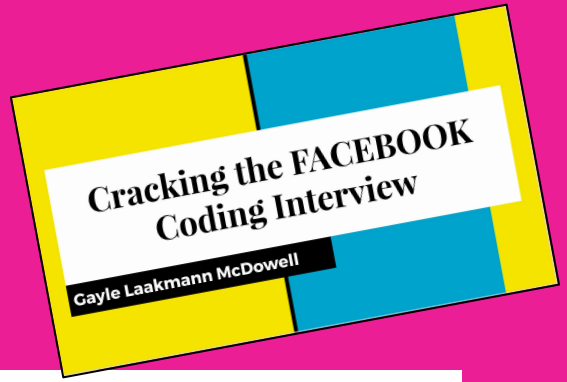
While you wait... find Big O of these:

Yes, this is hard to read. Go here:

bit.ly/fbprepbigo

(can be easily viewed on a phone too)



#1
```
min = ...
for (int v : array) {
    if (min > v) {
        min = v
    }
}
```

#2
```
min = ...
max = ...
for (int v : array) {
    if (min > v) {
        min = v
    }
}
for (int v : array) {
    if (max < v) {
        max = v
    }
}
```

#3
```
for (int a : A) {
    for (int b : B) {
        if (a * b < a + b) {
            print(a * b)
        }
    }
}
```

#4
```
for i from 0 to N
    for j from i to N {
        if i * j < K {
            print(i * j);
        }
    }
}
```

#5
```
int last_death = Integer.Min

/* step 1: get last death */
for (Person person : people) {
    last_death = max(last_death, person.death)
}

/* step 2: increment counter for each year someone is alive */
int[] counter = new int[last_death]
for (Person person : people) {
    for (int year = person.birth; year < person.death; year++) {
        counter[year]++;
    }
}

/* step 3: find population peak */
int highest_population = 0
for (int year = 0; year < counter.length; year++) {
    highest_population = max(highest_population, counter[year])
}
```

Q5: Given list of people with birth years and death years (so population changes over time), find highest population.

1. Get last death year.
2. Create histogram from 0 → last death year (this will end up giving us population at each year).
   a. Walks through all people
      i. For each person, increment histogram for each year they were alive.
3. Walk through histogram to get highest population.

#6
```
for i from 0 to A.length:
    if validate(A[i])
        print(i)
```

#7
```
int fib(int n) {
    if (n < 0) {
        return 0;
    } else if (n <= 1) {
        return 1;
    } else {
        return fib(n - 1) + fib(n - 2);
    }
}
```

We'll talk about the answers later. But seriously, use this link:

bit.ly/fbprepbigo

Cracking the FACEBOOK Coding Interview

Gayle Laakmann McDowell

gayle@gayle.com
subject: **fbprep**

why am I here

facebook wants you
to do your best

Hi! I'm Gayle Laakmann McDowell

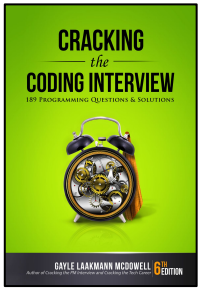
(CS) Penn University of Pennsylvania

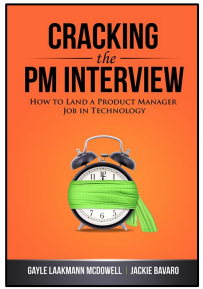
Wharton University of Pennsylvania (MBA)

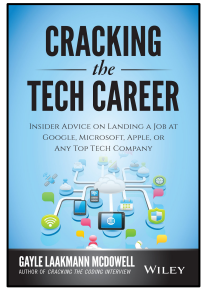
<dev> Microsoft Google </dev>

careercup.com

Author







Acquisition Interview Coaching

Hiring Consultant / Interview Training

Yes! Slides are online!

- Gayle.com > Events > Facebook

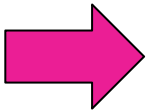
gayle@gayle.com
subject: **fbprep**

gayle@gayle.com
subject: **fbprep**

what to expect

the *typical* process (not universal!)

PHONE INTERVIEWS



ONSITE INTERVIEWS

behavioral + algorithms/coding

algorithms/coding

algorithms/coding

behavioral + algorithms/coding

design

coding interview

(typical)

5m	Behavioral
35m	Coding Question #1
	Coding Question #2
5m	Q&A

design interview

(typical)

5m	Behavioral
35m	Design Question #1
5m	Q&A

behavioral questions

gayle@gayle.com
subject: **fbprep**

So, walk me through your resume...

- “I’m a <title> at <company>
- I studied at …
- And then I …
- At my current company, I …
- On the side, I… <hobbies>”

Not too long!

Shows of success

Prompt the interviewer

Hobbies

Past Projects

- 3+ projects
  - Hard / cool
  - You were central

- TECHNICAL
  Challenges, architecture, technologies, etc

- SOFT
  Leadership, conflicts, decisions, etc

what did *you* do?
(not your team)

what are you trying to say?
(the message)

gayle@gayle.com
subject: **fbprep**

design questions

how to tackle

(it's not that scary!)

W

W

Y

D

A

W

how to tackle

(it's not that scary!)

