Coding Interview

Idaho State University | Computer Science

Isaac Griffith

Association for Computing Machinery
Department of Informatics and Computer Science
Idaho State University

ROAR

Inspiration

"An open mind, a deep sense of curiosity, and constant desire to learn. You can't be afraid of going into an area that you don't know much about – you have to be comfortable getting up to speed quickly in new and potentially intimidating areas. You need to be a consummate and life-long learner. The key is to ask questions, be curious and learn from your team." – Gayle Laakmann McDowell

Idaho State University
Computer Science

How CS Resumes Should Look

- **One Page Only!** unless > 10 years experience
- **A Real Resume Format** with organized columns
- **Short** (1 – 2) line bullets
- **Focus on Accomplishments** not responsibilities
  - "Accomplished X by implementing Y which led to Z"
- **GPA** if at lest 3.0 min (in-major, overall)
- **3 – 4 Projects** Courses & independent, finished or unfinished
- **List of Technical Skills** Short! Cut the "fluff"
- **No Objective** – Objectives/summaries are almost always useless

ROAR

Coding Interviews

- Can be part of
  - On-site Interview
  - Remote Interview
- Typically involve a non-ide based coding process
  - Whiteboard
  - Collaborative Editor

Idaho State University
Computer Science

How You're Judged

How did you do **RELATIVE to other** candidates on the **SAME** question?

- It's not about how quickly you solved the problem…
- … it's about how quickly you solved it **relative** to other candidates

ROAR

What Really Happens

- Knowledge Questions
  - Coding
- Design/Scalability
  - Coding
- Algorithms/Problem Solving
  - Coding

Technical Questions

1. Ask Questions!
   - Questions are more ambiguous than they appear
2. Talk out loud
   - Show us how you think
3. Think critically
   - Does your algorithm really work? What's the space and time complexity
4. Code slowly and methodically
   - It's not a race
5. Test your code
   - And make CAREFUL fixes

Knowledge Questions

- If you list it, know it
- If you don't know it, admit it
  - Derive it if possible

Idaho State University
Computer Science

How to Approach

1. **S**cope the Problem
   - Ask questiosn
   - Make appropriate assumptions
2. Define **K**ey Components
   - Can be somewhat naive
3. **I**dentify **I**ssues
   - Bottlenecks, tradeoffs
4. **R**epair & **R**edesign

ROAR

How to Prepare

- Read about design of major companies
  - Twitter, Facebook, Quora, Google, etc.
  - Think about WHY they're designed that way
- Learn/review key concepts
  - Task queues, databases, sharding, etc.
- Practice questions

Idaho State University
Computer Science

Why?

- String CS fundamentals
- Analytical skills
- Make tradeoffs

- Push through hard problems
- Communication
- How you think

ROAR

Essential Knowledge

Data Structures	Algorithms	Concepts
ArrayLists	Merge Sort	Big O Time
Hash Tables	Quick Sort	Big O Space
Trees (+ Tries)	Breadth-First Search	Recursion
Graphs	Depth-First Search	Recursion
Stacks/Queues	Binary Search	
Heaps		

Idaho State University
Computer Science

Essential Knowledge

Data Structures	Algorithms	Concepts
ArrayLists	Merge Sort	Big O Time
Hash Tables	Quick Sort	Big O Space
Trees (+ Tries)	Breadth-First Search	Recursion
Graphs	Depth-First Search	Recursion
Stacks/Queues	Binary Search	
Heaps		

CS 2235!!!

ROAR

Preparation

- Practice Implementation of DS/Algorithms
- MASTER Big O
- Practice with interview questions
- Code on paper/whiteboard
- Mock interviews

Preparation

- Practice Implementation of DS/Algorithms
- MASTER Big O
- Practice with interview questions
- Code on paper/whiteboard
- Mock interviews

PUSH YOURSELF!

What is NOT expected

- To know the answers
- To solve immediately
- To code perfectly

What is expected

- Be excited about hard problems
- More than just “correct”
- Drive!
- Keep trying when stuck
- Write real code

What is expected

- Be excited about hard problems
- More than just “correct”
- Drive!
- Keep trying when stuck
- Write real code

Show them how you think!

Idaho State University
Computer Science

Approach

1. Listen (for clues)
2. Draw an Example
3. Brute Force / Naive
4. Optimize
5. Walkthrough
6. Write Beautiful Code
7. Testing

ROAR

Best Conceivable Runtime

- BCR is the runtime you know you can't beat.
- For example:
  - If asked to compute the intersection of two sets
  - You know you can't beat $O(|A| + |B|)$

Step 1 Listen (for clues)

- **Pay very close attention**
  - absorb all info from the problem description
- You will need this for an optimal algorithm

What's the clue?

- Anagram server
  - Ex: rates -> aster, stare, taser, tears
- Clue: why is it on a server?

Step 2 Draw an Example

- Most examples are too small or are special cases
- **Debug your example.**
  - Is there any way it's a special case
  - Is it big enough?

