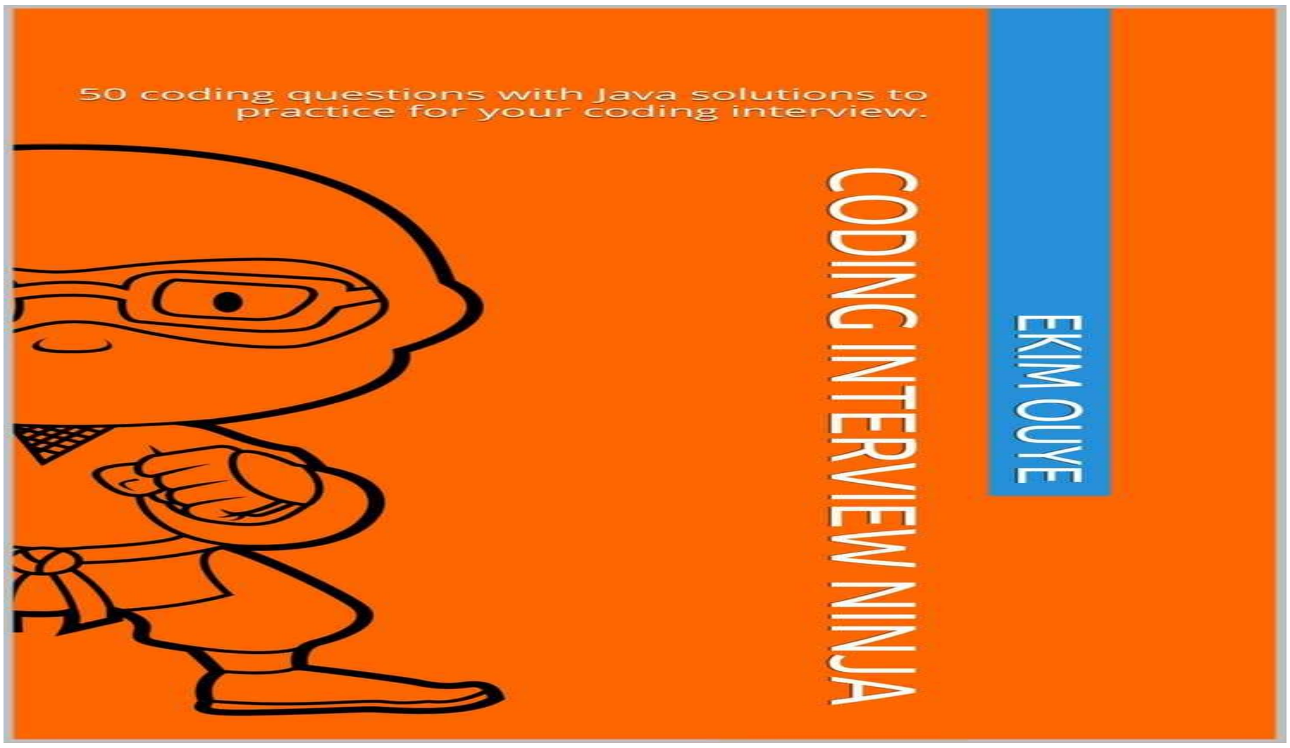
50 coding questions with Java solutions to practice for your coding interview.
CODING INTERVIEW NINJA
EKIM OUYE

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 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;
		}
	}

	return maj;
}

public static boolean verifyMajCandidate(int[] arr, int maj) {
	if (arr == null || arr.length == 0) {
		return false;
	}

	int n = arr.length, count = 0;
	for (int i = 0; i < n; i++) {
		if (arr[i] == maj) {
			count++;
		}
		if (count > n / 2) {
			return true;
		}
	}
	return false;
}


.
.
.
int maj = findMajCandidate(arr);
if (maj == Integer.MIN_VALUE || verifyMajCandidate(arr, maj)==false) {
	System.out.println("No majority element found.");
} else {
	System.out.println("Majority element: " + maj);
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

2. Find the maximum sum of a continuous subsequence in an array of integers.
e.g:
input: 1,-3,2,5,-8
output: 7

____

SOLUTION:

```java
public static int maxSubSequence(int[] arr) {
	if (arr == null || arr.length == 0) {
		return Integer.MIN_VALUE;
	}

	int maxSoFar = arr[0], curMax = arr[0];
	for (int i = 1; i < arr.length; i++) {
		curMax = Math.max(curMax + arr[i], arr[i]);
		maxSoFar = Math.max(curMax, maxSoFar);
	}

	return maxSoFar;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

3. Given two binary trees find if they are equal (have the same content and structure).

____

SOLUTION:

```
public static boolean isTreeSame(TNode T1, TNode T2) {
        if (T1 == null && T2 == null) {
                return true;
        }
        if (T1 == null || T2 == null) {
                return false;
        }
        if (T1.data != T2.data) {
                return false;
        }
        return isTreeSame(T1.left, T2.left)
                && isTreeSame(T1.right, T2.right);
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

4. In a Binary Search Tree (BST) find the number of elements in a given range.
e.g:
input:

```
        range: (2,6)
        tree:
                5
        3               7
1
```

output: 2

____

SOLUTION:

```
public static int numbersInRange(TNode node, int start, int end) {
        if (node == null) {
                return 0;
        }

        if ((node.data > start) && (node.data < end)) {
                return 1
                + numbersInRange(node.left, start, end)
                + numbersInRange(node.right, start, end);
        } else if (node.data >= end) {
                return numbersInRange(node.left, start, end);
        } else { // if (node.data <= start)
                return numbersInRange(node.right, start, end);
        }
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

5. *Given a tree that its nodes have pointers to their parents, find the next node in an in-order traversal.*

e.g:
input:
```
        node: 3
        tree:
                5
        3               7
1

output: 5
```
____

SOLUTION:

