

Username: Pralay Patoria **Book:** Coding Interviews: Questions, Analysis & Solutions. No part of any chapter or book may be reproduced or transmitted in any form by any means without the prior written permission for reprints and excerpts from the publisher of the book or chapter. Redistribution or other use that violates the fair use privilege under U.S. copyright laws (see 17 USC107) or that otherwise violates these Terms of Service is strictly prohibited. Violators will be prosecuted to the full extent of U.S. Federal and Massachusetts laws.

Index



A

Anagrams, [212](#)
 Arrays.sort in $O(n \log n)$ time, [202](#)



B

Backtracking
 definition, [90](#)
 robot move
 C++ code, [93–4](#)
 source code, [94](#)
 test cases, [94](#)
 string path in matrix
 C++ code, [91–2](#)
 source code, [92](#)
 test cases, [92](#)
 Bidirectional communication, [220](#)
 Big numbers as strings
 function to add two arbitrary positive integers, [120–1](#), [123](#)
 simulating increment on a string, [118–9](#)
 simulating permutation, [120](#)
 source code, [120](#)
 test cases, [120](#)
 underestimating complexity, [117](#)
 Binary search algorithm
 on a diagonal, [38–40](#)
 Java code, [37](#)
 partially sorted matrix, [40–42](#)
 source code, [38](#)
 test cases, [38](#)
 Binary search trees and double-linked lists
 divide and conquer, [175–6](#)
 node rotations, [177–8](#)
 Binary trees
 binary search trees and double-linked lists
 divide and conquer, [175–6](#)
 node rotations, [177–8](#)
 mirror of, [143–146](#)
 paths, [165–168](#)
 print
 line printing, [161–2](#)
 printing process, [160](#)
 queue class, [160–1](#)
 three levels sample, [159–60](#)
 tree traversal algorithms, [159](#)
 zigzag printing, [162–165](#)
 traversal sequences
 deserialization, [171–2](#)
 disadvantages, [171](#)
 in-order traversal sequence, [168–9](#)
 post-order traversal sequence, [173–4](#)
 pre-order traversal sequence, [168–9](#)
 sample code, [169](#)
 serialization, [171–2](#)
 Binary tree traversals
 balanced binary trees, [225–7](#)
 depth of a binary tree, [224–5](#)
 k^{th} node in a binary search, [223–4](#)
 Bit operations
 bitwise AND, OR, and XOR, [101](#)
 find missing two numbers
 based on arithmetic calculation, [107–8](#)
 based on bit operations, [108](#)
 source code, [108](#)
 test cases, [108](#)
 modify a number to another, [105](#)
 number of 1s in binary
 checking rightmost bit with endless loop, [102](#)
 left-shift operation on 1, [102–3](#)

- minus one and then bitwise AND, [103–4](#)
- source code, [104](#)
- test cases, [104](#)
- numbers occurring only once
 - Java code, [106](#)
 - source code, [107](#)
 - test cases, [107](#)

Breadth-first traversal algorithm, [63](#)



C

- C, [13](#)
 - allocateMemory, [14–5](#)
 - macros, [15–6](#)
 - palindrome numbers, [16–7](#)
 - static variable, [13](#)
- C#, [22](#)
 - definition, [22](#)
 - singleton, [23, 25–27](#)
 - static constructor, [22](#)
- C++, [17](#)
 - assignment operator, [19–21](#)
 - class implementation/member function, [19](#)
 - execution of, [18–9](#)
 - key word sizeof, [18](#)
- Clone complex lists
 - boundary cases, [152](#)
 - first step cloning, [150](#)
 - five nodes, [149](#)
 - functional cases, [152](#)
 - m_pNext link, [149](#)
 - m_pSibling link, [149](#)
 - robustness cases, [152](#)
 - second step cloning, [150–1](#)
 - third step cloning, [151–2](#)
- combination(String str) method, [183](#)
- Communications skills, [219](#)
- ConstructCore function, [170](#)
- curSum, [198](#)



D

- Data structures
 - arrays
 - C code, [33](#)
 - definition, [33](#)
 - hash table, [33](#)
 - Java code to get duplicated number, [34–6](#)
 - sorted matrix search (*see* Binary search algorithm)
 - linked lists (*see* Linked lists)
 - stack and queue
 - definition, [70](#)
 - queue with two stacks, [70–2](#)
 - stack with two queues, [72–4](#)
 - string (*see* Strings)
 - trees (*see* Trees)
- Divergent thinking skills
 - array construction, [261–2](#)
 - $1+2+\dots+n$ calculation
 - based on constructors, [252–3](#)
 - based on function pointers, [254](#)
 - based on templates, [254–5](#)
 - based on virtual functions, [253–4](#)
 - final/sealed classes in C++, [259–60](#)
 - +, -, *, and / implementation
 - code to add, [255–6](#)
 - code to divide, [257–9](#)
 - code to multiply, [256–7](#)
 - code to subtract, [256](#)
- Divide and conquer approach
 - binary search trees (*see* Binary search trees and double-linked lists)
 - permutation and combination
 - bit operations, [183–5](#)
 - eight queens puzzle, [180–2](#)
 - n arrays, [182](#)
 - string, [179–80](#)
 - string combinations, [183](#)
 - sorted double-linked list, [168](#)
 - traversal sequences and binary trees (*see* Binary trees, traversal sequences)
- Double-linked lists, [168, 175](#)
 - divide and conquer, [175–6](#)
 - node rotations, [177–8](#)

