Data structures and algorithms

How to crack interviews at big companies?!

Hi, I’m Vusal Dadalov

Senior Software Engineer at Uber (Amsterdam)

Hi, I’m Vusal Dadalov

Senior Software Engineer at Uber (Amsterdam)

Past

- Careem - Ride sharing (Berlin)
- OLX - Classifieds (Berlin)
- Azercell Telecom (Baku)

How to crack an interview at big companies?!

- DS & Algorithms
- System Design - OS internals, Networking, Distributed Systems etc.
- Behavioural skills - leadership, communication, etc.

DS & Algorithms

Why study Algorithms?

- To become a proficient programmer
- Understand time and memory complexities and be able to make smart tradeoffs
- Supercomputer won't help much; good algorithm enables solution.

Why study Algorithms?

“ I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”

— Linus Torvalds (creator of Linux)

Why study Algorithms?

For fun :)

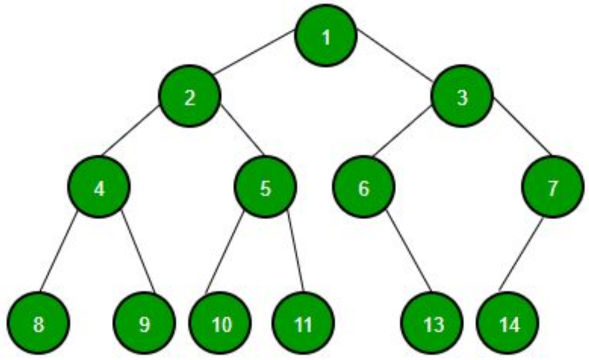
Why study Algorithms?

and profit...

KASPERSKY lab
lenovo
Aol.
AMD
IBM
ORACLE
Adobe
facebook.
Microsoft
FUJITSU
Sun microsystems
Java
DELL
Google
hp
NVIDIA
YAHOO!
intel
Bai du 百度
SAP
WD Western Digital
amazon web services
McAfee
TOSHIBA Leading Innovation
xerox
cisco

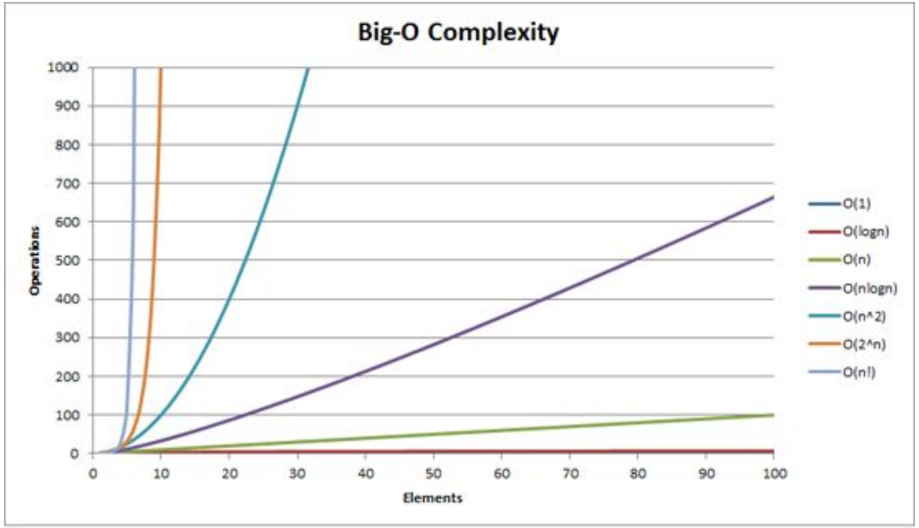
Main data structures

- Lists - ArrayList, LinkedList
- Stack & Queue
- Sets
- Dictionary
- Tree



Complexity - Big O

- Time complexity
- Memory complexity



Complexity - Big O

O(N)

```
for (int i = 1; i <= n; i++) {
    // some O(1) expressions
}
```

O(N^2)