What

Would

You

Do

At

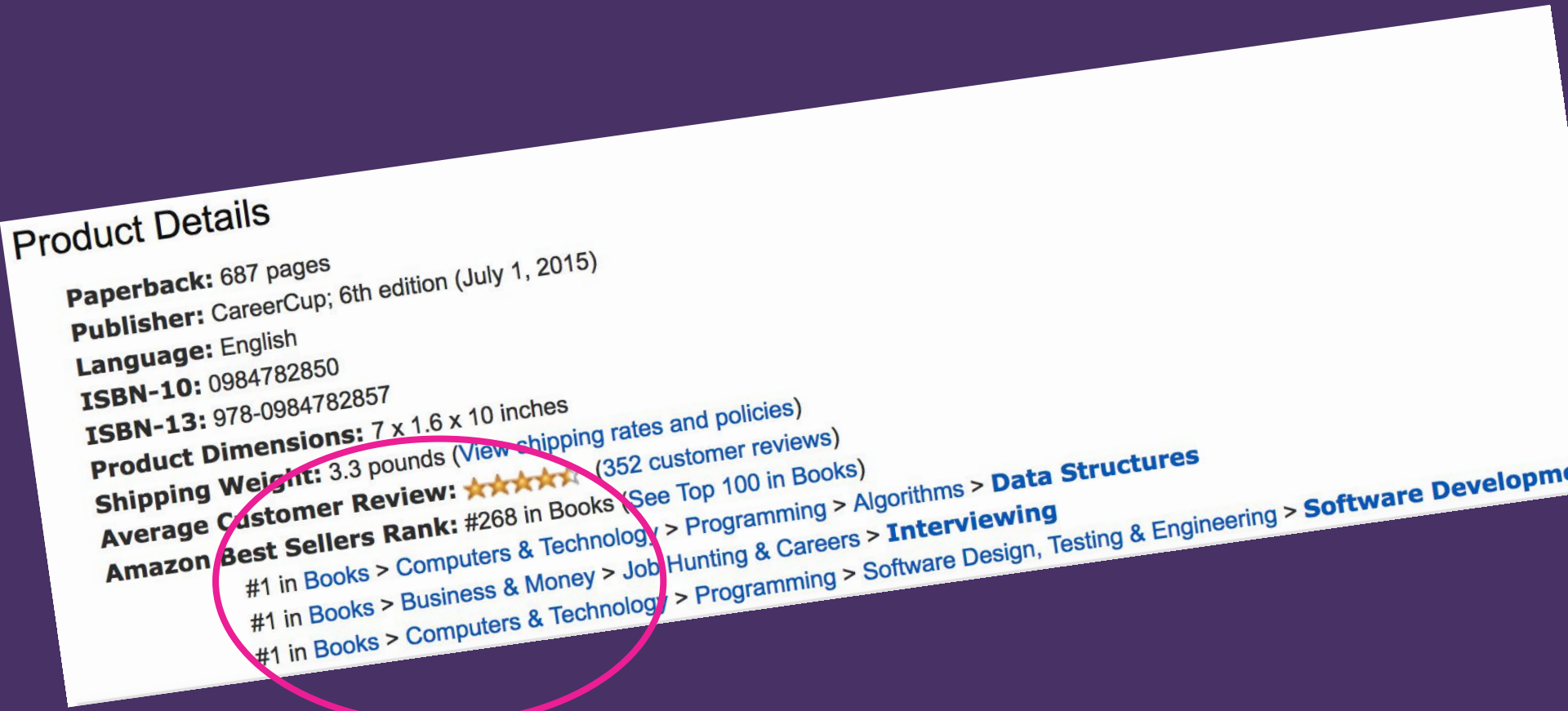
Work

the process - SKIR

- **S**COPE
  ask questions & make assumptions
- **K**EY COMPONENTS
  use the whiteboard!
- **I**DENTIFY ISSUES
  Bottlenecks, tradeoffs, ...
- **R**EPAIR

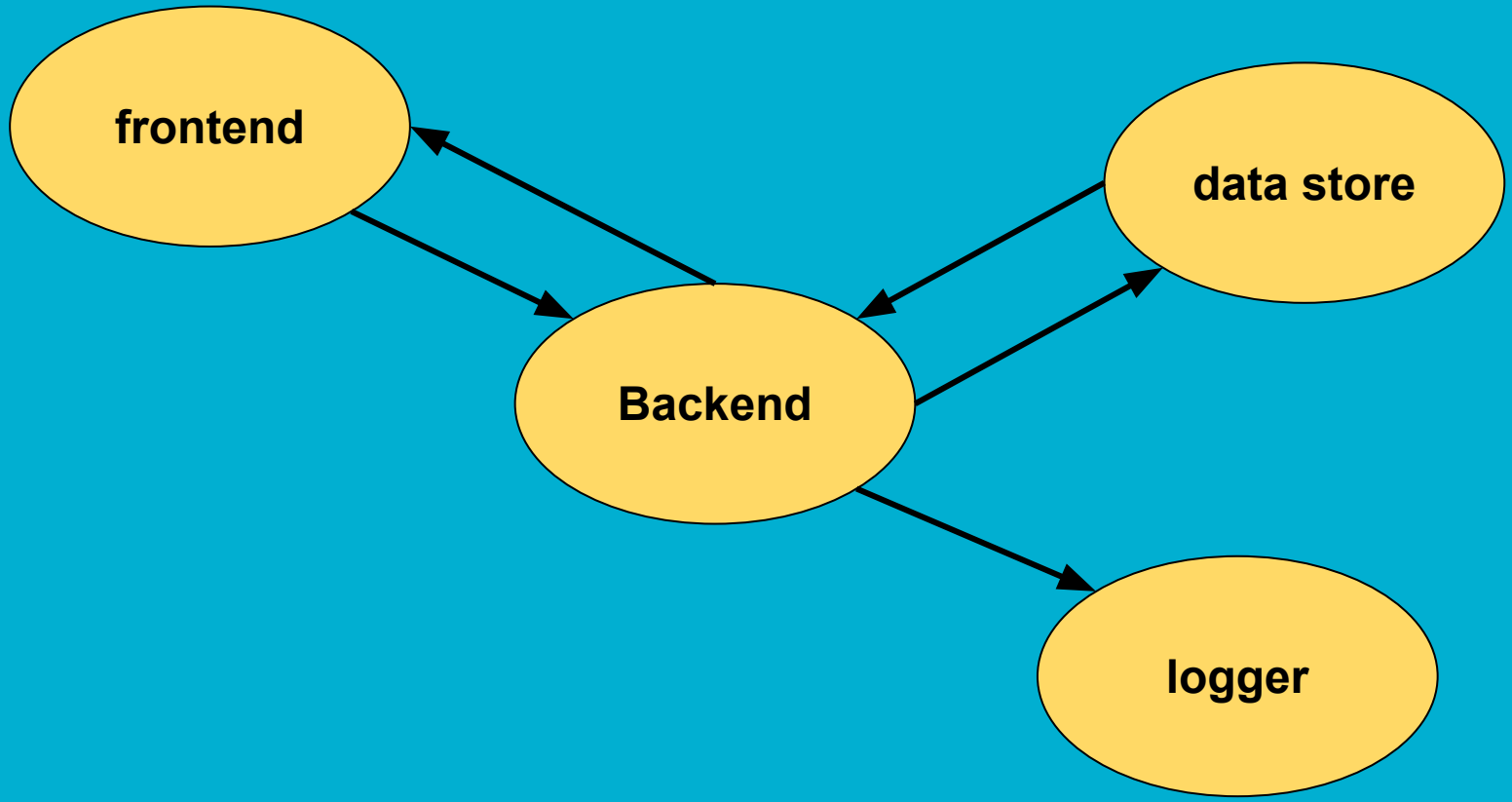
breadth-first

not depth-first



Product Details

Paperback: 687 pages
Publisher: CareerCup; 6th edition (July 1, 2015)
Language: English
ISBN-10: 0984782850
ISBN-13: 978-0984782857
Product Dimensions: 7 x 1.6 x 10 inches
Shipping Weight: 3.3 pounds (View shipping rates and policies)
Average Customer Review: ★★★★½ (352 customer reviews)
Amazon Best Sellers Rank: #268 in Books (See Top 100 in Books)
#1 in Books > Computers & Technology > Programming > Algorithms > **Data Structures**
#1 in Books > Business & Money > Job Hunting & Careers > **Interviewing**
#1 in Books > Computers & Technology > Programming > Software Design, Testing & Engineering > **Software Developme**


frontend
data store
Backend
logger

a collaborative discussion
(but one that you are driving)

LEADER & TEAMMATE

How to prepare

- Read about architecture
- Use friends at startups
- Know key concepts
- Practice back-of-the-envelope calculations

gayle@gayle.com
subject: **fbprep**

algorithm questions

What the interviewer is looking for

- Analytical skills / problem-solving skills
- How you think
- Ability to make tradeoffs
- Communication
- Willingness to push through hard problems
- Strong CS fundamentals
  - CS degree not required

CS degree is not required
(or even that important)

Essential CS Knowledge

Often not covered in algorithms class!

