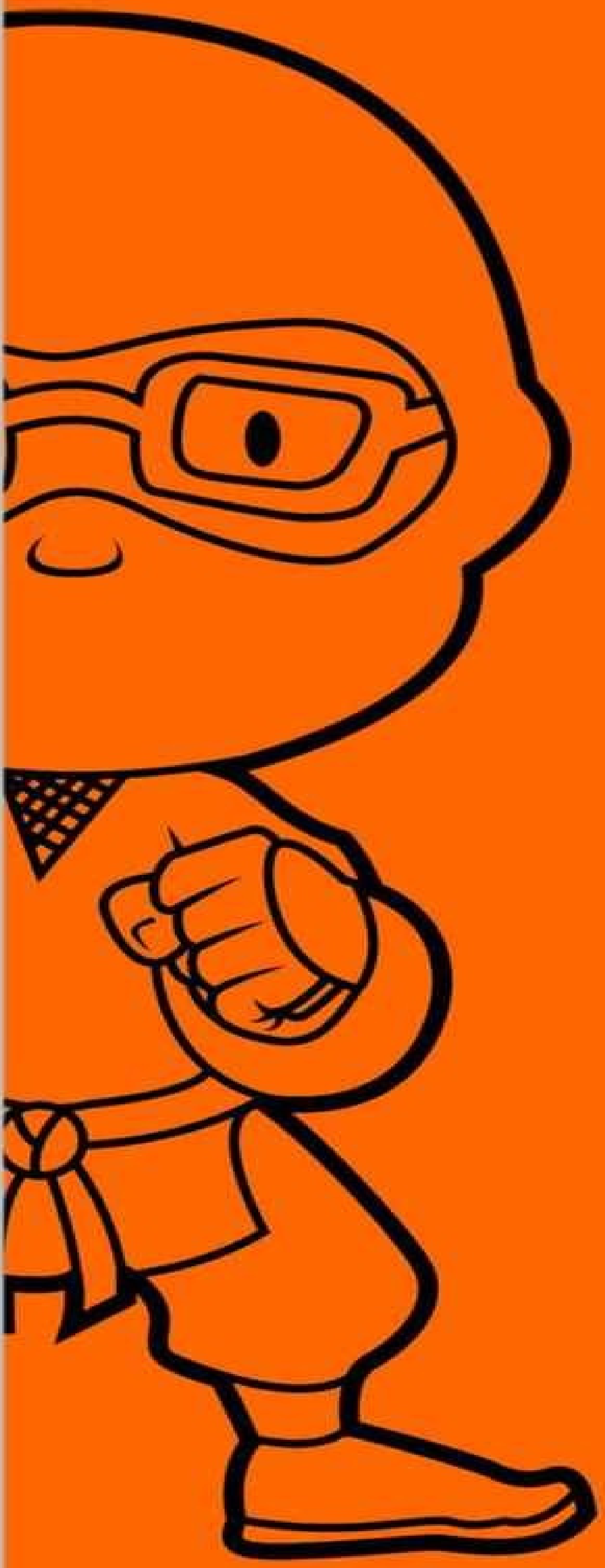
CODING INTERVIEW NINJA

50 coding questions with Java solutions to practice for your coding interview.

EKIM OUYE



Introduction

Getting your dream software engineering job could be a matter of how well you perform in your coding interview part. Perhaps it is the most important part of your interview process.

Your recruiter will recommend you to read again your university algorithms and data structures book to brush up on Computer Science fundamentals. And although this is necessary, it is not enough. The types of questions that you will find in an algorithms book are not designed to be solved under pressure in a short 45-minutes period. The best way to prepare yourself for the coding interview is to practice on similar questions to the ones that you will be asked to solve. This is the aim of this book; to present you some sample interview coding questions with a sample solution code.

Since every company has a slightly different interview process and question style, it is highly recommended after you finish practicing with the questions of this book, to take a look at questions that have been recently asked to candidates from sites such as CareerCup and Glassdoor.

