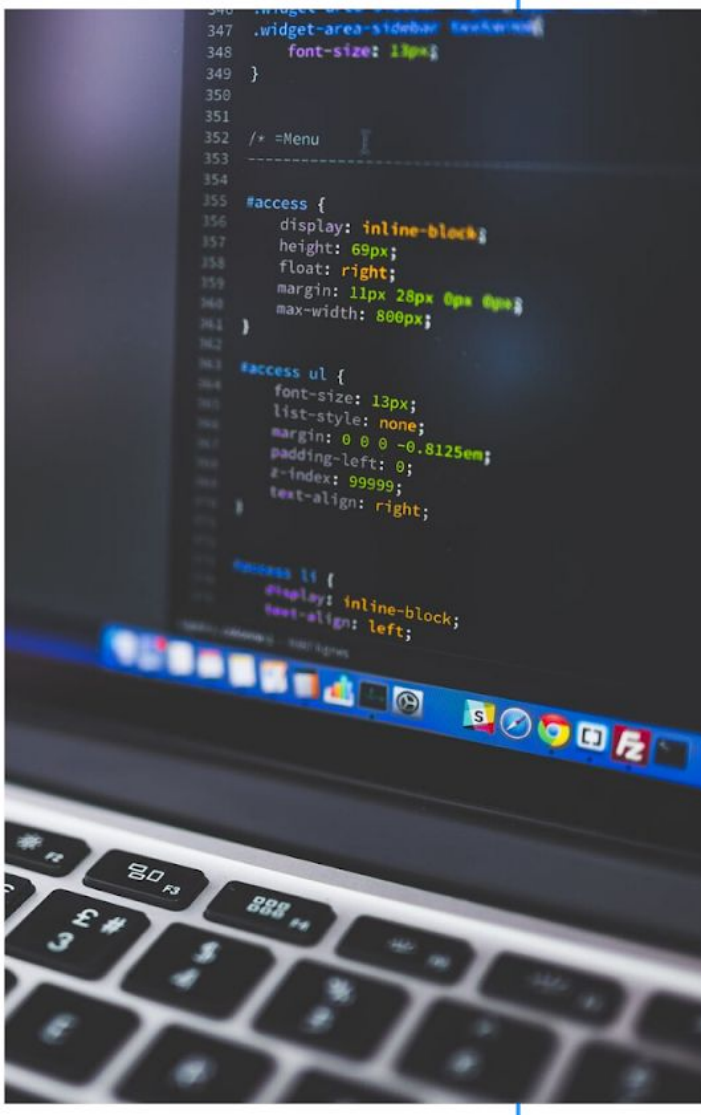
50 CODING INTERVIEW QUESTIONS

Learn to solve the questions asked in ACTUAL interviews



By Sam Gavis-Hughson
Founder of Byte by Byte

bytebybyte

Table of Contents

Introduction

Hey there! Sam here from Byte by Byte.

I'm so excited that you decided to download this guide and I want to tell you a little bit about why I created it and how to get the most out of it.

Back in 2014, I was interviewing for jobs. I was a senior at Princeton and I'll tell you, it wasn't going particularly well. Everyone else was so much smarter than me. They knew all the things that I didn't.

But I needed a job, so I didn't stop. I just kept doing interviews. Week in and week out, I had interviews. And most of them didn't go particularly well…

... but I started to get better.

And that's when I realized what I was missing: Really solid practice.

When I was doing actual interviews over and over again, I found that it started to get easier. I wasn't as nervous. I was comfortable in the interviews.

After landing multiple great job offers at companies like Amazon and Yext, I knew that I wanted to help others accomplish what I had just done. I started Byte by Byte to do just that.

Since then, I've helped thousands of students get jobs at FAANG companies (Facebook, Amazon, Apple, Netflix, Google) and many others. I've helped bootcamp graduates finally break into tech. I've helped 10-year industry veterans move into that next position.

And now I want to help you too!

Thank you so much for joining me on this journey. I can't wait to help you find your dream job in tech.

Best,

Sam Gavis-Hughson, Byte by Byte

How To Use This Guide

In this guide, you'll find 50 of the most popular coding interview questions that I've seen asked. And for every single one, I've provided code and a detailed video explanation.

But here's the thing. It's not going to be enough for you just to read through the whole thing and head off to your interview. If you really want to get the most out of this guide, you're going to have to put in the work.

The key when you study these problems is to solve them exactly as you will when you go into your interview. Developing a systematic approach and practicing it will allow you not only to solve these problems, but solve so many others in your interview.