DATA STRUCTURES	ALGORITHMS	CONCEPTS
ArrayLists	Merge Sort	Big-O Time
Hash Tables	Quick Sort	Big-O Space
Trees, Tries & Graphs	Breadth-First Search	Recursion
Linked Lists	Depth-First Search	Memoization & Dynamic Programming
Stacks / Queues	Binary Search	
Heaps		

The concept/process, not specific famous problems

How to Prepare

1. **MASTER Big O**
2. Implement core data structures & algorithms
3. Practice on real interview questions
4. Code by hand (paper, whiteboard, window/dry-erase)
5. Mock interviews

gayle@gayle.com
subject: **fbprep**

big O crash course

01. MIN

```
1 min = ...
2 for (int v : array) {
3     if (min > v) {
4         min = v
5     }
6 }
```

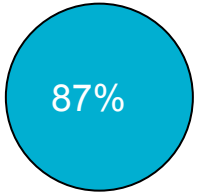
→ O(N)

02. MIN AND MAX

```
1  min = ...
2  max = ...
3  for (int v : array) {
4      if (min > v) {
5          min = v
6      }
7  }
8  for (int v : array) {
9      if (max < v) {
10         max = v
11     }
12 }
```

→ O(N)

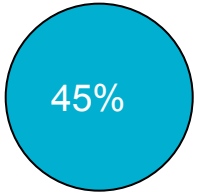
DROP CONSTANTS



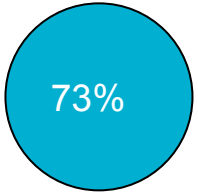
03. NESTED WITH TWO ARRAYS

```
1 for (int a : A) {
2     for (int b : B) {
3         if (a * b < a + b) {
4             print(a * b)
5         }
6     }
7 }
```

→ O(A * B)



04. NESTED WITH J STARTING AT I




```
1 for i from 0 to N
2     for j from i to N {
3         if i * j < K {
4             print(i * j);
5         }
6     }
7 }
```

$\longrightarrow O(N^2)$

	j : 0 → N					
i : 0 → N	x	x	x	x	x	x
		x	x	x	x	x
			x	x	x	x
				x	x	x
					x	x
						x

05. MAX POPULATION

5%

```
1  int last_death = Integer.Min
2
3  /* step 1: get last death */
4  for (Person person : people) {
5      last_death = max(last_death, person.death)
6  }
7
8  /* step 2: increment counter for each year someone is alive */
9  int[] counter = new int[last_death]
10 for (Person person : people) {
11     for (int year = person.birth; year < person.death; year++) {
12         counter[year]++;
13     }
14 }
15
16 /* step 3: find population peak */
17 int highest_population = 0
18 for (int year = 0; year < counter.length; year++) {
19     highest_population = max(highest_population, counter[year])
20 }
```

Step 1: O(P)

P = # people

Step 2: O(P*Y)

Y = max life span

Step 3: O(L)

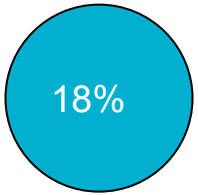
L = last death year

→ O(P + P*Y + L)

→ O(P*Y + L)

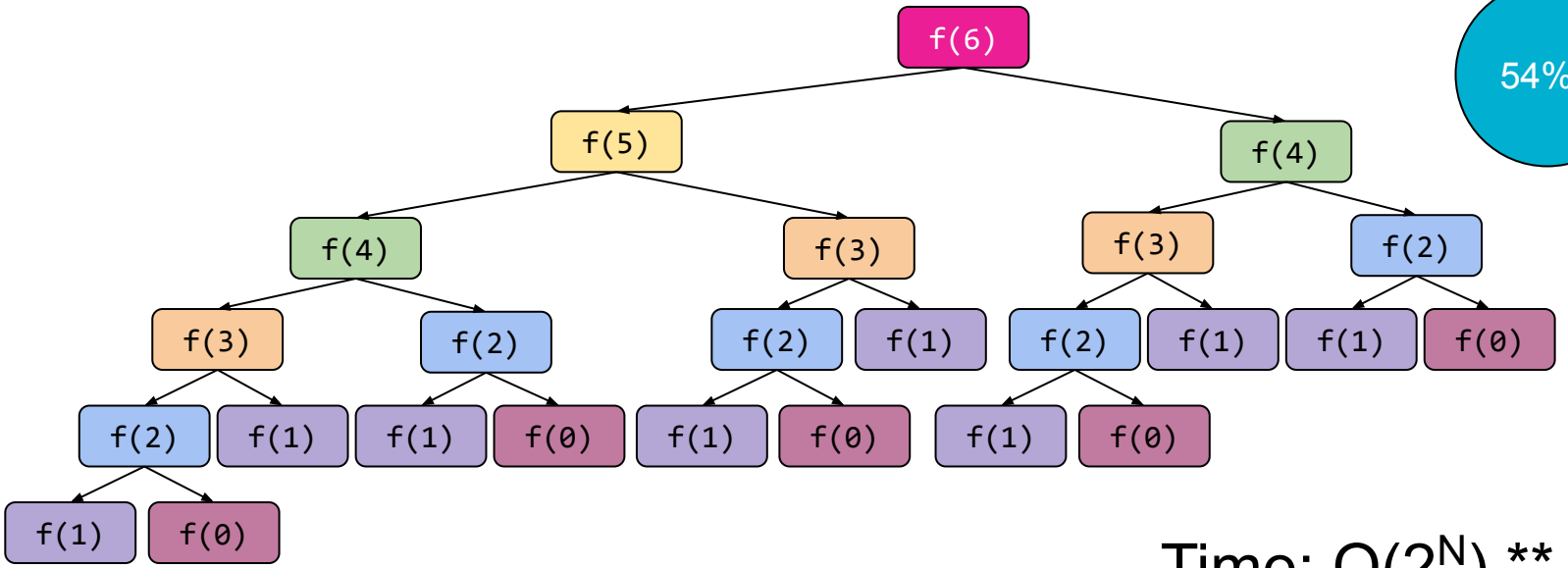
06. VALIDATE

```
1 for i from 0 to A.length: → Undefined
2     if validate(A[i])
3         print(i)
```



07. RECURSION

```
1 int fib(int n) {
2     if (n < 0) {
3         return 0;
4     } else if (n <= 1) {
5         return 1;
6     } else {
7         return fib(n - 1) + fib(n - 2);
8     }
9 }
```