Don’t forget that you don’t have to know everything and solve perfectly all the questions that you will be asked in order to proceed to the next phase of the interview process. Most of the times, the interviewers just want to see your thought process when dealing with new problems and how well you know to code.

Finally, if you find something that you believe it is wrong, don’t hesitate to get in touch with me.

Good luck with your interviews!

Preparation

Before starting practicing for the coding interview, you will need to brush up some of the fundamentals of computer science. Below is a list of the most important topics, in my opinion, that you need to look into. This list is by no means a complete one.

- **PROGRAMMING LANGUAGE**

You must know *really well* a programming language of your choice. Being fluent in the language of your choice will give you more time to think about the actual problem rather than thinking how to implement the solution that you have in your mind. The most popular languages among the tech companies are Java, C/C++ and Python.

- **RUNNING TIME COMPLEXITY**

Most of the times, the interviewer will ask you the running time of your program meaning the big O complexity, that is the worst-case analysis of the running time. In some cases, such as when dealing with popular algorithms that have a bad big O analysis (eg. quicksort), it is useful to know the average running time as well.

- **DATA STRUCTURES**

  - *Linked Lists*

Each node in a linked list has a data element and a pointer to the next node. A sample definition of a linked list node is:

```
class LLNode {

 int data;

LLNode next;



LLNode(int data) {

this.data = data;

                                   }


                                   }
```

A linked list may be double linked, which means that each node has a pointer to

A linked list may be double linked, which means that each node has a pointer to the previous node as well. A sample definition would be:

```
class DLLNode {

 int data;

DLLNode next, prev;



DLLNode(int data) {

this.data = data;

                              }

                              }
```

You can insert a new node in a linked list in O(1) time. To look up for an element in a linked list you need O(n) time.

- *Trees*

Each node in a tree has a data element and a list of pointers to its children. The most popular form of trees are the binary trees which have only two children. A sample definition, that is used throughout all the practice questions that follow, is:

```
class TNode {

 int data;

TNode left, right;



TNode(int data) {

this.data = data;

                                   }

                                   }
```

In some cases the tree nodes could have a pointer to their father-node.

A popular specialization of a binary tree is the *Binary Search Tree (BST)* where each node's element must be greater than every element in its left subtree and less than every element in its right subtree.

The process of visiting each node in a tree is called tree traversal. There are three basic ways to explore a tree in a depth-first order. The following sample code prints a tree with the three basic traversals:

```
public static void preOrder(TNode root) {

if (root == null) { return; };

System.out.println(root.data);

preOrder(root.left);

preOrder(root.right);


                                        }



public static void inOrder(TNode root) {

if (root == null) { return; };
inOrder(root.left);

System.out.println(root.data);

inOrder(root.right);

}

public static void postOrder(TNode root) {

if (root == null) { return; };

postOrder(root.left);

postOrder(root.right);

System.out.println(root.data);

}
```

- *Hash Tables*

A hash map is used to associate a key with a value. The advantage of this data structure is that theoretically each operation (insertion, removal, look-up) requires on average O(1) time.

To insert a new element into the hash map, you compute the hash code of the key. If the hash code that was computed already exist in the structure, then a collision resolution technique is used. The two most popular resolution techniques are:

- Separate Chaining: Each bucket is independent and some sort of dynamic list is used for every hash code index.

- Open Addressing: The buckets are examined until an unoccupied bucket is found.

To create a new hash map data structure with String key and values in Java:

```
HashMap<String, String> hm = new HashMap<String, String>();
```

This is maybe the most popular data structure among interviewers!

- *Stacks*

A Last-In-First-Out data structure. It would be useful to know how to implement a stack from scratch using arrays since an interviewer could ask to create a modified stack data structure with special functionality. For all the other cases that you might need to use a stack, you can use the built-in implementation of your language. To use a stack of Strings in Java:

```
Stack<String> stack = new Stack<String>();
```

- *Queues*

A First-In-First-Out data structure. It would be useful to know how to implement a queue from scratch using arrays since an interviewer could ask to create a modified queue data structure. To use a queue of Strings in Java:

```
Queue<String> queue = new LinkedList<String>();
```

- *Graphs*

A set of nodes that are connected by links. Can be represented using an adjacency matrix or an adjacency list. The graph theory can be asked by interviewers since it is used extensively in many popular algorithms.