```java
public static TNode nextInOrder(TNode n) {
        if (n == null) {
                return null;
        }

        if (n.right != null) {
                n = n.right;
                while (n.left != null) {
                        n = n.left;
                }
                return n;
        } else {
                TNode p = n.parent;
                while (p != null && p.left != n) {
                        n = p;
                        p = p.parent;
                }
                return p;
        }
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

6. *Given a tree that its nodes have pointers to their parents, find the next node in a post-order traversal.*

e.g:
input:

```
        node: 7
        tree:
                5
        3               7
1

output: 5
```

____

SOLUTION:

```
public static TNode nextPostOrder(TNode n) {
        if (n == null || n.parent == null) {
                return null;
        }

        if (n.parent.left == n) {
                TNode h = n.parent.right;
                if (h == null) {
                        return n.parent;
                }
                while (h.left != null) {
                        h = h.left;
                }
                return h;
        } else {
                return n.parent;
        }
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

7. Design a class that you can add elements and find the mean of the last N elements.
e.g:
input:

data: 2,3,1,4,2

N: 2

output: 3

SOLUTION:

```
class DS {
	ArrayList<Integer> mData = new ArrayList<Integer>();
	ArrayList<Integer> mDataSum = new ArrayList<Integer>();

	public void add(int i) {
		mData.add(i);

		int lastIndex = mDataSum.size() - 1;
		int sum = i;
		if (lastIndex >= 0) {
			sum += mDataSum.get(lastIndex);
		}
		mDataSum.add(sum);
	}

	public float mean(int N) {
		if (N <= 0 || mDataSum.size()==0 || N > mDataSum.size()) {
			return -Float.MAX_VALUE;
		}

		int lastIndex = mDataSum.size() - 1;
		int firstIndex = Math.max(lastIndex - N, 0);
		float sum;
		sum = mDataSum.get(lastIndex)-mDataSum.get(firstIndex);
		return sum / N;
	}
}
```

RUNNING TIME COMPLEXITY:
O(1)

SPACE COMPLEXITY:
O(N)

8. Given an array of Intervals sorted by their start time, merge them into non-overlapping intervals.
e.g:
input: (1,3), (5,10), (9,31), (12,30)
output: (1,3), (5,31)

SOLUTION:

```
class Interval {
	public int start, end;
}

public static List<Interval> mergeIntervals(List<Interval> in) {
	if (in == null || in.size() == 0) {
		return null;
	}

	List<Interval> solution = new ArrayList<Interval>();
	Interval last = in.get(0);
	solution.add(last);
	for (int i = 1; i < in.size(); i++) {
		Interval current = in.get(i);
		if (current.start > last.end) {
			solution.add(current);
			last = current;
		} else {
			last.end = Math.max(last.end, current.end);
		}
	}

	return solution;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

9. Given an array of sorted integers find the position of a given number, if exists.
e.g:
input:
array: 2, 4, 5, 7, 8
num: 4
output: 1

SOLUTION:

```
public static int binarySearch(int[] arr, int num) {
        if (arr == null) {
                return Integer.MIN_VALUE;
        }

        int st = 0, end = arr.length - 1;
        while (st <= end) {
                int mid = (st + end) / 2;
                if (arr[mid] == num) {
                        return mid;
                } else if (num > arr[mid]) {
                        st = mid + 1;
                } else {
                        end = mid - 1;
                }
        }

        return Integer.MIN_VALUE;
}
```

RUNNING TIME COMPLEXITY:
O(logN)

SPACE COMPLEXITY:
O(1)

10. Find one local minimum from an array of numbers.
e.g:
input: 9, 7, 2, 8, 5, 6, 3, 4
output: 2 or 5 or 3

SOLUTION:

```
public static int findLocalMin(int[] arr, int st, int end) {
        int mid = (st + end) / 2;
        if ((mid - 1 < 0) || (mid + 1 >= arr.length)) {
                return Integer.MIN_VALUE;
        }
        if ((arr[mid - 1] > arr[mid]) && (arr[mid + 1] > arr[mid])) {
                return mid;
        } else if (arr[mid - 1] < arr[mid]) {
                findLocalMin(arr, st, mid);
        } else {
                findLocalMin(arr, mid, end);
        }
        return Integer.MIN_VALUE;
}
```

RUNNING TIME COMPLEXITY:
O(logN)

SPACE COMPLEXITY:
O(1)

11. Given an array of numbers that is first strictly increasing and then strictly decreasing, find the maximum number.
e.g:
input: 1, 2, 3, 4, 5, 4, 3, 2
output: 5

____

SOLUTION:

```
public static int findMax(int[] arr) {
        if (arr == null || arr.length <= 3) {
                return Integer.MIN_VALUE;
        }

        int st = 0, end = arr.length - 1;
        while (st <= end) {
                int mid = (st + end) / 2;
                int midVal = arr[mid];
                if (midVal > arr[mid - 1] && midVal > arr[mid + 1]) {
                        return midVal;
                } else if (arr[mid - 1] < midVal) {
                        st = mid + 1;
                } else {
                        end = mid - 1;
                }
        }

        return Integer.MIN_VALUE;
}
```

RUNNING TIME COMPLEXITY:
O(logN)

SPACE COMPLEXITY:
O(1)

12. Implement square root function (sqrt) only with integers.
e.g:
input: 4
output: 2

input: 5
output: 2 or 3
(both values allowed since the correct result is decimal)

SOLUTION:

```
public static int sqrt(int target, int st, int end) {
        if (st >= end) {
                return Math.min(st, end);
        }

        int mid = (st + end) / 2;
        if (mid * mid == target) {
                return mid;
        }

        if (mid * mid > target) {
                return sqrt(target, st, mid - 1);
        } else {
                return sqrt(target, mid + 1, end);
        }
}
```

RUNNING TIME COMPLEXITY:
O(logN)

SPACE COMPLEXITY:
O(1)