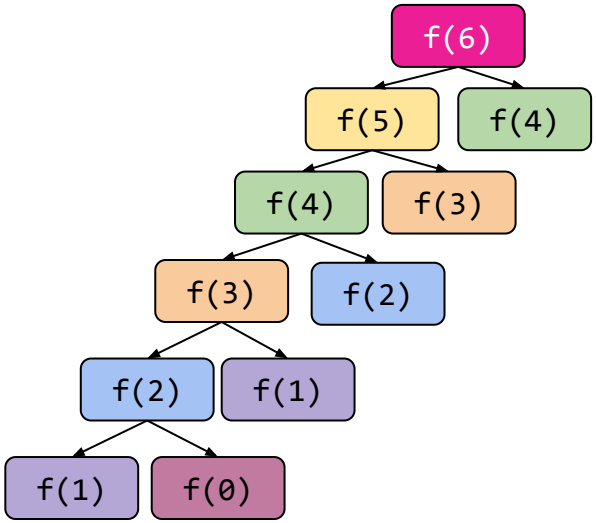
- N levels in tree
- 2 children per node → # nodes doubles at each level
- # nodes = 1 * 2 * 2 * … * 2 =

Time: $O(2^N)$ **
Space: O(N)

** actually slightly less for complex math reasons

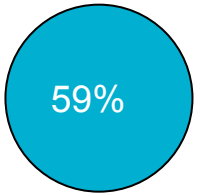
08. MEMOIZATION

```
1  int fib(int n, int[] memo) {
2      if (n < 0) {
3          return 0;
4      } else if (n <= 1) {
5          return 1;
6      } else if (memo[n] == 0) /* not computed yet */
7          memo[n]  = fib(n - 1, memo) + fib(n - 2, memo);
8      }
9      return memo[n];
10 }
```




```
1  int fib(int n, int[] memo) {
2      if (n < 0) {
3          return 0;
4      } else if (n <= 1) {
5          return 1;
6      } else if (memo[n] == 0) /* not computed yet */
7          memo[n]  = fib(n - 1, memo) + fib(n - 2, memo);
8      }
9      return memo[n];
10 }
```

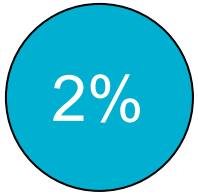
- N levels in tree
- 2 children per level
- # nodes = 1 + 2 + 2 + … + 2 =



Time: O(N)
Space: O(N)

REMINDERS

- Drop constants
  - But careful about dropping other stuff
- Use other variables
  - But avoid throwing around N
- Careful for add vs. multiply
- Recursion → call tree
  - Especially with multiple branches!



gayle@gayle.com
subject: **fbprep**

Expectations & Evaluation

Non-Expectations

More perfect is better than less perfect...

... but perfect doesn't really happen

- Knowing the answers
- Solving immediately
- Perfect coding

Evaluation & Expectations

- Thought process
  - (Not minutes to optimal answer)
- Real code
  - (Not pseudocode)

It's about signal, not perfection

- **Incorrect and not worrisome:**

```
1 LinkedList<Integer> list = new LinkedList<Integer>();
2 list.insert(0);
```

- **Incorrect and worrisome:**

```
1 if (a = 0) a += 1
```

Assuming actual confusion, not a typo

Quick tips to do well

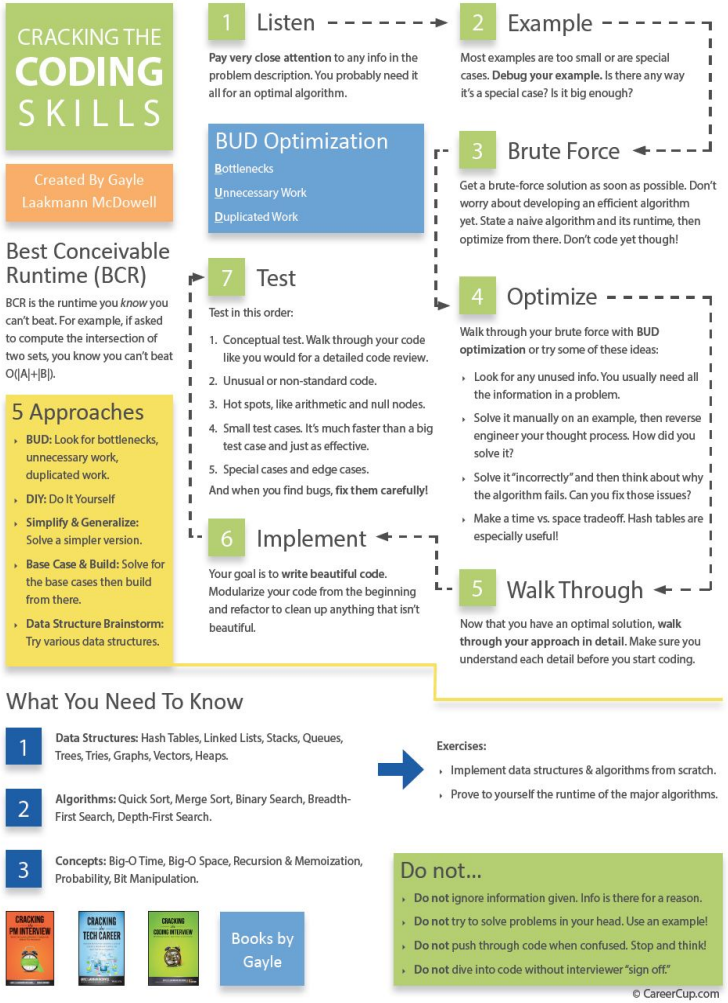
- Drive!
  - **Lead the problem solving**
  - More than just “correct”
  - Even when you’re stuck
- **Show me your thought process**
- Pay attention to meeeeeeee!
  - I might help you. Make sure you use that help.

gayle@gayle.com
subject: **fbprep**

The Seven Steps

gayle@gayle.com
subject: **fbprep**

CrackingTheCodingInterview.com
→ Resources



gayle@gayle.com
subject: **fbprep**

(no, seriously, follow it!)

CrackingTheCodingInterview.com → Resources

01. LISTEN (FOR CLUES)

01. LISTEN

(for clues)

Given two arrays that are sorted and distinct, find the number of elements in common.

01. LISTEN

(for clues)

WHY?

Given two arrays that are **sorted** and distinct, find the number of elements in common.

01. LISTEN

(for clues)

Given a list of valid English words, build an anagram server that can spit out all the anagrams of a word

(anagram = permutation that is also valid word)

rates = aster, stare, tears, tares, taser

- Caching?
- Database?
- **Precomputation?**

01. LISTEN

(for clues)

What does a server let us do?

Given a list of valid English words, build an anagram **server** that can spits out all the anagrams of a word

aerst	→	aster, stare, taser, tares, tears, rates
dgo	→	dog, god
aeelps	→	please, elapse, asleep
...		...