- *Tries*

A tree data structure that usually holds characters and has many applications in string manipulation algorithms. Usually, all the descendants of a node have a common prefix of the string associated with that node and the root is associated with the empty string. A popular application of this data structure is the look up in a dictionary of words (such as a simple auto-complete functionality in a text box).

- **ALGORITHMS**

Its really rare for an interviewer to ask you to implement a specific complicated and long algorithm (such as Dijkstra, A*, etc). However, you should have an idea how these work, what they are used for and to be able to conduct a basic conversation when the subject is involving a well-known algorithm. Also, it is recommended to know what and how Dynamic Programming works and to be able to recognize it. The following are popular simpler algorithms that is suggested to know how to implement in case the interviewer asks you to implement a modification of the algorithm:

- *Breadth-first search (DFS)*
- *Depth-first search (BFS)*
- *Mergesort*
- *Quicksort*
- *Binary Search*

- **OTHER**

These are techniques or concepts that an interviewer might ask you, even only theoretically.

- *Recursion*

A powerful technique that is used to solve computer science problems and asked frequently in interviews. Some times makes the solution of a problem simpler and produces compact code. Remember that this technique requires more memory than an iterative solution. Also, when writing a recursive solution don’t forget the terminating condition of the recursion.

- *Object Oriented Design*

Interviewers might ask you to describe the basic objects for a given system. Therefore, knowing the fundamentals of object oriented principles is crucial.

- *Testing*

Interviewers might ask you to test your own solution. You need to know how to create test cases that cover even the edge cases (that is having input something unexpected). Also, knowing the basic principles of Unit Testing and mentioning it would be really recommended.

- *System Programming*

Concepts such as threading, locks and mutex might be asked, especially if the company has a theoretical part in their interview.

- *Bitwise*

Binary operations using bitwise operators might be asked from time to time especially in companies specializing in lower level software.

- *Program memory*

The difference between the stack and the heap memory areas and when each one is used.

Practise Questions

The following practice questions are consisted from a concise description, a short example of the input and output required if applicable, the sample solution coded in Java and the time and space complexity if applicable. The built-in classes and utility functions of Java are used wherever possible since the aim of an interview is not to re-invent the wheel but to come up with an efficient solution to the asked question.

You will notice that the selected questions and their answers are not very long. This is due to the limited time that you have in an interview that does not allow the interviewer to ask a question that requires a long solution. If you are come up with a really complicated and long solution to a question, you be thinking too complicated. The candidate in less than 45 minutes would need time to clarify the question, come up with the solution, time to discuss the solution with the interviewer and time to test the solution.

In most of the sample solutions, there is a basic input data validation, such as checking that the passed object is not a null value. This input validation might not be exhaustive and you should let your interviewer know that you are skipping the input validation or you are not doing an exhaustive check. Have in mind that this might result in a follow-up question on how to test your program.

In case that the solution to a question needs to be called in a specific way from the main program then this driver program’s calls are noted as follow:

.

.

.

main program calls()

In a coding interview, you might be asked theoretical questions and open-ended system design questions as well. These are not in the scope of this book but the final two questions are consisted by some samples of what you might be asked.

1. *Find the majority element. In this problem, majority element is defined as the number that appears more than n/2 times in an array of numbers.*

e.g:

input: 3,2,2,1,2,2,1

output: 2

____

SOLUTION:

```
public static int findMajCandidate(int[] arr) {

if (arr == null || arr.length == 0) {

 return Integer.MIN_VALUE;

                                                  }

int count = 1, maj = arr[0];
```

```
for (int i = 1; i < arr.length; i++) {

if (arr[i] == maj) {

count++;

} else {

count--;

                    }



if (count == 0) {

maj = arr[i];

count = 1;

                    }

                    }
```