13. Given an array that was sorted but was rotated an unknown number of times, find the position of a target element.

e.g:

input:

rotated array: 4, 5, 6, 7, 0, 1, 2

target: 6

output: 2

____

SOLUTION:

```
public static int binaryRoatedSearch(int[] arr, int target) {
        if (arr == null) {
                return Integer.MIN_VALUE;
        }

        int st = 0;
        int end = arr.length - 1;
        while (st <= end) {
                int mid = (st + end) / 2;
                if (arr[mid] == target) {
                        return mid;
                }

                // at least one part must be sorted
                if (arr[st] <= arr[mid]) { // left part is sorted
                        if (arr[st] < target && target <= arr[mid]) {
                                end = mid - 1;
                        } else {
                                st = mid + 1;
                        }
                } else { // right part is sorted
                        if (arr[mid] < target && target <= arr[end]) {
                                st = mid + 1;
                        } else {
                                end = mid - 1;
                        }

                }
        }

        return Integer.MIN_VALUE;
}
```

INFO:
If the input array contains duplicate entries, the target can only be found with linear search in O(N) running time.

RUNNING TIME COMPLEXITY:
O(logN)

SPACE COMPLEXITY:
O(1)

14. Given a sorted array of integers that contains each entry multiple times, find the start and end position of a target integer.

e.g:
input:
sorted array: 1, 2, 3, 7, 7, 7, 9, 9
target: 7
output:
start position: 3
end position: 5

SOLUTION:

```
public static int modifiedBinSearch(int[] arr, double target) {
	int st = 0, end = arr.length - 1;
	while (st <= end) {
		int mid = (st + end) / 2;
		if (target > arr[mid]) {
			st = mid + 1;
		} else {
			end = mid - 1;
		}
	}

	return st;
}

public static int[] findRange(int[] arr, int target) {
	int[] res = { -1, -1 };
	if (arr == null) {
		return res;
	}

	int lo = modifiedBinSearch(arr, target - 0.5);
	if (lo >= arr.length || arr[lo] != target) {
		return res;
	}
	int hi = modifiedBinSearch(arr, target + 0.5);

	res[0] = lo;
	res[1] = hi;
	return res;

}
```

```
.
.
.
// res[0] will be the start position
// res[1] will be the end position
int[] res = findRange(arr, 7);
```

RUNNING TIME COMPLEXITY:
O(logN)

SPACE COMPLEXITY:
O(1)

15. *Given an array that at each place represents how many steps forward you can make in the array, find the minimum number of steps needed to reach the end of the array.*
e.g:
input: 2, 1, 1, 8, 1, 1, 1
output: 3

SOLUTION:

```
public static int minNumOfSteps(int[] arr) {
        if (arr == null) {
                return Integer.MIN_VALUE;
        }

        int[] opt = new int[arr.length];
        opt[0] = 0;
        for (int i=1; i<arr.length; i++) {
                int min=arr.length;
                for (int k=0; k<i; k++) {
                        if (i-k <= arr[k]) {
                                min = Math.min(min,  opt[k]+1);
                        }
                }
                opt[i] = min;
        }

        return opt[arr.length-1];
}
```

RUNNING TIME COMPLEXITY:
$O(N^2)$

SPACE COMPLEXITY:
O(N)

16. Given a 2D array of integers and a starting point, find the continuous area covered with the same value as the starting point.

e.g:
input:
starting point: (0,0)
array:
1 1 2
1 3 2
2 2 1
output: 3

———————

SOLUTION:

```
public static void findAreaHelper(int[][] arr, boolean[][] visited, int value, int x, int y) {
	if (x < 0 || x > arr[0].length || y < 0 || y >= arr.length) {
		return;
	}
	if (visited[y][x] == true) {
		return;
	}
	visited[y][x] = true;
	if (arr[y][x] == value) {
		mArea++;
		findAreaHelper(arr, visited, value, x + 1, y);
		findAreaHelper(arr, visited, value, x, y + 1);
		findAreaHelper(arr, visited, value, x - 1, y);
		findAreaHelper(arr, visited, value, x, y - 1);
	}

}

static int mArea = 0;
public static void findArea(int[][] arr, int x, int y) {
	boolean[][] visited = new boolean[arr.length][arr[0].length];
	mArea = 0;
	findAreaHelper(arr, visited, arr[y][x], x, y);
}
```

RUNNING TIME COMPLEXITY:
$O(N^2)$

SPACE COMPLEXITY:
O(N)

17. Given an array of Triples that represent the the amount of RAM needed for an app and the start and termination times of the app, find the maximum amount of RAM needed.

e.g:

input:

start time	*end time*	*ram needed*
2	*4*	*1*
3	*6*	*2*
3	*9*	*3*

output: 6

____

SOLUTION:

```
class Triple {
	int startTime, endTime, ramNeeded;
}

public static int maxRamNeeded(List<Triple> in) {
	if (in == null) {
		return Integer.MIN_VALUE;
	}

	Map<Integer, Integer> tm = new TreeMap<Integer, Integer>();
	for (Triple t : in) {
		tm.put(t.startTime, t.ramNeeded);
		tm.put(t.endTime, -t.ramNeeded);
	}
	int curRam = 0, maxRam = 0;
	for (Map.Entry<Integer, Integer> entry : tm.entrySet()) {
		curRam += entry.getValue();
		maxRam = Math.max(curRam, maxRam);
	}

	return maxRam;
}
```

INFO: TreeMap always will keep its entries sorted based on their key. The cost for insertion is O(logN).