02. DRAW AN EXAMPLE

02. EXAMPLE

Given two arrays that are sorted and distinct, find the number of elements in common.

02. EXAMPLES // BAD VS. GOOD

A: [1, 12, 15, 19]

too small
special case-y

B: [2, 12, 13, 20]

Makes you
work for it

A: [1, 12, 15, 19, 20, 21]

B: [2, 15, 17, 19, 21, 25, 27]

SOMETHING THAT MAKES YOU WORK

BIG & GENERIC

02. EXAMPLE

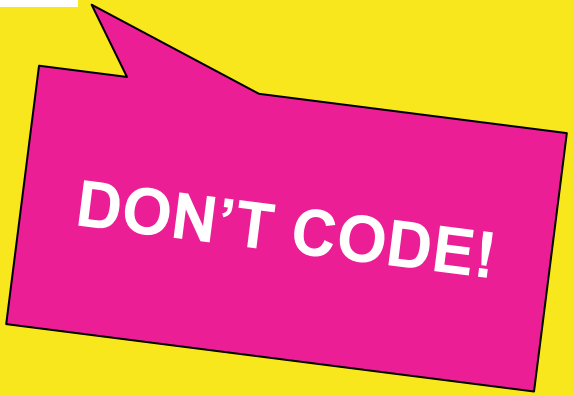
Make your examples large and avoid special cases!

03. BRUTE FORCE

03. BRUTE FORCE

Slow & terrible > nothing

- Explain the best algorithm you have (even slow and stupid)
- State runtime
- Optimize!



04. OPTIMIZE

04. OPTIMIZE

expect to maybe spend
a while here

- BUD
- Space & Time
- Do It Yourself
- Recursion

4a. BUD

- **B**ottlenecks
- **U**nnecessary work
- **D**uplicated work

4a. (B)UD // BOTTLENECKS

Given two arrays that are sorted and distinct, find the number of elements in common.

```
[1, 12, 15, 19, 20, 21]

[2, 15, 17, 19, 21, 25, 27]
```

- Brute Force: O(A*B)
- Binary Search: O(A log B)
- Hash Set: O(A + B)
  - But O(B) space
- Sorted Merge:
  - O(A+B) time
  - O(1) space

4a. B(U)D // UNNECESSARY WORK

Find all solutions to

$$a^3 + b^3 = c^3 + d^3$$

where $1 <= a, b, c, d < n$

4a. B(U)D // UNNECESSARY WORK

All solutions to $a^3 + b^3 = c^3 + d^3$

```
1 for a from 1 to N
2     for b from 1 to N
3         for c from 1 to N
4             for d from 1 to N
5                 if a^3 + b^3 == c^3 + d^3
6                     print a, b, c, d
                      break;
```

4a. B(U)D // UNNECESSARY WORK

Find d in

$$4^3 + 32^3 = 18^3 + d^3$$

$$d^3 = 4^3 + 32^3 - 18^3$$

$$d = (4^3 + 32^3 - 18^3)^{⅓}$$

$$d = 30$$

4a. B(U)D // UNNECESSARY WORK

All solutions to $a^3 + b^3 = c^3 + d^3$

```
1 for a from 1 to N
2     for b from 1 to N
3         for c from 1 to N
4             d = power(a^3 + b^3 - c^3, 1 / 3)
5             if d is an integer
6                 print a, b, c, d
```

4a. BU(D) // DUPLICATED WORK

All solutions to $a^3 + b^3 = c^3 + d^3$

```
1 for a from 1 to N
2     for b from 1 to N
3         for c from 1 to N
4             for d from 1 to N
5                 if a^3 + b^3 == c^3 + d^3
6                     print a, b, c, d
```

4a. BU(D) // DUPLICATED WORK

All solutions to $a^3 + b^3 = c^3 + d^3$

```
1 for a, b from 1 to N
2     for c, d from 1 to N
3         if a^3 + b^3 = c^3 + d^3
4             print a, b, c, d
```

4a. BU(D) // DUPLICATED WORK

$$a^3 + b^3 = c^3 + d^3$$

```
1 for a, b from 1 to N
2     for c, d from 1 to N
3         if a^3 + b^3 = c^3 + d^3
4             print a, b, c, d
```

c	d	$c^3 + d^3$
…	…	…
4	31	29855
4	32	32832
4	33	36001
…	…	…
18	29	30221
18	30	32832
18	31	35623
…	…	…

4a. BU(D) // DUPLICATED WORK

$$a^3 + b^3 = c^3 + d^3$$

```
1 for a, b from 1 to N
2     for c, d from 1 to N
3         if a^3 + b^3 = c^3 + d^3
4             print a, b, c, d
```

$c^3 + d^3$	→	(c, d)
…	→	…
29855	→	(4, 31)
32832	→	(4, 32), (18, 30)
36001	→	(4, 33)
…		…
216125	→	(5, 60), (45, 50)
227106	→	(5, 61)
…	→	…

4a. BU(D) // DUPLICATED WORK

```
lookup = new map from integer -> list of pairs
for c from 1 to n
    for d from 1 to n
        result = c^3 + d^3
        listOfPairs = lookup.get(result) // get list of pairs with this cube sum
        listOfPairs.append(new Pair(c, d)) // append the new pair

for a from 1 to n
    for b from 1 to n
        result = a^3 + b^3
        listOfPairs = lookup.get(result) // find everything with the same cube sum
        for each pair in listOfPairs
            print a, b, pair.1, pair.2
```

4b. Space / Time Tradeoffs

- Hashtables
- Other data structures
  - Tries, graphs, heaps, etc
- Precomputation
  - & Reorganizing input

4c. Do It Yourself (DIY)

Given two strings, s and b,
find all permutations of
s within b

$s = \text{abbcd}$

$b =$

$s = \text{abbcd}$

$b =$

$s = \text{abbcd}$

$b =$

b	a	b	c	d	b	a	e	f	d	b	b	a	c	b	d	d	f	a	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

4c. Do It Yourself

1. Create nice big example input
    - Remember step 2! BIG & GENERIC
    - Something that makes you WORK
2. Get the output *instinctively*
    - Like you would if you'd never coded ever
    - Forget about computer science, interviews, algorithms, etc
3. Reverse engineer your thought process
    - What exactly did you do?
    - Pay attention to the little optimizations
4. Translate into algorithm

$s = \text{abbcd}$

$b =$

4d. Recursion

- Use recursion instinct, but don't cling to it
- Try bottom-up
  - Subsets
    - Base cases:
      {} → {}
      {a} → {}, {a}
      {a, b} → {}, {a}, {b}, {a, b}
    - Build up:
      {a, b, c} → …
      {a, b, c, d} → ...
- Backtracking → recursion
- Multi-branch? Draw call-tree
  - Get (exponential?) runtime
  - Find repeated problems
  - Memoize

04. OPTIMIZE

expect to maybe spend
a while here

- BUD
- Space & Time
- Do It Yourself
- Recursion

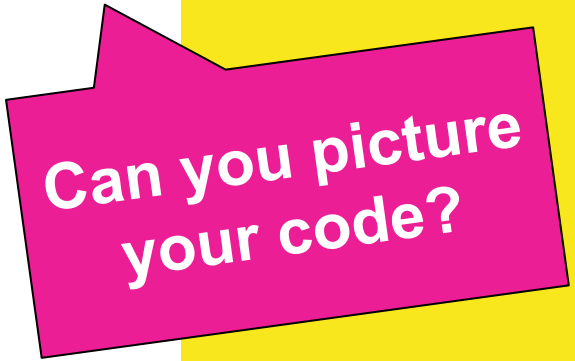
Your perception is *not* predictive of performance

05. WALK THROUGH

Rushing wastes time.