Here are the steps that you should follow:

1. **Understand the problem**
   While this may sound obvious, it is a step that is constantly missed. My favorite go-to problem when evaluating people is asking them to print out a linked list in reverse order. Consistently, at least half of the people will spend a lot of time reversing the linked list and returning it, rather than using a simple stack-based or recursive solution that prints out the list and returns nothing.

   Understanding exactly what is being asked of you is critical to your success. Ask any clarifying questions necessary (you can also ask later so you don't have to figure out all your questions now). Also consider the example you're given. Look at the input and figure out how that resulted in the given output. If you see any obvious problem areas or edge cases, add in your own example. Based on the input and output, what should be the signature of your function?

2. **Find a brute force solution**
   While I've already somewhat belabored the point here, it is absolutely essential that you start with a brute force solution. Far too many people try to jump into an optimized solution and get lost. At that point it can be really hard to recover without simply starting over, and in an interview you definitely don't have time to start over.

   Finding a brute force solution also guarantees that you find a solution. When searching for a brute force solution, there are a couple of things you can try. If a question asks for the most something (longest path, largest weight, etc), you can solve it by finding every possible solution and comparing them.

   If you're unsure how to approach a problem, try solving it by hand first and then codifying your approach into an algorithm. Remember that your solution can be super inefficient. It doesn't matter at this point.

3. **Optimize the brute force solution**
   Here is where you have the chance to shine. You shouldn't spend more than 5-10 minutes on this step before moving on to coding, but this is your chance to make your algorithm as efficient as you can.

   There are plenty of approaches you can try here, including brainstorming more efficient data structures, looking at duplicated or unnecessary work that you're performing, or just looking for more efficient solutions unrelated to your brute force solution. The key is, though, to not spend too much time here before simply selecting the best solution you have and starting to code it up.

4. **Code the solution**
   Now that you've done all the legwork, coding your solution should hopefully be the easy part. If not, it's worth dedicating time to practicing coding, particularly on paper or a whiteboard.

5. **Test the solution**
   This final step is critical, and something that many people do wrong. Not only does it show your interviewer that you are careful and thorough, it gives you yourself the confidence that your solution is correct. How would it feel to Solution your interviewers "Is your solution correct?" with a confident "Yes!"?

   The key to testing your code is to actually go through the code line by line. Too many people simply talk through the logic. This is a good first step, but it guarantees that you miss small typos or indexing errors. Missing these could leave a bad taste in your interviewer's mouth even if you did well otherwise. Therefore you should pick a simple example and go through the code in its entirety.

With a defined process at your side, there is no reason to be nervous in an interview. Even if you see a problem that confuses the hell out of you, you know where to begin and can simply take it one step at a time.

1. Median of Arrays

Question: Find the median of two sorted arrays.

eg.

```
arr1 = [1, 3, 5]
arr2 = [2, 4, 6]
median(arr1, arr2) = 3.5
```

Solution: https://www.byte-by-byte.com/median/

---

2. 0-1 Knapsack

Question: Given a list of items with values and weights, as well as a max weight, find the maximum value you can generate from items where the sum of the weights is less than the max.

eg.

```
items = {(w:1, v:6), (w:2, v:10), (w:3, v:12)}
maxWeight = 5
knapsack(items, maxWeight) = 22
```

Solution: https://www.byte-by-byte.com/01knapsack/

3. Matrix product

Question: Given a matrix, find the path from top left to bottom right with the greatest product by moving only down and right.

eg.

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

1 -> 4 -> 7 -> 8 -> 9
2016

[-1, 2, 3]
[4, 5, -6]
[7, 8, 9]

-1 -> 4 -> 5 -> -6 -> 9
1080
```

Solution: https://www.byte-by-byte.com/matrixproduct/

4. Find Duplicates

Question: Given an array of integers where each value 1 <= x <= len(array), write a function that finds all the duplicates in the array.

eg.

```
dups([1, 2, 3])    = []
dups([1, 2, 2])    = [2]
dups([3, 3, 3])    = [3]
dups([2, 1, 2, 1]) = [1, 2]
```

Solution: https://www.byte-by-byte.com/findduplicates/

---

5. Consecutive Array

Question: Given an unsorted array, find the length of the longest sequence of consecutive numbers in the array.

eg.

```
consecutive([4, 2, 1, 6, 5]) = 3, [4, 5, 6]
consecutive([5, 5, 3, 1]) = 1, [1], [3], or [5]
```

Solution: https://www.byte-by-byte.com/consecutivearray/

6. Zero Matrix

Question: Given a boolean matrix, update it so that if any cell is true, all the cells in that row and column are true.

eg.

