IV

Hints for Additional Review Problems

#465. 16.3 Think about what you're going to design for.

#466. 16.12 Consider a recursive or tree-like approach.

#467. 17.1 Walk through binary addition by hand (slowly!) and try to really understand what is happening.

#468. 16.13 Draw a square and a bunch of lines that cut it in half. Where are those lines located?

#469. 17.24 Start with a brute force solution.

#470. 17.14 There are actually several approaches. Brainstorm these. It's okay to start off with a naive approach.

#471. 16.20 Consider recursion.

#472. 16.3 Will all lines intercept? What determines if two lines intercept?

#473. 16.7 Let k be 1 if a > b and 0 otherwise. If you were given k, could you return the max (without a comparison or if-else logic)?

#474. 16.22 The tricky bit is handling an infinite grid. What are your options?

#475. 17.15 Try simplifying this problem: What if you just needed to know the longest word made up of two other words in the list?

#476. 16.10 Solution 1: Can you count the number of people alive in each year?

#477. 17.25 Start by grouping the dictionary by the word lengths, since you know each column has to be the same length and each row has to be the same length.

#478. 17.7 Discuss the naive approach: merging names together when they are synonyms. How would you identify transitive relationships? A == B, A == C, and C == D implies A == D == B == C.

#479. 16.13 Any straight line that cuts a square in half goes through the center of the square. How then can you find a line that cuts two squares in half?

#480. 17.17 Start with a brute force solution. What is the runtime?

#481. 16.22 Option #1: Do you actually need an infinite grid? Read the problem again. Do you know the max size of the grid?

#482. 16.16 Would it help to know the longest sorted sequences at the beginning and end?

#483. 17.2 Try approaching this problem recursively.