Take the time to *deeply* understand your algorithm

- What variables / data structures
- How do they change
- How is your code structured



06. CODE

Code *beautifully*
Not just correctly

- Correctness
- Readability
- Maintainability
- Performance

Show me you're a great coder

06. CODE // MAKING YOUR LIFE EASIER

- Write straight

// BAD

```
boolean iAmAwesome(Person p) {
   if (p == this) { // Only me
      return true;
   } else {
      return false;
   }
   return false;
   /* TODO: find more concise way of
    * coding this. */
}
```

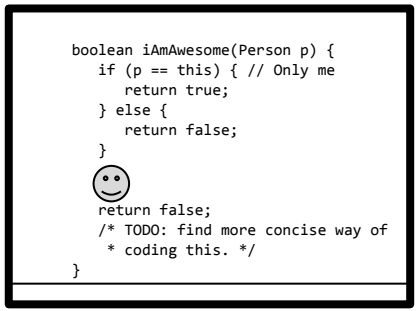
// GOOD

```
boolean iAmAwesome(Person p) {
   if (p == this) { // Only me
      return true;
   } else {
      return false;
   }
   return false;
   /* TODO: find more concise way of
    * coding this. */
}
```

06. CODE // MAKING YOUR LIFE EASIER

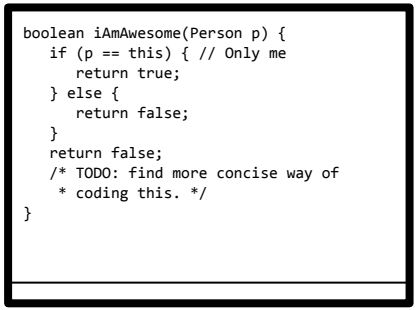
- Write straight
- Code in top-left corner
  - Get rid of stuff in your way

// BAD



```
boolean iAmAwesome(Person p) {
   if (p == this) { // Only me
      return true;
   } else {
      return false;
   }
   return false;
   /* TODO: find more concise way of
    * coding this. */
}
```

// GOOD

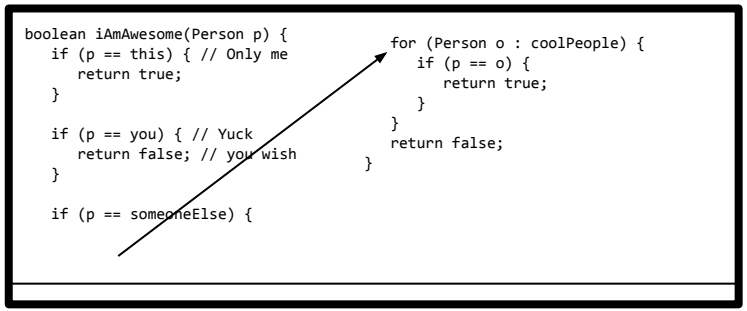


```
boolean iAmAwesome(Person p) {
   if (p == this) { // Only me
      return true;
   } else {
      return false;
   }
   return false;
   /* TODO: find more concise way of
    * coding this. */
}
```

06. CODE // MAKING YOUR LIFE EASIER

- Write straight
- Code in top-left corner
  - Get rid of stuff in your way
- Arrows are fine

// FINE



06. CODE // MAKING YOUR LIFE EASIER

- Write straight
- Code in top-left corner
  - Get rid of stuff in your way
- Arrows are fine

- Shorthand probably okay
  - (But ask first)

/* instead of this */

```
if (a != null && a.length > 0 &&
    b != null && b.length > 0 &&
    c != null && c.length > 0 &&
    d != null && d.length > 0) {
    ...
}
```

```
HashMap<String, Integer> myHashMap = new HashMap<String, Integer>();
```

```
node.numberOfChildren = ...
if (node.numberOfChildren > 0) {
    ...
}
```

/* try this */

```
if (a != null && a.length > 0 &&
    ... b, c, d...) {
}
```

```
HM<S, I> myHashMap = new HM<S, I>();
```

```
node.numberOfChildren = ...
if (node.n~c~ > 0) {
    ...
}
```

06. CODE // SHOWING YOU'RE A GOOD CODER

- Yes, consider error cases, boundary checks, etc
  - (Shorthand might be useful)
  - At least discuss!

/* instead of this */

```
1  if (a != null && a.length > 0 &&
2      b != null && b.length > 0 &&
3      c != null && c.length > 0 &&
4      d != null && d.length > 0) {
5      ...
6  }
```

/* try this */

```
1  if (a != null && a.length > 0 &&
2      ... b, c, d...) {
3  }
```

```
1  boolean findMinMaxSubarray(int[] array) {
2      for (int a : array) {
3          if (a < 0) {
4              return false;
5          }
6      }
7      ...
8      ...
```

```
1  boolean findMinMaxSubarray(int[] array) {
2      if (!validate(array)) return false;
3      ...
4      ...
```

06. CODE // SHOWING YOU'RE A GOOD CODER

- Yes, style matters (even on a whiteboard!)
  - Good variable names
  - Consistent spacing

ADHERE TO A STYLE GUIDELINE

/* instead of this */

```
boolean doSomething(String a, String b) {
    int[] ss = new int[a.length + b.length];
    ...
}
```

```
if (x== 0){
    ...
}
for(i=0;i < N;i++)
{
    ...
}
```

/* try this */

```
boolean doSomething(String shorter, String longer) {
    int[] mergedStringCounts = new int[s~.length + l~.length];
    ...
}
```

```
if (x == 0) {
    ...
}
for (i = 0; i < N; i++) {
    ...
}
```

06. CODE // SHOWING YOU'RE A GOOD CODER

- Languages
  - Specialists? Use that one.
  - Everyone else? Use your best one.

	Advantages	Disadvantages	Mitigation
Java, Objective C		Verbose Fewer built-in	Use shorthand Modularize
Ruby, Python, JS	Built-in functions	… those aren't always O(1)	Just be careful
C		**Many** fewer built-ins	Modularize

06. CODE // SHOWING YOU'RE A GOOD CODER

- Modularize (upfront!)