```
[true,  false, false]      [true,  true,  true ]
[false, false, false]  ->  [true,  false, false]
[false, false, false]      [true,  false, false]
```

Solution: https://www.byte-by-byte.com/zeromatrix/

---

7. Square Submatrix

Question: Given a 2D array of 1s and 0s, find the largest square subarray of all 1s.

eg.

```
subarray([1, 1, 1, 0]
         [1, 1, 1, 1]
         [1, 1, 0, 0]) = 2
```

Solution: https://www.byte-by-byte.com/squaresubmatrix/

8. Merge K Arrays

Question: Given k sorted arrays, merge them into a single sorted array.

eg.

```
merge({{1, 4, 7},{2, 5, 8},{3, 6, 9}}) = {1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Solution: https://www.byte-by-byte.com/mergekarrays/

---

9. Matrix Search

Question: Given an n x m array where all rows and columns are in sorted order, write a function to determine whether the array contains an element x.

eg.

```
contains([1,  2,  3,  4]
         [5,  6,  7,  8]
         [9, 10, 11, 12]) = True
```

Solution: https://www.byte-by-byte.com/matrixsearch/

10. Merge Arrays

Question: Given 2 sorted arrays, A and B, where A is long enough to hold the contents of A and B, write a function to copy the contents of B into A without using any buffer or additional memory.

eg.

```
A = {1,3,5,0,0,0}
B = {2,4,6}
mergeArrays(A, B)
A = {1,2,3,4,5,6}
```

Solution: https://www.byte-by-byte.com/mergearrays/

---

11. Zero Sum Subarray

Question: Given an array, write a function to find any subarray that sums to zero, if one exists.

eg.

```
zeroSum({1, 2, -5, 1, 2, -1}) = [2, -5, 1, 2]
```

Solution: https://www.byte-by-byte.com/zerosum/

12. Permutations

Question: Write a function that returns all permutations of a given list.

eg.

```
permutations({1, 2, 3})
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]
```

Solution: https://www.byte-by-byte.com/permutations/

13. N Stacks

Question: Implement N > 0 stacks using a single array to store all stack data (you may use auxiliary arrays in your stack object, but all of the objects in all of the stacks must be in the same array). No stack should be full unless the entire array is full.

eg.

```
N = 3;
capacity = 10;
Stacks stacks = new Stacks(N, capacity);
stacks.put(0, 10);
stacks.put(2, 11);
stacks.pop(0) = 10;
stacks.pop(2) = 11;
```

Solution: https://www.byte-by-byte.com/nstacks/

14. Anagrams

Question: Given two strings, write a function to determine whether they are anagrams.

eg.

```
isAnagram("", "") = true
isAnagram("A", "A") = true
isAnagram("A", "B") = false
isAnagram("ab", "ba") = true
isAnagram("AB", "ab") = true
```

Solution: https://www.byte-by-byte.com/anagrams/

15. Build Order

Question: Given a list of packages that need to be built and the dependencies for each package, determine a valid order in which to build the packages.

eg.

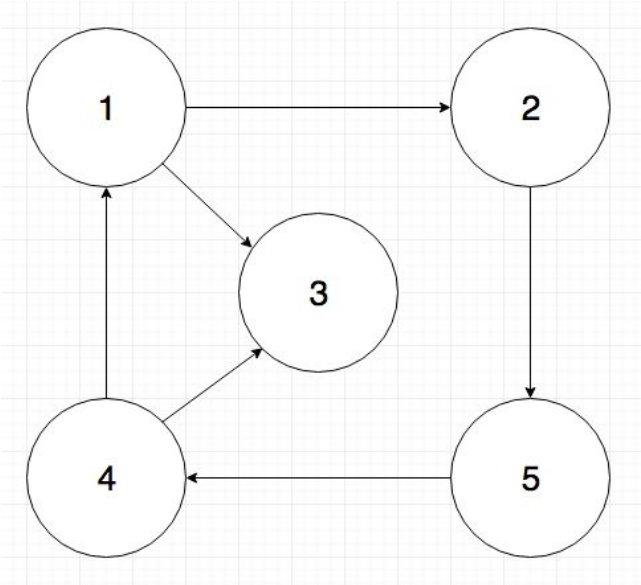
```
0:
1: 0
2: 0
3: 1, 2
4: 3
output: 0, 1, 2, 3, 4
```

Solution: https://www.byte-by-byte.com/buildorder/

16. Shortest Path

Question: Given a directed graph, find the shortest path between two nodes if one exists.

eg.