Dynamic programming, [198](#)

Dynamic programming and greedy algorithms

definition, [95](#)

edit distance, [95–7](#)

minimal number of coins for change, [98–9](#)

minimal times of presses on keyboards, [99–100](#)

E

Equality of decimals, [116](#)

F

Fibonacci sequence

efficient $O(\log n)$ time solution, [78–9](#)

iterative solution with $O(n)$ time complexity, [77](#)

recursive and inefficient solution, [76–7](#)

source code, [79](#)

test cases, [80](#)

FirstAppearingOnce, [209](#)

G, H

HashMap, [212](#)

Hash table

anagrams, [212](#)

delete characters contained in another string, [210](#)

delete duplicated characters in string, [211](#)

index of character in stream, [208](#)

occurrence numbers in string, [207](#)

High-quality code

clearness, [111–2](#)

completeness

big numbers as strings (*see* Big numbers as strings)

delete nodes from a list, [123–7](#)

partition numbers in arrays (*see* Partition numbers in arrays)

power of integers (*see* Power of integer codes)

strategies to handle errors, [113–4](#)

test cases, [112–3](#)

robustness

k^{th} node from end, [132–5](#)

reverse a list, [135–8](#)

substructures in trees, [138–41](#)

I

Increment method, [184](#)

In-order traversal algorithm, [63](#)

Interview cases

integer value from a string

code for member initialization order, [263](#)

code to convert a string to an integer, [264–6](#)

interviewer's comments, [267](#), [269](#)

source code, [269](#)

test cases, [269](#)

lowest common parent node in a tree

code to get the lowest ancestor, [272–3](#)

interviewer's comments, [273](#)

source code, [274](#)

test cases, [274](#)

tree diagram, [270](#)

tree with no links to parents, [271](#)

Interview process, [1](#)

behavior, [4](#)

avoid complaints, [6](#)

project experience, [4–5](#)

technical skills, [5](#)

on-site interview, [3](#)

phone-interview, [1–3](#)

Q/A time, [11](#)

technical, [7](#)

high quality code, [8–9](#)

problem solving, [10](#)

programming knowledge, [7](#)

soft skills, [11](#)

time and space efficiency, [10–1](#)

Interviews skills

communications skills, [219](#)

divergent thinking skills (*see* Divergent thinking skills)

knowledge migration skills (*see* Knowledge migration skills)

learning skills, [220](#)

mathematical modeling skills (*see* Mathematical modeling skills)

J

Java, 27

- data containers, 29–30
- final variables, 28–9
- keyword final, 28
- thread scheduler, 30, 32

Josephus problem, 243

K

Knowledge migration skills

- binary tree traversals (*see* Binary tree traversals)
- maximums in queue, 239–41
- maximums in sliding window, 236–9
 - reversing words and rotating strings
 - reversing a segment, 233–5
 - string left rotation, 235–6
- sorted array
 - boundary test cases, 223
 - code to count k , 222
 - code to get first k , 221
 - code to get last k , 222
 - functional test cases, 223
 - source code, 223
- sum in sequences
 - finding continuous sequences with sum s , 231–3
 - finding number pairs, sum equals s , 227–9
 - getting a pair with a sum excluding a number, 229–30
 - getting a subset with sum 0, 230–1

L

Learning skills, 220

Linked lists

- loop in list, 59–62
- memory allocation, 53
- printing lists from tail to head, 54–5
- sort lists, 56–9

M

Master theory, 250

Mathematical modeling skills

- last number in a circle
 - deleted numbers pattern, 244–6
 - looped list circle simulation, 243–4
- minimum number of moves to sort cards
 - binary search costing $O(n \log n)$ time, 247–9
 - dynamic programming costing $O(n^2)$ time, 246–7
- most profit from stock
 - divide and conquer based, 249–50
 - storing minimum numbers while scanning, 251
- probabilities of dice points, 241–3

 $\max[f(i)]$, 198

maxQueue, 194

Median in a stream

- binary search algorithm, 188
- C++ code, 190
- numbers sorting, 188–9
- time efficiency comparisons, 189

Minimum k numbers

- comparison between two solutions, 194
- $O(n)$ time efficiency, 193
- $O(n \log k)$ time efficiency, 191