/* instead of this */

```
public static boolean canBuildRansomNote1(String magazine, String note) {
    // Count ransom note
    int[] noteCount = new int[26];
    for (int i = 0; i < note.length(); i++) {
        int c = (int) note.charAt(i);
        if (c >= (int) 'a' && c <= ((int) 'z')) {
            c -= (int) 'a';
        } else if (c >= (int) 'A' && c <= ((int) 'Z')) {
            c -= (int) 'A';
        }
        if (0 <= c && c < 26) {
            noteCount[c]++;
        }
    }

    // Count magazine
    int[] magazineCount = new int[26];
    for (int i = 0; i < magazine.length(); i++) {
        int c = (int) magazine.charAt(i);
        if (c >= (int) 'a' && c <= ((int) 'z')) {
            c -= (int) 'a';
        } else if (c >= (int) 'A' && c <= ((int) 'Z')) {
            c -= (int) 'A';
        }
        if (0 <= c && c < 26) {
            magazineCount[c]++;
        }
    }

    // Compare
    for (int i = 0; i < magazineCount.length; i++) {
        if (noteCount[i] > magazineCount[i]) {
            return false;
        }
    }
    return true;
}
```

/* try this */

```
public static boolean canBuildRansomNote2(String magazine, String note) {
    int[] noteCount = buildCharFrequencyTable(note); // Count ransom note
    int[] magazineCount = buildCharFrequencyTable(magazine); // Count magazine
    return isIncluded(magazineCount, noteCount); // Compare
}

public static int[] buildCharFrequencyTable(String sequence) {
    int[] counter = new int[26];
    for (int i = 0; i < sequence.length(); i++) {
        int c = convertCharToNumber(sequence.charAt(i));
        if (c > 0) {
            counter[c]++;
        }
    }
    return counter;
}

public static boolean isIncluded(int[] magazineCount, int[] noteCount) {
    for (int i = 0; i < magazineCount.length; i++) {
        if (noteCount[i] > magazineCount[i]) {
            return false;
        }
    }
    return true;
}

public static int convertCharToNumber(char ch) {
    int c = (int) ch;
    if (c >= (int) 'a' && c <= ((int) 'z')) {
        return c - (int) 'a';
    } else if (c >= (int) 'A' && c <= ((int) 'Z')) {
        return c - (int) 'A';
    }
    return -1;
}
```

06. CODE // SHOWING YOU'RE A GOOD CODER

- Modularize (upfront!)

/* instead of this */

I've learned nothing

```
boolean canSplitEqually(int[] array) {
    int sum = 0;
    for (int i = 0; i < array.length; i++) {
        sum += array[i];
    }
    if (sum % 2 != 0) {
        return false;
    }
    return canMakeSum(array, 0, sum / 2);
}

boolean canMakeSum(int[] array, int index, int targetSum) {
    ...
}
```

/* try this */

```
boolean canSplitEqually(int[] array) {
    int sum = sumOfArray(array);
    if (sum % 2 != 0) {
        return false;
    }
    return canMakeSum(array, 0, sum / 2);
}

boolean canMakeSum(int[] array, int index, int targetSum) {
    ...
}
```

06. CODE // EFFECTIVE WHITEBOARD CODING

- Write straight
- Code in top-left corner
  - Erase stuff you don’t need
- Arrows are fine
- Shorthand probably okay (ask)
  - Especially in a verbose language
- Consider error cases and boundaries
  - At minimum, mention them
- Style matters
  - Good variable names
  - Consistent spacing, etc
- Use your best language
- Modularize (upfront!)

07. TEST

Test code (not algorithm).
Think as you test.
Think before you fix.

1. Conceptual Testing
    a. Walk-through
    b. High-risk code
2. Test cases
    a. Small test cases
    b. Edge cases
    c. Big test cases

07. TEST // TESTING PROCESS

- Conceptual walk-through
  - Think about each line of code
- Double check high-risk lines of code

- Small tests are more effective
  - Faster
  - You'll be more thorough
- Edge cases
  - Boring ones: nulls, empty
  - Interesting ones: base cases, all same chars, etc

Plus, most likely bug would be a minor typo, like swapping a and b

/* low risk */

```
1 String x = processString(a, b);
```

test cases... *maybe*

/* ridiculously high risk */

```
1 String x = a.substring(i, i + k / 2 - 1);
```

07. TEST // EFFECTIVE TESTING

- Test your code, not your algorithm

"It's supposed to get the char counts. Therefore, I decree that it does."

"This is supposed to walk through until the end. Therefore, I decree that it does."

```
1  int countPermutations(String s, String b) {
2      int count = 0;
3      int[] sCharCounts = getCharCounts(s);
4      for (int i = 0; i < b.length - s.length; i++) {
5          int[] bCharCounts = getCharCounts(b);
6          if (isEqual(sCharCounts, bCharCounts)) {
7              count++;
8          }
9      }
10     return count;
11 }
12
13 char[] getCharCounts(String s) { /* CASE INSENSITIVE */
14     int[] charCounts = new int[26];
15     for (int i = 0; i < s.length; i++) {
16         char c = s.getChar(i);
17         charCounts[c.toLower()] += 1;
18     }
19     return charCounts;
20 }
```

07. TEST // EFFECTIVE TESTING

- Test your code, not your algorithm
- Think through the partial results

"Got zero here. Cool, whatever.
I'll just check result at the end."

"Got 15 here. Cool, whatever.
I'll just check result at the end."

```
1  int countPermutations(String s, String b) {
2      int count = 0;
3      int[] sCharCounts = getCharCounts(s);
4      for (int i = 0; i < b.length - s.length; i++) {
5          int[] bCharCounts = getCharCounts(b);
6          if (isEqual(sCharCounts, bCharCounts)) {
7              count++;
8          }
9      }
10     return count;
11 }
12
13 char[] getCharCounts(String s) { /* CASE INSENSITIVE */
14     int[] charCounts = new int[26];
15     for (int i = 0; i < s.length; i++) {
16         char c = s.getChar(i);
17         charCounts[c.toLower()] += 1;
18     }
19     return charCounts;
20 }
```

07. TEST // EFFECTIVE TESTING

- Test your code, not your algorithm
- Think through the partial results
- Think before you fix

What happens if:
s = “cat”
b = “cat”

```
3     int[] sCharCounts = getCharCounts(s);
4     if (b.length == s.length) { /* special case on the same length */
5         int[] bCharCounts = getCharCounts(b);
6         return isEqual(sCharCounts, bCharCounts) ? 1 : 0;
7     }
8     for (int i = 0; i < b.length - s.length; i++) {
```

```
3       int[] sCharCounts = getCharCounts(s);
4       for (int i = 0; i < b.length - s.length + 1; i++) {
```

```
1  int countPermutations(String s, String b) {
2      int count = 0;
3      int[] sCharCounts = getCharCounts(s);
4      for (int i = 0; i < b.length - s.length; i++) {
5          int[] bCharCounts = getCharCounts(b);
6          if (isEqual(sCharCounts, bCharCounts)) {
7              count++;
8          }
9      }
10     return count;
11 }
12
13 char[] getCharCounts(String s) { /* CASE INSENSITIVE */
14     int[] charCounts = new int[26];
15     for (int i = 0; i < s.length; i++) {
16         char c = s.getChar(i);
17         charCounts[c.toLower()] += 1;
18     }
19     return charCounts;
20 }
```