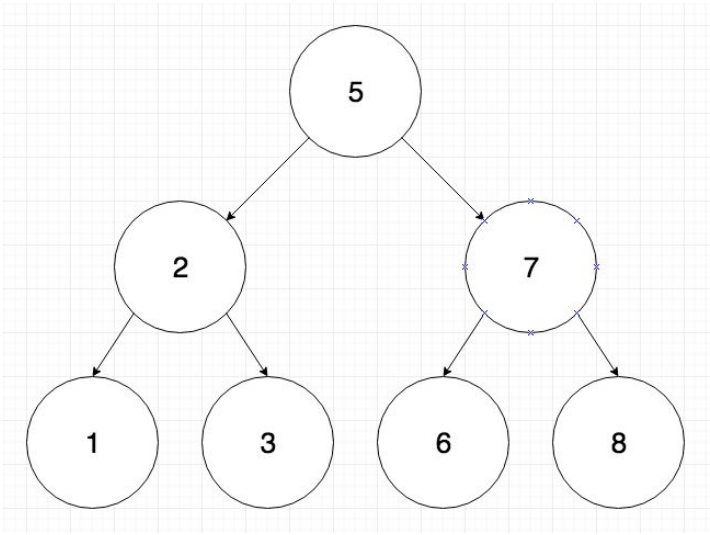

```
shortestPath(2, 3) = 2 -> 5 -> 4 -> 3
```

Solution: https://www.byte-by-byte.com/shortestpath/

17. Random Binary Tree

Question: Implement a binary tree with a method getRandomNode() that returns a random node.

eg.



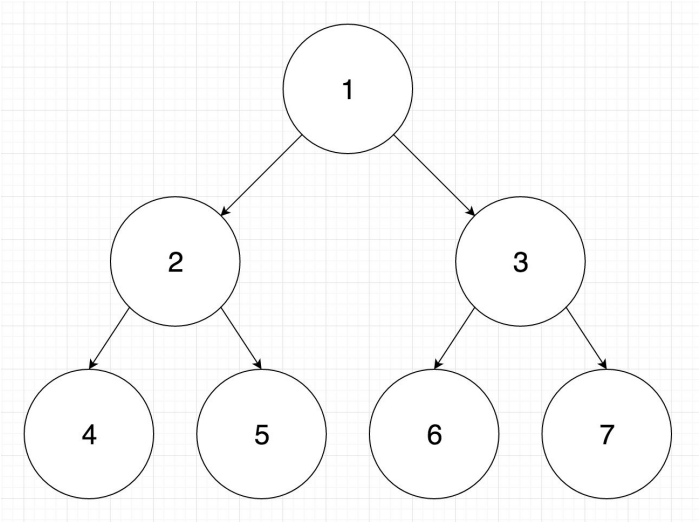

```
getRandomNode() = 5
getRandomNode() = 8
getRandomNode() = 1
```

Solution: https://www.byte-by-byte.com/randombinarytree/

18. Lowest Common Ancestor

Question: Given two nodes in a binary tree, write a function to find the lowest common ancestor.

eg.




```
lcs(4, 3) = 1
lcs(6, 7) = 3
```

Solution: https://www.byte-by-byte.com/lowestcommonancestor/

---

19. Sum

Question: Given two integers, write a function to sum the numbers without using any arithmetic operators.

Solution: https://www.byte-by-byte.com/sum/

20. Reverse Stack

Question: Given a stack, reverse the items without creating any additional data structures.

eg.

```
reverse(1->2->3) = 3->2->1
```

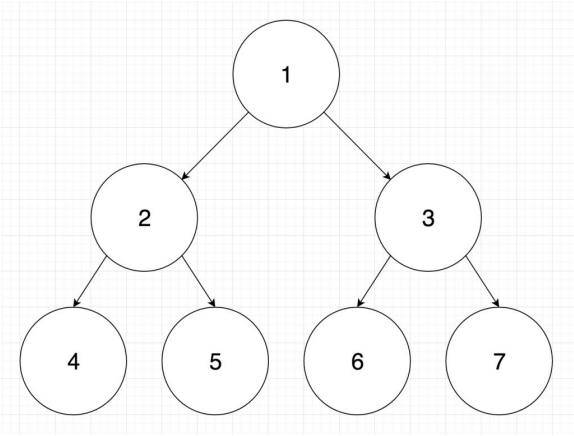
Solution: https://www.byte-by-byte.com/reversestack/

---

21. Tree to Doubly Linked List

Question: Given a tree, write a function to convert it into a circular doubly linked list from left to right by only modifying the existing pointers.

eg.