N

Nonverbal communication, 219

NumericComparator, 202

O $\text{occurrence}[i]$, 209

Optimization

- space-time trade-off
 - first intersection node in two lists, 216–8
 - hash tables for characters, 207–13
 - reversed pairs in array, 213–6
 - ugly numbers, 204–6
- time efficiency
 - data structures and algorithms, 187
 - Digit 1 occurrence, 198–201

greatest sum of sub-arrays, 196–8
 median in stream, 188–91
 minimum k numbers, 191–4
 sorted arrays intersection, 194–6
 StringBuilder.Append, 187

$O(m+n)$ time, 195
 $O(n\log m)$ time, 195
 $O(n)$ time efficiency, 193
 $O(n\log k)$ time efficiency, 191



P

Partition numbers in arrays
 move numbers for $O(k)$ times, 131–2
 move numbers for $O(n)$ times, 130
 scalable solution, 128, 130
 workable but not scalable solution, 127–8

Performance optimization. *See* Optimization

pLastNodeInList, 176

pop_heap, 190

Post-order traversal algorithm, 63

Power of integer codes
 complete and efficient solution, 116–7
 complete but inefficient solution, 115–6
 incomplete solutions, 114–5
 source code, 117
 test cases, 117

Pre-order traversal algorithm, 63, 144–5

Print binary trees
 line printing, 161–2
 printing process, 160
 queue class, 160–1
 three levels sample, 159–60
 tree traversal algorithms, 159
 zigzag printing, 162–5

Print Matrix, spiral order
 java code, 147
 printRing method, 147
 ring printing code, 148–9
 set of rings, 147

Problem solutions
 divide and conquer approach (*see* Divide and conquer approach)
 examples
 binary tree paths, 165–8
 print binary trees (*see* Print binary trees)
 push and pop sequence of stacks, 157–9
 stack with min function (*see* Stack with min function)
 figures
 clone complex lists (*see* Clone complex lists)
 mirror of binary trees, 143–6
 print matrix, spiral order (*see* Print matrix, spiral order)

Programming languages

C, 13
 allocateMemory, 14–5
 macros, 15–6
 palindrome numbers, 16–7
 static variable, 13
 C#, 22
 definition, 22
 singleton, 23–7
 static constructor, 22
 C++, 17
 assignment operator, 19–21
 class implementation/member function, 19
 execution of, 18–9
 key word sizeof, 18
 Java, 27
 data containers, 29–30
 final variables, 28–9
 key word final, 28
 thread scheduler, 30, 32

push_heap, 190



Q, R

ReadStream function, 172

Recursion and iteration
 disadvantages, 76
 Fibonacci sequence
 efficient $O(\log n)$ time solution, 78–9
 iterative solution with $O(n)$ time complexity, 77

- recursive and inefficient solution, [76–7](#)
- source code, [79](#)
- test cases, [80](#)
- iterative C code, [75](#)
- recursive C code, [75](#)

Reversed pair in array

- Java code to count, [215–6](#)
- merge sub-arrays, [214](#)
- process to get the number of, [214](#)

S

Search and sort algorithms

- binary search
 - code for turning number in array, [87](#)
 - Java code to get minimum element, [85–6](#)
 - minimal element search, [84](#)
 - source code, [86](#)
 - test cases, [86](#)
 - two rotations of sorted array, [85](#)
- Java code
 - of count sort, [83](#)
 - to partition an array, [81–2](#)
 - for quicksort, [82](#)
- majorities in arrays
 - definition of majority, [89](#)
 - partition method, [88–9](#)
 - source code, [90](#)
 - test cases, [90](#)

Stack and queue

- definition, [70](#)
- queue with two stacks, [70–2](#)
- stack with two queues, [72–4](#)

Stack with min function

- with auxiliary stack, [153–5](#)
- without auxiliary stack, [155–6](#)

Strings

- in C#, [43–4](#)
- in C/C++, [42](#)
- in Java, [44–5](#)
- replacing blanks
 - from left to right in $O(n^2)$ time, [45–6](#)
 - merging sorted arrays, [48–9](#)
 - from right to left in $O(n)$ time, [46, 48](#)
 - source code, [48](#)
 - test cases, [48](#)
- string matching
 - code to scan digits, [52](#)
 - code to verify an exponential notation, [53](#)
 - code to verify numeric strings, [52](#)
 - simple regular expression matching, [50–1](#)

Symmetrical trees, [145–6](#)

T

toBePrinted variable, [162](#)

Trees

- binary search tree verification
 - increasing in-order traversal sequence, [67–8](#)
 - value range of each node, [66–7](#)
- code to get the largest size of subtrees, [68–70](#)
- next nodes in binary trees, [64–6](#)
- sample binary tree, [63–4](#)
- traversal algorithms, [63](#)

U

Ugly number

- check, [204](#)
- store found numbers into array, [205](#)

V, W, X, Y, Z

Verbal communication, [219](#)