RUNNING TIME COMPLEXITY:

O(NlogN)

SPACE COMPLEXITY:

O(N)

18. Calculate the power of a number to a base without using a built-in function.
e.g:
input: 2^3
output: 8

SOLUTION:

```
public static long pow(int a, int b) {
        long[] M = new long[b + 1];
        M[0] = 1;
        M[1] = a;
        return powHelper(a, b, M);
}

public static long powHelper(int a, int b, long[] M) {
        if (M[b] == 0) {
                if (b % 2 == 0) {
                        M[b] = powHelper(a, b / 2, M)
                        * powHelper(a, b / 2, M);
                } else {
                        M[b] = powHelper(a, (b - 1) / 2, M)
                        * powHelper(a, (b - 1) / 2, M) * a;
                }
        }
        return M[b];
}
```

19. Given an array of integers, rearrange the array into a wave-like array (a1 >= a2 <= a3 >= a4 <= a5 >= ...).
e.g:
input: 1, 3, 2, 4, 5, 6, 7
output: 2, 1, 4, 3, 6, 5, 7

SOLUTION:

```
public static void waveSort(int[] arr) {
        if (arr == null) {
                return;
        }

        Arrays.sort(arr);
        for (int i = 0; i < arr.length - 2; i += 2) {
                int temp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
        }
}
```

INFO: Might not work if there are duplicate entries in the input list.

RUNNING TIME COMPLEXITY:
O(NlogN)

SPACE COMPLEXITY:
O(1)

20. Reverse the bits of an integer.
e.g:
input: 1011
output: 1101

SOLUTION:

```
public static int swapBits(int x, int i, int j) {
        int lo = ((x >> i) & 1);
        int hi = ((x >> j) & 1);
        if ((lo ^ hi) == 1) {
                x ^= ((1 << i) | (1 << j));
        }
        return x;
}

public static int revBits(int x, int numOfBits) {
        for (int i = 0; i < numOfBits; i++) {
                x = swapBits(x, i, numOfBits - i - 1);
        }
        return x;
}
```

INFO:
XOR (^) with 1 will toggle the value of the bit.
XOR (^) with two bits will return 1 if they are different.

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

21. Find the k-th largest integer in an unsorted array, in-place.
e.g:
input:
array: 5, 8, 3, 1
k: 2
output: 3

SOLUTION:

```
public static void swap(int[] arr, int x, int y) {
	int temp = arr[x];
	arr[x] = arr[y];
	arr[y] = temp;
}

public static int partition(int[] arr, int first, int last) {
	int pivot = first;
	swap(arr, last, pivot);
	for (int i = first; i < last; i++) {
		if (arr[i] > last) {
			swap(arr, i, first);
			first++;
		}
	}
	swap(arr, first, last);
	return first;
}

public static int quickselect(int[] arr, int first, int last, int k) {
	if (first > last)
		return Integer.MIN_VALUE;
	int pivot = partition(arr, first, last);
	if (pivot == k) {
		return arr[pivot];
	}
	if (pivot > k) {
		return quickselect(arr, first, pivot - 1, k);
	} else {
		return quickselect(arr, pivot + 1, last, k);
	}
}
```

INFO:
If you choose the pivot more wisely, such as choosing the median of medians of sets of 5, you can avoid the worst case time complexity of $O(N^2)$.

AVERAGE RUNNING TIME COMPLEXITY:
O(N)

WORST RUNNING TIME COMPLEXITY:
$O(N^2)$

SPACE COMPLEXITY:
O(1)

22. Given that a tree is a full binary tree (= every node other than the leaves has two children) and the pre-order and post-order traversal of the tree, re-construct it.

e.g:

input:

```
        pre-order: 5, 3, 7
        post-order: 3, 7, 5
```

output:

```
        tree:
                5
        3               7
```

____

SOLUTION:

```
static int mPreIndex = 0;
public static TNode constructTree(int[] pre, int[] post, int lo, int hi) {
        if ((lo > hi) || (mPreIndex >= pre.length)) {
                return null;
        }

        TNode root = new TNode(pre[mPreIndex]);
        mPreIndex++;
        if (lo == hi) {
                return root;
        }

        int i;
        for (i = lo; i < hi; i++) {
                if (pre[mPreIndex] == post[i]) {
                        break;
                }
        }
        root.left = constructTree(pre, post, lo, i);
        root.right = constructTree(pre, post, i + 1, hi - 1);

        return root;
}
```

23. Given the in-order and pre-order traversal of a tree, re-construct it.
e.g:
input:
    in-order: 3, 5, 7
    pre-order: 3, 7, 5

output:
    tree:

```
        5
    3       7
```

____

SOLUTION:

```
static int mPreIndex = 0;
public static TNode constructTree(int[] in, int[] pre, int lo, int hi) {
	if ((lo > hi) || (mPreIndex >= pre.length)) {
		return null;
	}

	TNode root = new TNode(pre[mPreIndex]);
	mPreIndex++;
	if (lo == hi) {
		return root;
	}

	int inIndex;
	for (inIndex = lo; inIndex <= hi; inIndex++) {
		if (in[inIndex] == root.data) {
			break;
		}
	}
	root.left = constructTree(in, pre, lo, inIndex - 1);
	root.right = constructTree(in, pre, inIndex + 1, hi);

	return root;
}
```

24. Given that a tree is a Binary Search Tree (BST) and the pre-order traversal of the tree, re-construct it.

e.g:

input:

pre-order: 5, 3, 7

output:

```
tree:
        5
3               7
```

SOLUTION:

```
public static TNode constructTree(int[] pre, int lo, int hi) {
        if (lo > hi) {
                return null;
        }

        TNode root = new TNode(pre[lo]);
        if (lo == hi) {
                return root;
        }

        int i;
        for (i = lo + 1; i <= hi; i++) {
                if (pre[i] > root.data) {
                        break;
                }
        }
        root.left = constructTree(pre, lo + 1, i - 1);
        root.right = constructTree(pre, i, hi);

        return root;
}
```