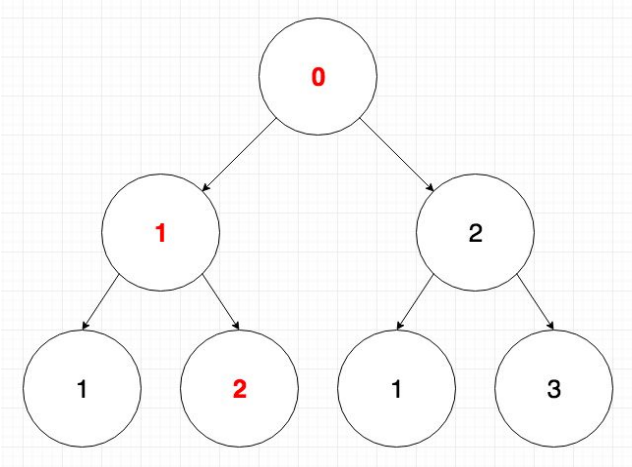

```
<- 4 <-> 2 <-> 5 <-> 1 <-> 6 <-> 3 <-> 7 ->
```

Solution: https://www.byte-by-byte.com/treetolist/

22. Longest Consecutive Branch

Question: Given a tree, write a function to find the length of the longest branch of nodes in increasing consecutive order.

eg.




```
length = 3
```

Solution: https://www.byte-by-byte.com/longestbranch/

23. Print Reversed Linked List

Question: Given a linked list, write a function that prints the nodes of the list in reverse order.

eg.

```
printReversedList(1 -> 2 -> 3)
3
2
1
```

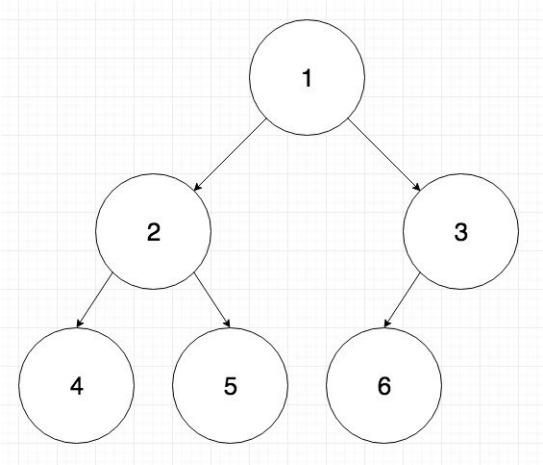
Solution: https://www.byte-by-byte.com/printreversedlist/

24. Balanced Binary Tree

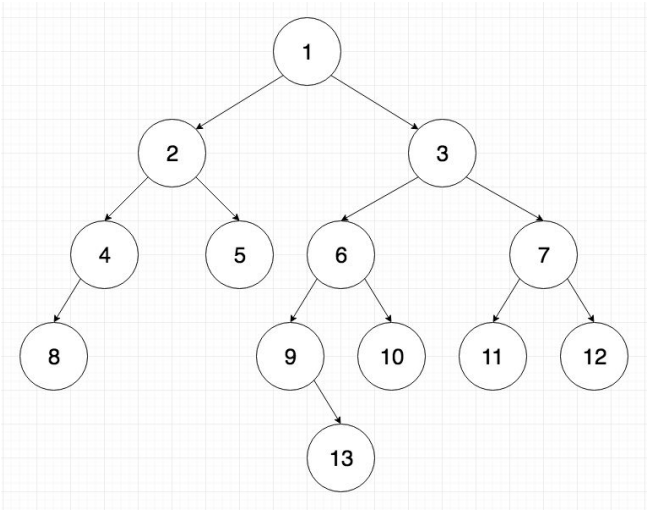
Question: Given a binary tree, write a function to determine whether the tree is balanced.

eg.