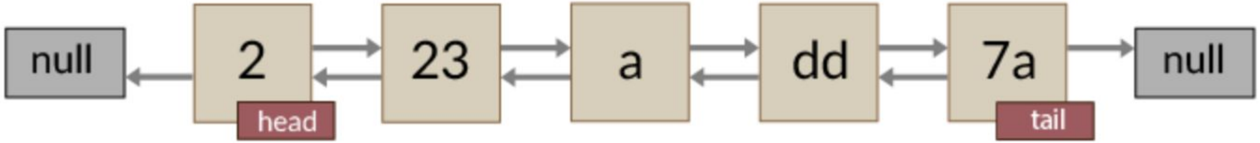
```
for (int i = 1; i <=n; i += c) {
   for (int j = 1; j <=n; j += c) {
      // some O(1) expressions
   }
}
```

O(LogN)

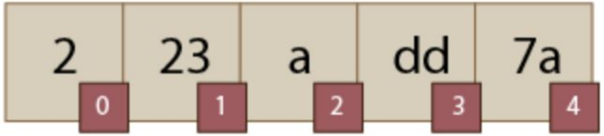
```
for (int i = 1; i <=n; i *= c) {
    // some O(1) expressions
}

for (int i = n; i > 0; i /= c) {
   // some O(1) expressions
}
```