25. Delete a node from a Binary Search Tree (BST).

SOLUTION:

```
public static TNode deleteNodeFromBST(TNode root, int key) {
	if (root == null) {
		return null;
	}
	if (key > root.data) {
		root.right = deleteNodeFromBST(root.right, key);
		return root;
	}
	if (key < root.data) {
		root.left = deleteNodeFromBST(root.left, key);
		return root;
	}

	// 0 or 1 child
	if (root.left == null) {
		return root.right;
	} else if (root.right == null) {
		return root.left;
	}

	// 2 children
	TNode n = root.right;
	while (n.left != null) {
		n = n.left;
	}
	int inOrderSuccessorVal = n.data;
	root.data = inOrderSuccessorVal;
	root.right = deleteNodeFromBST(root.right, inOrderSuccessorVal);
	return root;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

26. Check if a given string contains balanced parenthesis.
e.g:
input: ( ) (( ))
output: true

input: ( ) ())
output: false
____

SOLUTION:

```
public static boolean isBalanced(char[] str) {
        int count = 0, i;
        for (i = 0; i < str.length; i++) {
                if (str[i] == '(') {
                        count++;
                } else {
                        count--;
                        if (count < 0) {
                                return false;
                        }
                }
        }
        if (count != 0) {
                return false;
        }
        return true;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

27. Check if a given string contains balanced '[ ]', '( )' or '{ }'.
e.g:
input: [({ } ([ ])]
output: true

SOLUTION:

```
public static boolean isBalanced(char[] str) {
        HashMap<Character, Character> hm;
        hm = new HashMap<Character, Character>();
        hm.put('(', ')');
        hm.put('[', ']');
        hm.put('{', '}');
        Stack<Character> s = new Stack<Character>();
        for (int i = 0; i < str.length; i++) {
                char curCh = str[i];
                if (hm.containsKey(curCh) == true) {
                        s.add(hm.get(curCh));
                } else {
                        if (s.isEmpty() == true) {
                                return false;
                        }
                        if (s.poll() != curCh) {
                                return false;
                        }
                }
        }
        return s.isEmpty();
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(N)

28. Find the position of a number in a 2D array of integers where its columns and rows are sorted.
e.g:
input:

```
	array:
	1 4 7
	2 5 8
	3 6 9

	num: 3
```

output: (2, 0)

SOLUTION:

```
class Point {
	int x, y;
	Point(int x, int y) {  this.x = x; this.y = y; }
}

public static Point findNum(int[][] arr, int num) {
	if (arr == null) {
		return null;
	}

	int y = 0;
	int x = arr[0].length - 1;
	while (y < arr.length && x >= 0) {
		if (num == arr[y][x]) {
			return new Point(x, y);
		} else if (num > arr[y][x]) {
			y++;
		} else {
			x--;
		}
	}
	return new Point(-1, -1);
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

29. Find the lowest common ancestor of two nodes in a binary tree which each node has a pointer to its parent.

e.g:
input:

```
        node 1: 1
        node 2: 7
        tree:
                5
        3               7
1
```

output: 5

SOLUTION:

```java
public static TNode findLCA(TNode n1, TNode n2, TNode root) {
        HashSet<TNode> set1 = new HashSet<TNode>();
        HashSet<TNode> set2 = new HashSet<TNode>();

        set1.add(n1);
        set2.add(n2);
        TNode cur1 = n1, cur2 = n2;
        while (true) {
                cur1 = cur1.parent;
                set1.add(cur1);
                cur2 = cur2.parent;
                if (set1.contains(cur1) == true) {
                        return cur2;
                }
                set2.add(cur2);
        }
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(N)

30. Find the lowest common ancestor of two nodes in a binary tree, in-place (no additional memory).
e.g:
input:

```
        node 1: 1
        node 2: 7
        tree:
                5
        3               7
1
```

output: 5

____

SOLUTION:

```java
public static TNode findLCA(TNode n1, TNode n2, TNode root) {
        if (root == null) {
                return null;
        }

        if ((root == n1) || (root == n2)) {
                return root;
        }

        TNode l = findLCA(n1, n2, root.left);
        TNode r = findLCA(n1, n2, root.right);

        if ((l != null) && (r != null)) {
                return root;
        }

        if (l != null) {
                return l;
        } else {
                return r;
        }
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

31. You are given an array of integers that represent the price of a certain stock for each day. If you are allowed only to buy the stock on a single day and sell on another day, find the maximum revenue you can achieve.

e.g:
input: 2, 1, 6, 5, 9
output: 8

SOLUTION:

```
public static int maxRevenue(int[] arr) {
        if (arr == null || arr.length == 0) {
                return Integer.MIN_VALUE;
        }

        int lowest = arr[0];
        int maxRev = 0;
        for (int i = 1; i < arr.length; i++) {
                maxRev = Math.max(maxRev, arr[i] - lowest);
                lowest = Math.min(lowest, arr[i]);
        }

        return maxRev;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

32. Remove a single occurrence of an integer from a doubly linked list.
e.g:
input:
list: 2 -> 5 -> 8 -> 9
val: 5
output: 2 -> 8 -> 9

____

SOLUTION:

```
class DLLNode {
	int data;
	DLLNode next, prev;
}

public static DLLNode removeNode(DDLNode head, int x) {
	if (head == null) {
		return null;
	}

	if (head.data == x) {
		head.next.previous = null;
		return head.next;
	}

	DDLNode runner = head;
	while (runner.next != null) {
		if (runner.next.data == x) {
			runner.next = runner.next.next;
			runner.next.previous = runner;
			break;
		}
		runner = runner.next;
	}
	return head;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

33. Given a doubly linked list reverse it in-place.
e.g:
input: 2 -> 5 -> 8 -> 9
output: 9 -> 8 -> 5 -> 2

SOLUTION:

```
class DLLNode {
        int data;
        DLLNode next, prev;
}

public static DLLNode revList(DLLNode head) {
        if (head == null) {
                return null;
        }

        while (head.next != null) {
                DLLNode temp = head.next;
                head.next = head.prev;
                head.prev = temp;
                head = temp;
        }
        head.next = head.prev;
        head.prev = null;
        return head;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

34. Given a sorted array of integers and a target integer determine if it is possible to sum up two integers to generate the target integer.

e.g:

input:

array: 1, 2, 5, 7, 8, 9

target: 7

output: true

SOLUTION:

```
public static boolean is2SumTarget(int[] arr, int target) {
        if (arr == null) {
                return false;
        }

        int st = 0;
        int end = arr.length - 1;
        while (st < end) {
                int sum = arr[st] + arr[end];
                if (sum == target) {
                        return true;
                } else if (sum > target) {
                        end--;
                } else {
                        st++;
                }
        }

        return false;
}
```

RUNNING TIME COMPLEXITY:

O(N)

SPACE COMPLEXITY:

O(1)

35. Given a sorted array of integers and a target integer determine if it is possible to sum up three integers to generate the target integer.

e.g:
input:
array: 1, 2, 5, 7, 8, 9
target: 17
output: true

SOLUTION:

```
public static boolean is3SumTarget(int[] arr, int target) {
        if (arr == null) {
                return false;
        }

        for (int i = 0; i < arr.length - 2; i++) {
                int st = i + 1;
                int end = arr.length - 1;
                while (st < end) {
                        int sum = arr[i] + arr[st] + arr[end];
                        if (sum == target) {
                                return true;
                        } else if (sum > target) {
                                end--;
                        } else {
                                st++;
                        }
                }
        }

        return false;
}
```

RUNNING TIME COMPLEXITY:
$O(N^2)$

SPACE COMPLEXITY:
O(1)

36. Given a Binary Tree determine if it is a Binary Search Tree (BST) or not.
e.g:
input:

```
        tree:
                5
        9               7
```

output: false

____

SOLUTION:

```
public static boolean isBST(TNode root, int min, int max) {
        if (root == null) {
                return true;
        }

        if (root.data > min && root.data < max) {
                return isBST(root.left, min, root.data)
                        && isBST(root.right, root.data, max);
        }

        return false;
}

.
.
.
isBST(root, Integer.MIN_VALUE, Integer.MAX_VALUE);
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

37. Given an array of integers that all integers appear twice except one that appears once, find that single integer.
e.g:
input: 2, 4, 2, 3, 5, 3, 4
output: 5

SOLUTION:

```
public static int findSingle(int[] arr) {
	if (arr == null) {
		return Integer.MIN_VALUE;
	}

	HashSet<Integer> set = new HashSet<Integer>();
	int sum = 0;
	for (int i = 0; i < arr.length; i++) {
		int cur = arr[i];
		if (set.contains(cur) == true) {
			sum -= cur;
		} else {
			set.add(cur);
			sum += cur;
		}
	}

	return sum;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(N)

38. Given a sorted array of integers find the integer with the greatest number of repetition.
e.g:
input: 1, 1, 1, 4, 4, 4, 4, 5, 6
output: 4

SOLUTION:

```
public static int findMostPopular(int[] arr) {
	if (arr == null || arr.length == 0) {
		return Integer.MIN_VALUE;
	}

	int element = arr[0];
	int count = 1;
	int maxElement = arr[0];
	int maxCount = 1;
	for (int i = 1; i < arr.length; i++) {
		if (arr[i] == element) {
			count++;
		} else {
			count = 1;
			element = arr[i];
		}

		if (count > maxCount) {
			maxCount = count;
			maxElement = element;
		}
	}

	return maxElement;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

39. Print the vertical level order of a binary tree.
e.g:
input:

```
        tree:
                5
        9               7
```

output:
9
5
7

____

SOLUTION:

```
public static void makeVerticalLevelOrder(TNode root, int hd, HashMap<Integer, List<Integer>>
hm, int[] minMax) {
        if (root == null) {
                return;
        }

        // save the minimum and maximum
        // horizontal distance (hd) of current node from root
        minMax[0] = Math.min(minMax[0], hd);
        minMax[1] = Math.max(minMax[1], hd);

        List<Integer> curHdLevel;
        if (hm.containsKey(hd) == true) {
                curHdLevel = hm.get(hd);
        } else {
                curHdLevel = new LinkedList<Integer>();
        }
        curHdLevel.add(root.data);
        hm.put(hd, curHdLevel);

        makeVerticalLevelOrder(root.left, hd - 1, hm, minMax);
        makeVerticalLevelOrder(root.right, hd + 1, hm, minMax);
}
```

```
.
.
.
// generate the hash map containing the vertical level order
HashMap<Integer, List<Integer>> hm = new HashMap<Integer, List<Integer>>();
int minMax[] = new int[2];
makeVerticalLevelOrder(root, 0, hm, minMax);

// print the list
for (int lvl = minMax[0]; lvl <= minMax[1]; lvl++) {
        List<Integer> curHdLevel = hm.get(lvl);
        for (Integer node : curHdLevel) {
                System.out.print(node + " ");
}
System.out.println();
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(N)

40. Print the level order of a binary tree (= print the nodes in each level).
e.g:
input:

```
	tree:
		5
	9		7
```

output:

```
5
9 7
```

____

SOLUTION:

```
public static void printLevelOrder(TNode root) {
	if (root == null) {
		return;
	}

	Queue<TNode> curLevel = new LinkedList<TNode>();
	Queue<TNode> nextLevel = new LinkedList<TNode>();
	curLevel.add(root);

	while (curLevel.isEmpty() == false) {
		while (curLevel.isEmpty() == false) {
			TNode curNode = curLevel.poll();
			System.out.print(curNode.data + " ");
			if (curNode.left != null) {
				nextLevel.add(curNode.left);
			}
			if (curNode.right != null) {
				nextLevel.add(curNode.right);
			}
		}
		System.out.println();

		curLevel = nextLevel;
		nextLevel = new LinkedList<TNode>();
	}
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(N)

41. Given a linked list reverse it in-place, recursively and iteratively.
e.g:
input: 3 -> 9 -> 6 -> 2
output: 2 -> 6 -> 9 -> 3

SOLUTION - RECURSION:

```
public static LLNode revList(LLNode head, LLNode prev) {
        if (head == null && prev == null) {
                return null;
        }

        LLNode next = head.next;
        head.next = prev;
        if (next == null) {
                return head;
        } else {
                return revList(next, head);
        }
}

.
.
.
head = revList(head, null);
```