Balanced tree (all branches are same height +- 1):



Balanced tree (all subtrees are balanced):



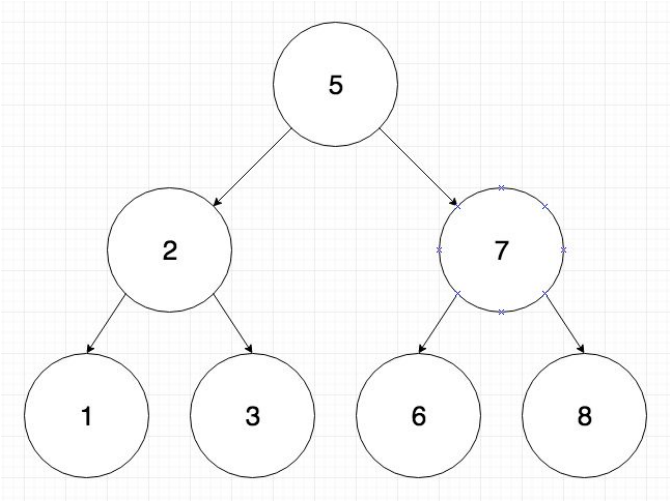
Solution: https://www.byte-by-byte.com/balancedtree/

25. Binary Search Tree Verification

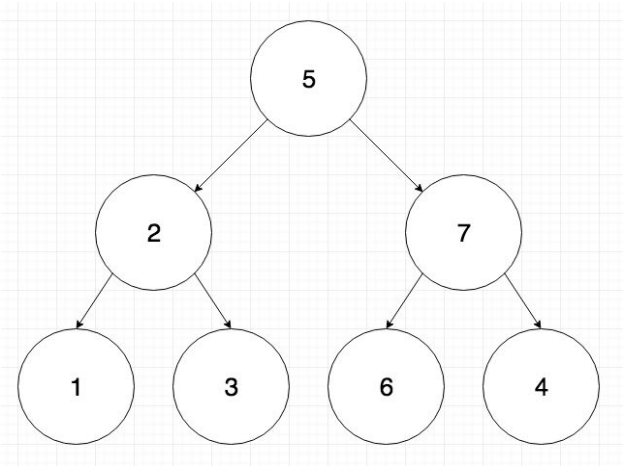
Question: Given a binary tree, write a function to test if the tree is a binary search tree.

eg.

Valid:



Invalid:



Solution: https://www.byte-by-byte.com/binarysearchtree/

26. Smallest Change

Question: Given an input amount of change x, write a function to determine the minimum number of coins required to make that amount of change.

eg. (using American coins)

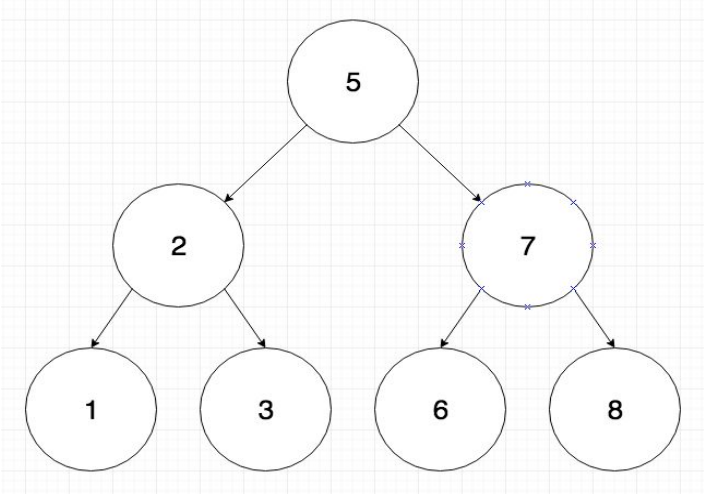
```
change(1) = 1
change(3) = 3
change(7) = 3
change(32) = 4
```

Solution: https://www.byte-by-byte.com/smallestchange/

27. Inorder Traversal

Question: Given a binary search tree, print out the elements of the tree in order without using recursion.

eg.




```
printTree(tree)
1
2
3
5
6
7
8
```

Solution: https://www.byte-by-byte.com/inordertraversal/

28. Sort Stacks

Question: Given a stack, sort the elements in the stack using one additional stack.

eg.

```
sort([1, 3, 2, 4]) = [1, 2, 3, 4]
```

Solution: https://www.byte-by-byte.com/sortstacks/

---

29. Stack from Queues

Question: Implement a LIFO stack with basic functionality (push and pop) using FIFO queues to store the data.