Lists

LinkedList



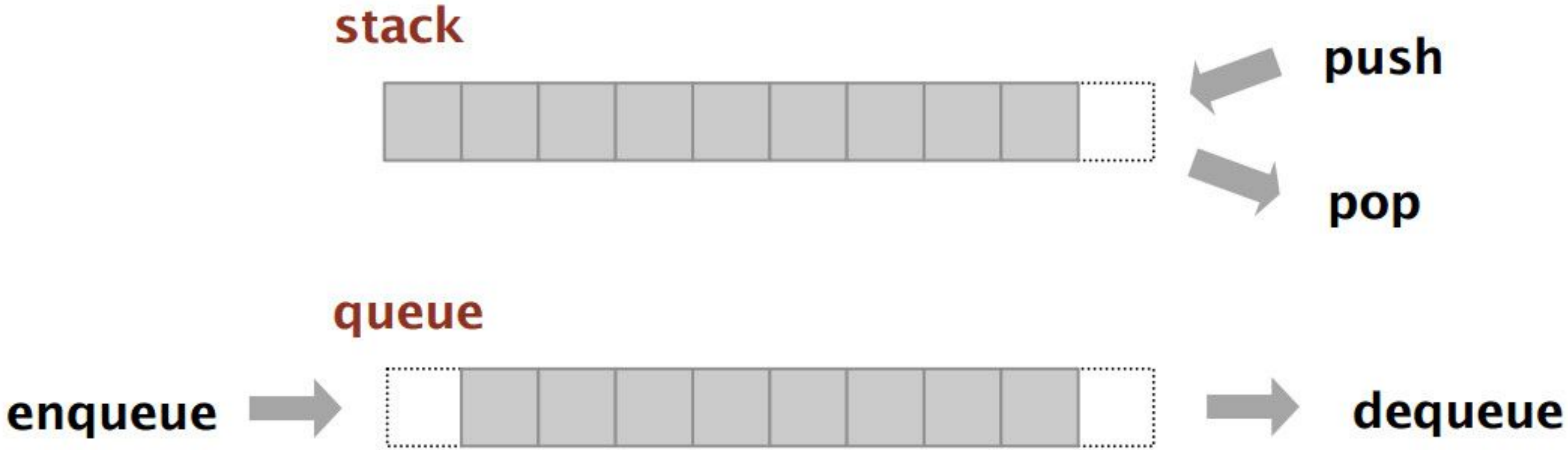
Array and ArrayList



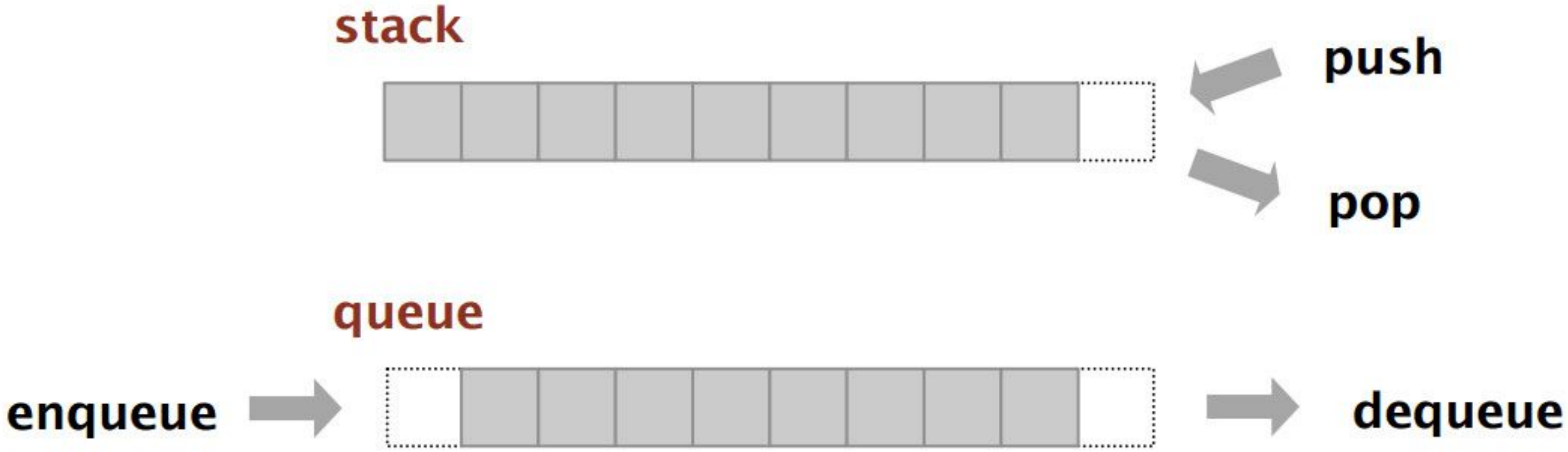
Lists

Operation	ArrayList	LinkedList
get()	O(1)	O(N)
add()	O(1) amortized	O(1)
remove()	O(N)	O(1)