SOLUTION - ITERATIVE:

```
public static LLNode revList(LLNode head) {
        if (head == null) {
                return null;
        }

        LLNode cur = head, prev = null;
        while (cur != null) {
                LLNode next = cur.next;
                cur.next = prev;
                prev = cur;
                cur = next;
        }

        return prev;
}

.
.
.
head = revList(head, null);
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

42. Implement a pattern matching function where:

1. *'.' matches any single character.*
2. *'*' matches 0 or more instances of the preceding character.*

e.g:
input:
    string: aabc
    *pattern: a*b.*
output: true

input:
    string: bc
    *pattern: a*b.*
output: true

____

SOLUTION:

```
public static boolean isMatch(char[] str, char[] pat, int s, int p) {
        if (s >= str.length) {
                return (p >= pat.length);
        }

        // the current char is a dot or a letter
        if (p == pat.length - 1 || pat[p + 1] != '*') {
                return (str[s] == pat[p] || pat[p] == '.')
                        && isMatch(str, pat, s + 1, p + 1);
        }

        // the current char is followed by a star, so either:
        // 1. ignore the star
        // 2. take the star in mind and check for equality
        return isMatch(str, pat, s, p + 2) ||
                (isMatch(str, pat, s + 1, p) && str[s] == pat[p]);

}
.
.
.
boolean result = isMatch(str, pat, 0 ,0);
```

43. Given an array that contains only 0, 1 or 2, sort it without using any sorting function.
e.g:
input: 0, 1, 1, 0, 2, 1
output: 0, 0, 1, 1, 1, 2

SOLUTION:

```
public static void swap(int[] arr, int x, int y) {
	int temp = arr[x];
	arr[x] = arr[y];
	arr[y] = temp;
}

public static void sort012(int[] arr) {
	if (arr == null) {
		return;
	}

	int st = 0, mid = 0, end = arr.length - 1;
	while (mid <= end) {
		switch (arr[mid]) {
		case 0:
			swap(arr, st, mid);
			st++;
			mid++;
			break;

		case 1:
			mid++;
			break;

		case 2:
			swap(arr, mid, end);
			end--;
			break;
		}
	}
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

44. Calculate the sum of two binary numbers that are given as arrays of characters. Return it as a string.
e.g:
input:
num 1: 001
num2: 101
output: 110

____

SOLUTION:

```
public static String addBinary(char[] num1, char[] num2) {
	int p1 = num1.length - 1;
	int p2 = num2.length - 1;
	int carry = 0;
	String res = "";

	while (p1 >= 0 || p2 >= 0 || carry != 0) {
		int curSum = carry;
		curSum += (p1 >= 0) ? (num1[p1] - ‘0’) : 0;
		curSum += (p2 >= 0) ? (num2[p2] - ‘0’) : 0;
		p1--;
		p2--;
		carry = (curSum >= 2) ? 1 : 0;
		int newDigit = curSum % 2;
		res = newDigit + res;
	}

	return res;
}
```

RUNNING TIME COMPLEXITY:
O(N)
N: max(num1.length, num2.length)

SPACE COMPLEXITY:
O(1)

45. Reverse an integer without using any built-in function.
e.g:
input: 123
output: 321

input: -123
output: -321
____

SOLUTION:

```
public static int revInt(int x) {
	boolean negative = false;
	if (x < 0) {
		x = x * -1;
		negative = true;
	}

	int res = 0;
	while (x > 0) {
		int LSB = x % 10;
		x = x / 10;
		res = res * 10 + LSB;
	}

	if (negative == true) {
		res = res * -1;
	}

	return res;
}
```

RUNNING TIME COMPLEXITY:
O(N)
N: number of digits of x

SPACE COMPLEXITY:
O(1)

46. Merge two sorted linked lists.
e.g:
input:
    list 1: 2 -> 4 -> 5
    list 2: 3 -> 7
output: 2 -> 3 -> 4 -> 5 -> 7

SOLUTION:

```
public static LLNode mergeLists(LLNode n1, LLNode n2) {
        if (n1 == null) {
                return n2;
        }
        if (n2 == null) {
                return n1;
        }

        LLNode res;
        if (n1.data <= n2.data) {
                res = n1;
                n1.next = mergeLists(n1.next, n2);
        } else {
                res = n2;
                n2.next = mergeLists(n1, n2.next);
        }

        return res;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

47. Given a list of words, print in each line the words that can be created using the same letters.
e.g:
input: abc, hello, cba
output:
abc cba
hello

SOLUTION:

```
public static void bucketAnagram(List<String> input) {
	HashMap<String, List<String>> hm;
	hm = new HashMap<String, List<String>>();
	for (String word : input) {
		char[] wordArr = word.toCharArray();
		Arrays.sort(wordArr);
		String key = new String(wordArr);

		List<String> list;
		if (hm.containsKey(key) == false) {
			list = new LinkedList<String>();
		} else {
			list = hm.get(key);
		}
		list.add(word);
		hm.put(key, list);
	}

	for (Map.Entry<String, List<String>> entry : hm.entrySet()) {
		for (String word : entry.getValue()) {
			System.out.print(word + " ");
		}
		System.out.println();
	}
}
```

RUNNING TIME COMPLEXITY:
O(N*MlogM)
N: number of input words
M: number of characters in the biggest input word

SPACE COMPLEXITY:
O(NM)

48. Given an array of integers of length K where the last element in the array is empty and all the other values (from 1 to k) are present in the array, find the missing number.
e.g:
input: 4, 5, 1, 3, _
output: 2

____

SOLUTION:

```
public static int missingNumber(int[] arr) {
        if (arr == null) {
                return Integer.MIN_VALUE;
        }

        int sumFrom1toK = arr.length;
        int sumOfArray = 0;
        for (int i = 0; i < arr.length - 1; i++) {
                sumOfArray += arr[i];
                sumFrom1toK += i + 1;
        }

        return sumFrom1toK - sumOfArray;
}
```

INFO: You can also use the formula for calculating the sum of an arithmetic progression: sumFrom1toK = (k*(1+k))/2. K is the length of the input array.

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(1)

49. Given a binary tree find the maximum sum that exist in a path of a tree. A path can start from any node and end at any node.

e.g:

input:

```
    tree:
        5
    3       1
-10
```

output: 9

____

SOLUTION:

```java
static int mMaxPathSum = Integer.MIN_VALUE;
public static int maxPathInSubtree(TNode root) {
	if (root == null) {
		return 0;
	}
	int leftSum = maxPathInSubtree(root.left);
	int rightSum = maxPathInSubtree(root.right);

	// update the maximum path cost,
	// take in mind the current node
	mMaxPathSum = Math.max(mMaxPathSum,
		leftSum + root.data + rightSum);

	// return the maximum path of the subtree below the current node
	return root.data + Math.max(0, Math.max(leftSum, rightSum));
}
.
.
.
mMaxPathSum = Integer.MIN_VALUE;
maxPathInSubtree(root);
System.out.println("Max path sum: " + mMaxPathSum);
```

INFO: The function needs to return two integers, that's why a global variable is used. If want to avoid this you can use a small array like question 14 to return the results.

RUNNING TIME COMPLEXITY:

O(N)

SPACE COMPLEXITY:

O(1)

50. Given an undirected graph where each node contains a label and a list of its neighbors, copy the whole data structure.

SOLUTION:

```
class Node {
	String label;
	ArrayList<Node> adjList = new ArrayList<Node>();

	Node(String label) {
		this.label = label;
	}
}

public static Node copyGraph(Node node) {
	if (node == null) {
		return null;
	}

	Queue<Node> queue = new LinkedList<Node>();
	HashMap<Node, Node> hm = new HashMap<Node, Node>();
	Node newNode = new Node(node.label);

	// the hash map holds the original nodes as keys
	// and the copied nodes as values
	hm.put(node, newNode);

	// the queue has the original nodes that
	// their adjacent nodes have not been copied yet
	queue.add(node);
```

```
        while (queue.isEmpty() == false) {
                Node curNode = queue.poll();
                for (Node adjNode : curNode.adjList) {
                        if (hm.containsKey(adjNode) == false) {
                                Node copyOfAdjNode = new Node(adjNode.label);
                                hm.put(adjNode, copyOfAdjNode);
                                hm.get(curNode).adjList.add(copyOfAdjNode);
                                queue.add(adjNode);
                        } else {
                                hm.get(curNode).adjList.add(hm.get(adjNode));
                        }
                }
        }

        return newNode;
}
```

RUNNING TIME COMPLEXITY:
O(N)

SPACE COMPLEXITY:
O(N)

51. *The following are some theoretical questions that might be asked in a coding interview. Note that bigger companies usually do not ask these kind of questions but many others do.*

- What is a Hash Map and how it works?
- What are the differences between HashMap and TreeMap in Java?
- If i want to use a HashMap with my custom objects as keys in Java, what functions do i need to override?
- What is virtual memory and how it works?
- What are the differences between a process and a thread?
- What are the virtual constructors and deconstructs in C/C++?
- What is pass by value and pass by reference?
- What are AVL and RBTrees? What are the insertion/deletion/look-up times for a self-balance tree and for a normal tree?
- Define what deadlock and starvation is in resource management.
- What does non-blocking thread-safe means?
- How to prevent a deadlock?
- How a mutex lock can be implemented?
- What are the difference between a Mutex and a Semaphore?
- How can a Garbage Collector detects which objects are not used anymore?
- What is the difference between preemptive and cooperative multitasking?
- What are some forms of Interprocess Communications (IPC)?
- What are the differences between Pipes/Message Queues/Shared Memory/Mapped Files in IPC?
- What are the differences between a struct and a class in C++?
- What are the differences between an Abstract class and an Interface in Java?
- In which cases you would use an Abstract and in which cases an Interface?
- What are the main differences between C++ and Java?
- What is Little Endian and Big Endian?
- What is the difference between Logical shift and an Arithmetic shift?
- What is a stack overflow?
- What is polymorphism?
- What are Generics in Java?
- What is Memory Fragmentation?
- What happens if an int value overflow in C++ and what happens in Java?
- What is Unit Testing?
- How would you make your solution to run in parallel?

- What software design patterns do you know?
- What are the systems development life cycles?
- What is software prototyping?
- What are some eviction policies for a cache?
- What is a distributed file system?

52. The following are some open-ended questions that might be asked in a coding interview as well. These questions might be system design questions or object-oriented design questions. There is no single answer for these questions and the interviewer would just want to see the way you approach the high level design of a solution to a big-scale problem.

- Design and explain the necessary classes and architecture for a parking management software.
- Design and explain the necessary classes and architecture for a software that you can draw different shapes.
- Design and explain the necessary classes and architecture for a poker game.
- Design and explain the necessary classes and architecture for an instant messaging application.
- Design and explain the necessary classes and architecture for a snake video game.
- Design an application that provides directions around a campus.
- Design a cache with LRU eviction policy and O(1) access time.
- Design an HTTP downloader that can cache its results.
- Design a service like PasteBin.
- Design a URL shortener service.
- Design a server architecture for dynamically serving map images.
- Design a system to return a unique ID for each request. The ID should increase as time increases.
- How to make sure that a program does not have memory leaks?
- Design a distributed system for storing key-value pairs.
- Design a system where the employees of a company will use access cards to keep track of their work time.
- In a system where millions of users are issuing requests at the same time (e.g. a search engine), design a way to give a prize to the requester of the billionth request.
- How would you design a fully automated system for build/test/deploy for an application.
- How would you design a text editor?
- How would you represent URLs in a storage medium in order to avoid duplication?