Solution: https://www.byte-by-byte.com/stackfromqueues/

---

30. Palindromes

Question: Given a linked list, write a function to determine whether the list is a palindrome.

eg.

```
palindrome(1 -> 2 -> 3) = false
palindrome(1 -> 2 -> 1) = true
```

Solution: https://www.byte-by-byte.com/palindromes/

31. Max Stacks

Question: Implement a LIFO stack that has a push(), pop(), and max() function, where max() returns the maximum value in the stack. All of these functions should run in O(1) time.

eg.

```
push(1)
max() = 1
push(2)
max() = 2
push(1)
max() = 2
pop() = 1
max() = 2
pop() = 2
max() = 1
```

Solution: https://www.byte-by-byte.com/maxstack/

32. Two Missing Numbers

Question: Given an array containing all the numbers from 1 to n except two, find the two missing numbers.

eg.

```
missing([4, 2, 3]) = 1, 5
```

Solution: https://www.byte-by-byte.com/twomissingnumbers/

---

33. Big Int Modules

Question: Given a list of bytes a, each representing one byte of a larger integer (ie. {0x12, 0x34, 0x56, 0x78} represents the integer 0x12345678), and an integer b, find a % b.

eg.

```
mod({0x03, 0xED}, 10) = 5
```

Solution: https://www.byte-by-byte.com/bigintmod/

34. Swap Variables

Question: Given two integers, write a function that swaps them without using any temporary variables.

Solution: https://www.byte-by-byte.com/swapvariables/

---

35. Gray Code

Question: Given two integers, write a function to determine whether or not their binary representations differ by a single bit.

eg.

```
gray(0, 1) = true
gray(1, 2) = false
```

Solution: https://www.byte-by-byte.com/graycode/

36. Rotate Bits

Question: Given a number, write a function to rotate the bits (ie circular shift).

eg.

```
rotate(0xFFFF0000, 8) = 0x00FFFF00
rotate(0x13579BDF, 12) = 0xBDF13579
rotate(0b10110011100011110000111110000000, 17) =
0b00011111000000010110011100011110
```

Solution: https://www.byte-by-byte.com/rotatebits/

---

37. Number of Ones in a Binary Number

Question: Given an integer, write a function to compute the number of ones in the binary representation of the number.

eg.

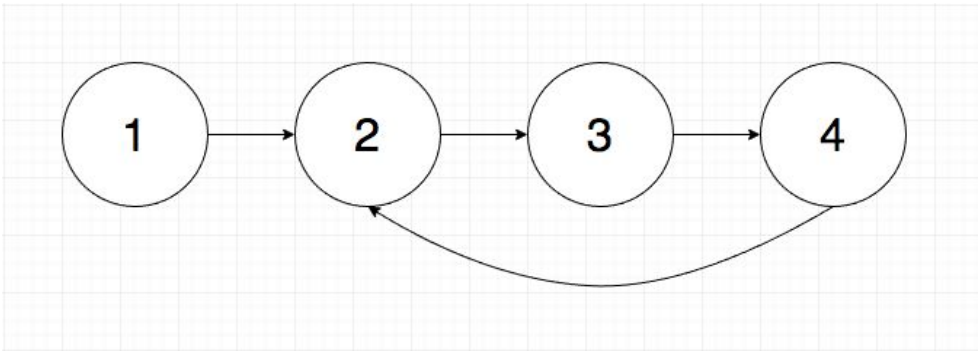
```
ones(0) = 0
ones(1) = 1
ones(2) = 1
ones(3) = 2
ones(7) = 3
```

Solution: https://www.byte-by-byte.com/onesinbinary/

38. Linked List Cycles

Question: Given a linked list, determine whether it contains a cycle.

eg.



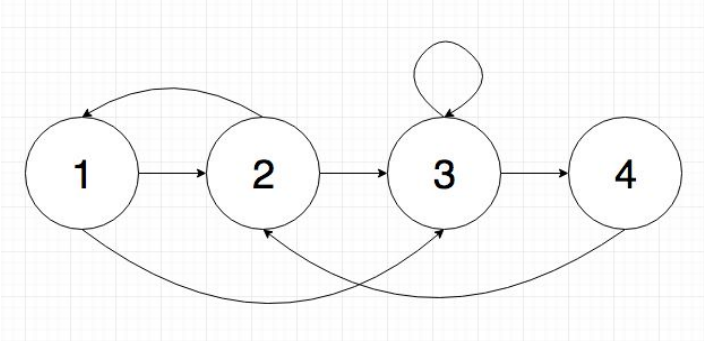

```
1 -> 2 -> 3 -> 4
     ^          |
     |__________|
```

Solution: https://www.byte-by-byte.com/listcycles/

39. Random Linked List

Question: Given a linked list where each node has two pointers, one to the next node and one to a random node in the list, clone the linked list.

eg.




```
1 -> 2 -> 3 -> 4 -> null
|    |    |    |
v    v    v    v
3    1    3    2
```

Solution: https://www.byte-by-byte.com/randomlinkedlist/

---

40. Dedup Linked List

Question: Given an unsorted linked list, write a function to remove all the duplicates.