Stack & Queue

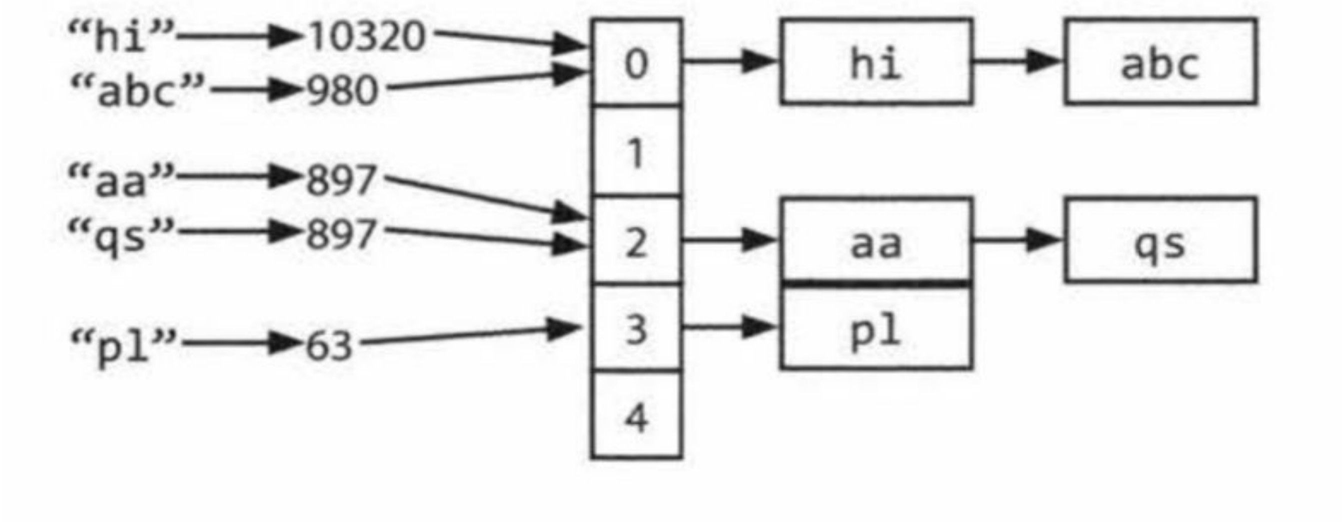


Stack & Queue



Dictionary

HashMap & HashSet in Java

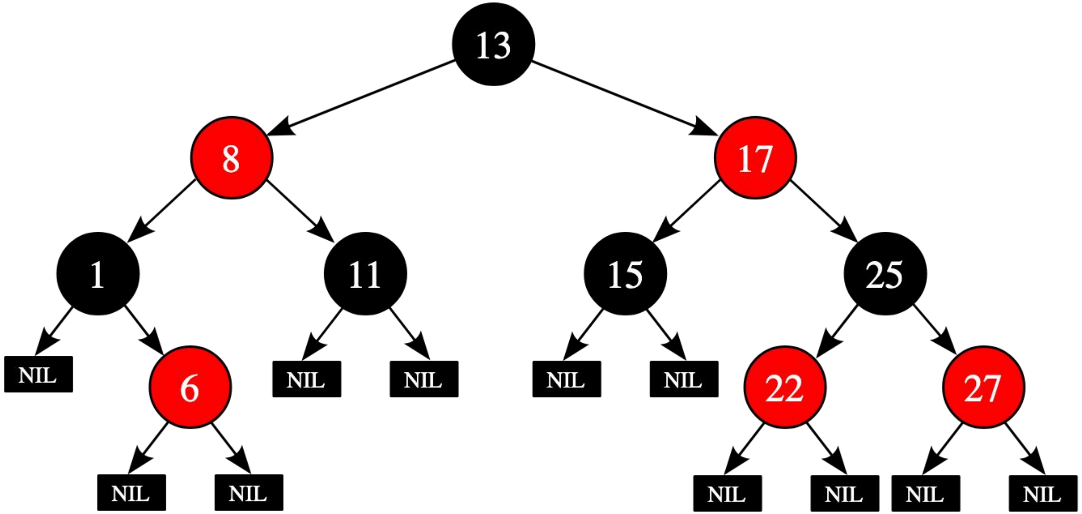


HashSet

HashSet internally uses HashMap (Java)

TreeMap and TreeSet

Internally uses Red-black tree



HashMap & TreeMap

Operation	HashMap (Java)	TreeMap (Java)
get()	O(1)	O(Log N)
put()	O(1)	O(Log N)
remove()	O(1)	O(Log N)

Always expect to get questions about these main data structures

Practical example

Let's do something fun :)

Given an array of integers, write a method to return the k most frequent elements.

For example:

arr = [15, 22, 3, 1, 10, 2, 8, 9, 15, 5, 7, 31, 2, 3, 2]   k=3

Answer: 2, 3, 15

- 2: 3
- 3: 2
- 15: 2

Create a frequency map and sort it

```python
from collections import defaultdict

def top_k(arr, k):
    frequency_map = defaultdict(int)
    # O(N)
    for val in arr:
        frequency_map[val] += 1

    # O(NlogN)
    sorted_frequency_map = sorted(frequency_map.items(),
        key=lambda kv: kv[1], reverse=True)

    # O(k)
    return sorted_frequency_map[:k]
```

Create a frequency map and sort it

```
Total cost is: O(NlogN)


For 10^9 items it is roughly 10^9 x log(10^9) => 40 x 10^9
```

Can we do better?

Use Priority Queue

```
from collections import defaultdict
import heapq

def top_k(arr, k):
    frequency_map = defaultdict(int)
    # O(N)
    for val in arr:
        frequency_map[val] += 1

    # O(NlogK)
    pq = []
    for item,frequency in frequency_map.items():
        heapq.heappush(pq, (frequency, item))

        if len(pq) > k:
            heapq.heappop(pq)
    # O(K)
    return [(v,k) for k,v in pq][::-1]
```

Use Priority Queue

```
Total cost is: O(NlogK)

let's say K is 100

For 10^9 items it is roughly 10^9 x log(100) => 7 x 10^9
```

Use Priority Queue

Woow! That is pretty good improvement!

Can we do better?

Maybe? :)

Other data structures worth to know

- Priority queue (Heap)
- Trie, Segment Tree, B-Tree
- Graph
- etc.