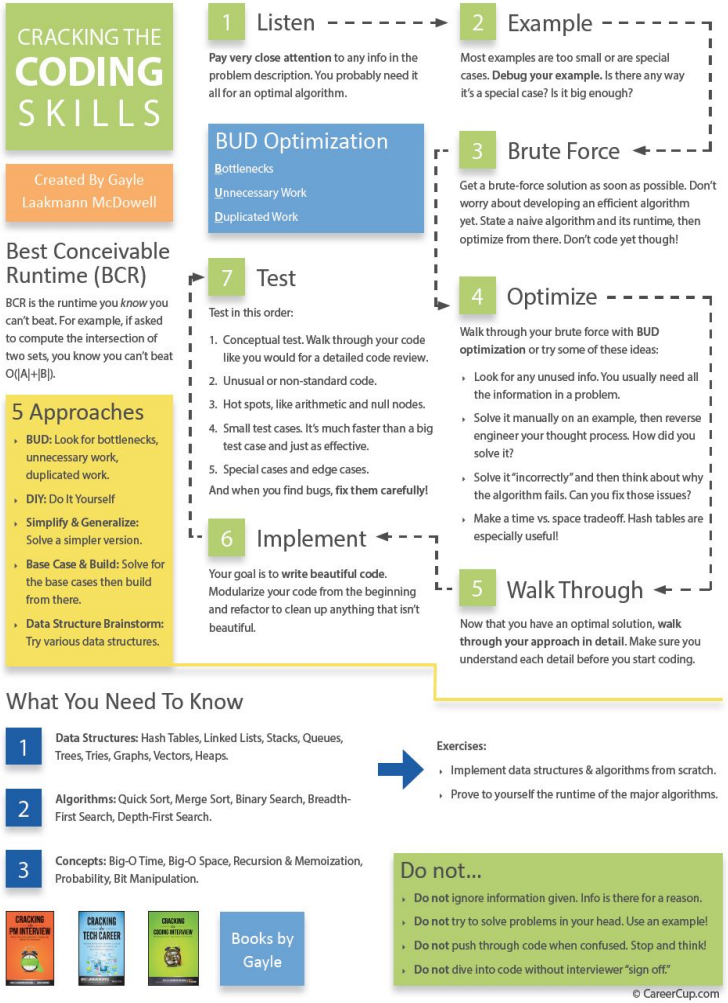
07. TEST // EFFECTIVE TESTING

- Conceptual Testing
  - Walk-through
  - High-risk code
- Test cases
  - Small test cases
  - Edge cases
  - Big test cases

- Test code, not algorithm
- Think through partial results
- Think before you fix

gayle@gayle.com
subject: **fbprep**

CrackingTheCodingInterview.com
→ Resources



gayle@gayle.com
subject: **fbprep**

If You Forget Everything Else

07. TEST // EFFECTIVE TESTING

- Create Big Input
  - Big
  - Generic
  - *Make yourself do work*
- Walk Through It
  - Get output
  - Instinctively

- **DO NOT CODE UNTIL YOU'RE READY!**
  - Interviewer wants you to code
  - You know *exactly* what you're doing

gayle@gayle.com
subject: **fbprep**

Questions For Interviewer

Don’t sweat this.

Inspiration for Questions

- What's made you happy / unhappy in past jobs?
- What are your career goals?
- Culture & work habits
- Career paths
- Technology, infrastructure & tools
- Interviewer's experience

gayle@gayle.com
subject: **fbprep**

Final Thoughts

Great Engineer
&
Great Teammate

Your perception is *not* predictive of performance

Just keep working at it.

Little Things

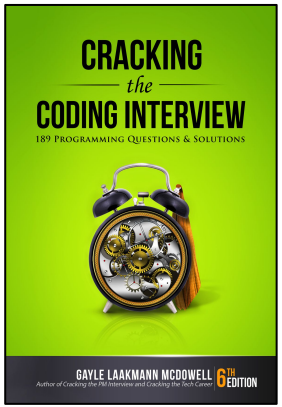
- Survey?
  - Coming soon
- Ready?
  - Follow up with recruiter
- Job postings?

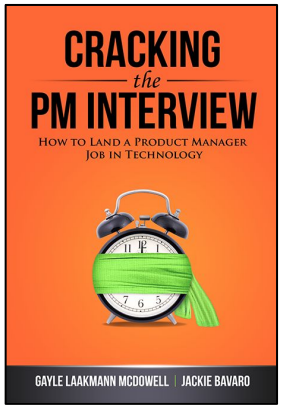
facebook.com/careers/teams/engineering

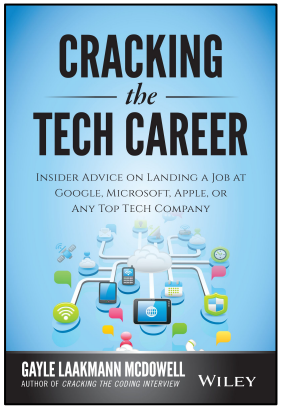
- Slides?
  - Gayle.com → Events → Facebook

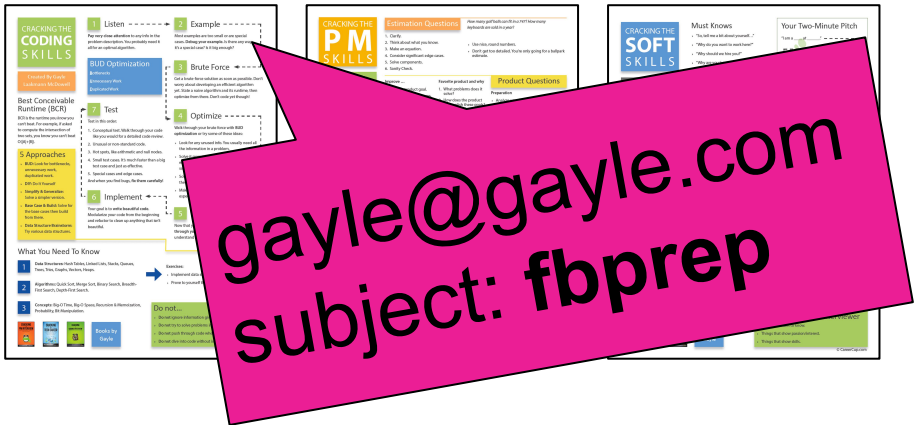
gayle@gayle.com
subject: **fbprep**

More Resources









Gayle.com

CareerCup.com

CrackingTheCodingInterview.com

// FOLLOW ME

@gayle

@gayle

@gayle-laakmann-mcdowell