eg.

```
dedup(1 -> 2 -> 3 -> 2 -> 1) = 1 -> 2 -> 3
```

Solution: https://www.byte-by-byte.com/deduplinkedlist/

41. Split a Linked List

Question: Given a linked list, write a function to split the list into two equal halves.

eg.

```
divide(1 -> 2 -> 3 -> 4) = 1 -> 2, 3 -> 4
divide(1 -> 2 -> 3 -> 4 -> 5) = 1 -> 2 -> 3, 4 -> 5
```

Solution: https://www.byte-by-byte.com/splitlinkedlist/

---

42. Nth to the Last Element

Question: Given a linked list, and an input n, write a function that returns the nth-to-last element of the linked list.

eg.

```
list = 1 -> 2 -> 3 -> 4 -> 5 -> null
nthToLast(list, 0) = 5
nthToLast(list, 1) = 4
nthToLast(list, 4) = 1
nthToLast(list, 5) = null
```

Solution: https://www.byte-by-byte.com/nthtolastelement/

43. Three Sum

Question: Given a list of integers, write a function that returns all sets of 3 numbers in the list, a, b, and c, so that a + b + c == 0.

eg.

```
threeSum({-1, 0, 1, 2, -1, -4})
[-1, -1, 2]
[-1, 0, 1]
```

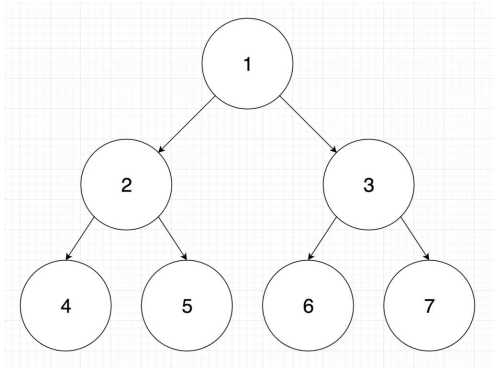
Solution: https://www.byte-by-byte.com/threesum/

---

44. Tree Level Order

Question: Given a tree, write a function that prints out the nodes of the tree in level order.

eg.




```
traverse(tree) = 1 2 3 4 5 6 7
```

Solution: https://www.byte-by-byte.com/treelevelorder/

45. Autocomplete

Question: Write an autocomplete class that returns all dictionary words with a given prefix.

eg.

```
dict:   {"abc", "acd", "bcd", "def", "a", "aba"}
prefix: "a" -> "abc", "acd", "a", "aba"
prefix: "b" -> "bcd"
```

Solution: https://www.byte-by-byte.com/autocomplete/

---

46. String Deletion

Question: Given a string and a dictionary HashSet, write a function to determine the minimum number of characters to delete to make a word.

eg.

```
dictionary: ["a", "aa", "aaa"]
query: "abc"
output: 2
```

Solution: https://www.byte-by-byte.com/stringdeletion/

47. Longest Common Substring

Question: Given two strings, write a function that returns the longest common substring.

eg.

```
longestSubstring("ABAB", "BABA") = "ABA"
```

Solution: https://www.byte-by-byte.com/longestsubstring/

---

48. String Compression

Question: Given a string, write a function to compress it by shortening every sequence of the same character to that character followed by the number of repetitions. If the compressed string is longer than the original, you should return the original string.

eg.

```
compress("a") = "a"
compress("aaa") = "a3"
compress("aaabbb") = "a3b3"
compress("aaabccc") = "a3b1c3"
```

Solution: https://www.byte-by-byte.com/stringcompression/

49. Fibonacci Number

Question: Given an integer n, write a function to compute the nth Fibonacci number.

eg.

```
fibonacci(1) = 1
fibonacci(5) = 5
fibonacci(10) = 55
```

Solution: https://www.byte-by-byte.com/fibonacci/

---

50. Priority Queue

Question: Implement a Priority Queue

Solution: https://www.byte-by-byte.com/priorityqueue/

---

51. Kth Most Frequent String

- **Question**: Given a list of strings, write a function to get the kth most frequently occurring string.
- Eg.

```
kthMostFrequent({"a","b","c","a","b","a"}, 0) = "a"
kthMostFrequent({"a","b","c","a","b","a"}, 1) = "b"
kthMostFrequent({"a","b","c","a","b","a"}, 2) = "c"
kthMostFrequent({"a","b","c","a","b","a"}, 3) = null
```

Solution: https://www.byte-by-byte.com/kthmostfrequentstring/