Some Techniques

- Recursion
- Dynamic programming
- BFS - Breadth First Search
- DFS - Depth First Search

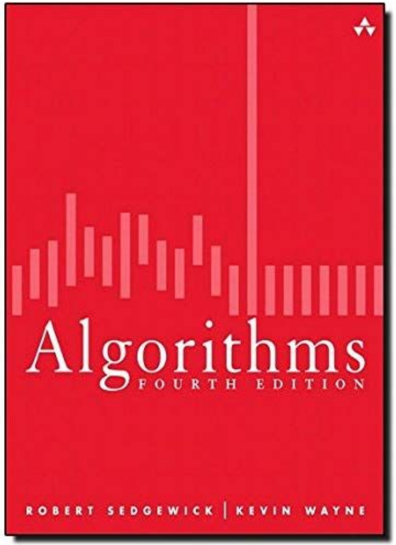
Some Techniques

Unfortunately, we don’t have enough time to explain all these in one session :(

How to prepare for data structures and algorithms

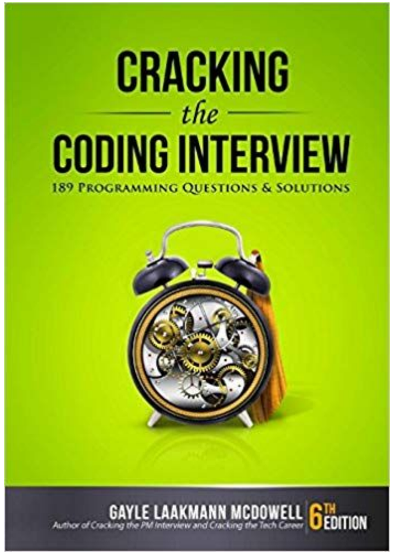
Algorithms by Robert Sedgewick

- Coursera - Algorithms part 1
- Coursera - Algorithms part 2

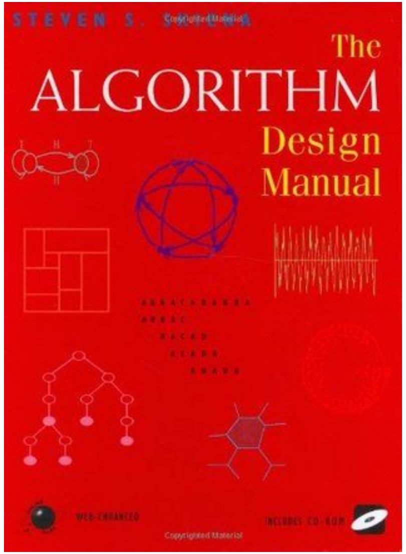


Cracking the Coding Interview

Book by Gayle Laakmann McDowell


CRACKING the CODING INTERVIEW
189 PROGRAMMING QUESTIONS & SOLUTIONS
GAYLE LAAKMANN MCDOWELL
Author of Cracking the PM Interview and Cracking the Tech Career
6TH EDITION

Algorithm design manual



Places to practise

- https://leetcode.com/
- https://codesignal.com/
- https://www.hackerrank.com/

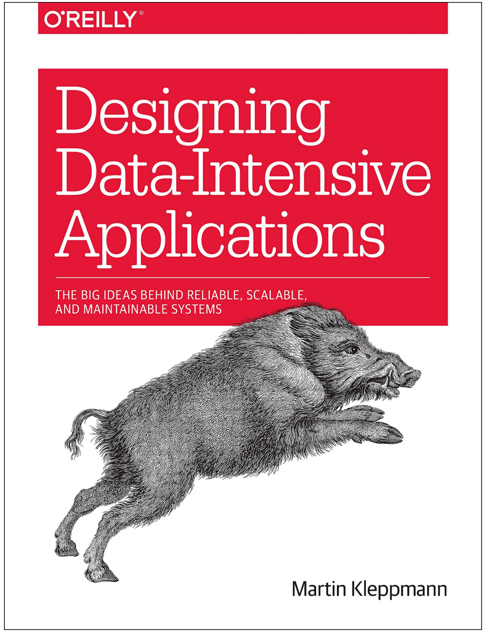
More resources

https://www.geeksforgeeks.org/
https://www.programcreek.com/2012/11/top-10-algorithms-for-coding-interview/

A few words about system design interviews

Here, mostly talking...

Design data intensive applications


O'REILLY
Designing Data-Intensive Applications
THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS
Martin Kleppmann

More links

- Grokking the System Design Interview - paid
- https://github.com/donnemartin/system-design-primer
- http://highscalability.com/

And, read big engineering blogs of big tech companies and understand how did they build their systems

Questions?