Example

Intersection of Two Sorted Arrays

- Most people draw something like this:

$$[1, \underline{12}, 15, 19]$$

$$[2, \underline{12}, 13, 20]$$

- Too small
- Too special-case-y
  - same size, one common element, same index

Example

- Better:

$$[1, 12, \underline{15}, \underline{19}, 20, \underline{21}]$$

$$[2, \underline{15}, 17, \underline{19}, \underline{21}, 25, 27]$$

- Big
- No special cases

Step 3

- Get a brute-force solution as soon as possible
- Don’t worry about developing an efficient algorithm yet
- State a naive algorithm and its runtime
  - then optimize from there
- Don’t code yet!

Idaho State University
Computer Science

Step 4 Optimize

Walkthrough your Brute force with

- Optimize
  - BUD
  - Space/Time
  - Do it yourself
- Solve
  - Recursion
  - Solve “incorrectly”
  - Other data structures (i.e., HashTables)

Push Yourself

ROAR

Look for BUD

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

What's the bottleneck?

- Ex: counting the intersection

$$[1, 12, \underline{15}, \underline{19}, 20, \underline{21}]$$

$$[2, \underline{15}, 17, \underline{19}, \underline{21}, 25, 27]$$

- Bottleneck: searching

What's unnecessary?

- Ex: $a^3 + b^3 = c^3 + d^3$ where (1 <= a, b, c, d <= 1000)

```
n = 1000
for a from 1 to n
  for b from 1 to n
    for c from 1 to n
      for d from 1 to n
        if a^3 + b^3 == c^3 + d^3
          print a, b, c, d
```

What's unnecessary?

- Ex: $a^3 + b^3 = c^3 + d^3$ where (1 <= a, b, c, d <= 1000)

```
n = 1000
for a from 1 to n
  for b from 1 to n
    for c from 1 to n
      for d from 1 to n
        if a^3 + b^3 == c^3 + d^3
          print a, b, c, d
```

- Unnecessary: looking for d

What's unnecessary?

- Ex: $a^3 + b^3 = c^3 + d^3$ where (1 <= a, b, c, d <= 1000)

```
n = 1000
for a from 1 to n
  for b from 1 to n
    for c from 1 to n
      d = pow(a^3 + b^3 - c^3, 1/3) // will round to int
      if a^3 + b^3 == c^3 + d^3
        print a, b, c, d
```

- Unnecessary: looking for d

What's duplicated?

- Ex: $a^3 + b^3 = c^3 + d^3$ where (1 <= a, b, c, d <= 1000)

```
n = 1000
for a from 1 to n
  for b from 1 to n
    for c from 1 to n
      for d from 1 to n
        if a^3 + b^3 == c^3 + d^3
          print a, b, c, d
```

What's duplicated?

- Ex: $a^3 + b^3 = c^3 + d^3$ where (1 <= a, b, c, d <= 1000)

```
n = 1000
for a from 1 to n
  for b from 1 to n
    for c from 1 to n
      for d from 1 to n
        if a^3 + b^3 == c^3 + d^3
          print a, b, c, d
```

- Duplicated: c, d pairs

What's duplicated?

- Ex: $a^3 + b^3 = c^3 + d^3$ where (1 <= a, b, c, d <= 1000)

```
n = 1000
for a, b from 1, 1 to n, n
  for c, d from 1, 1 to n, n
    if a^3 + b^3 == c^3 + d^3
      print a, b, c, d
```

- Duplicated: c, d pairs

What's duplicated?

- Ex: $a^3 + b^3 = c^3 + d^3$ where (1 <= a, b, c, d <= 1000)

```
n = 1000
for c from 1 to n
  for d from 1 to n
    result = c^3 + d^3
    append (c, d) to list at value map[result]
for a from 1 to n
  for b from 1 to n
    list = map.get(result)
    for each pair in list
      print a, b, pair
```

What's duplicated?

- Ex: $a^3 + b^3 = c^3 + d^3$ where (1 <= a, b, c, d <= 1000)

```
n = 1000
for c from 1 to n
  for d from 1 to n
    result = c^3 + d^3
    append (c, d) to list at value map[result]

for each result, list in map
  for each pair1 in list
    for each pair2 in list
      print pair1, pair2
```

Space/Time tradeoffs

- Hash tables & other data structures
- Precomputing

Precomputing

- Find rectangle at origin with biggest sum
- Brute force: compute all rectangles and sums

Do it yourself

- Find permutations of s within b
  - s = abbc
  - b = babcabbacaabcbabcacbb
- Find them!
  - … now how did you actually do it?

Techniques to Develop Algorithms

- Optimize
  - BUD
  - Space/Time
  - Do it yourself
- Solve
  - Recursion
  - Solve “incorrectly”
  - Other data structures

Recursion / Base Case & Build

- Subsets of a set
  - {} -> {}
  - {a} -> {}, {a}
  - {a, b} -> {}, {a},. {b}, {a b}
  - {a, b, c} -> ...
- Subsets of $S_1 \ldots S_{n-1} + S_n$ to each

Solve “incorrectly”

1. Develop **incorrect** solution
2. Identify why **precisely** it’s incorrect
3. Repair
4. (& Repeat)

Other Data Structures

- Giving out phone numbers
  - "I want any available number"
  - "I want **this** number"
- Try: Sorted array? Sorted linked list? Hash table? BST?

Step 5 Walkthrough

- With optimal solution in hand...
- Walkthrough your approach
- Understand each detail before coding

Step 6 Implement

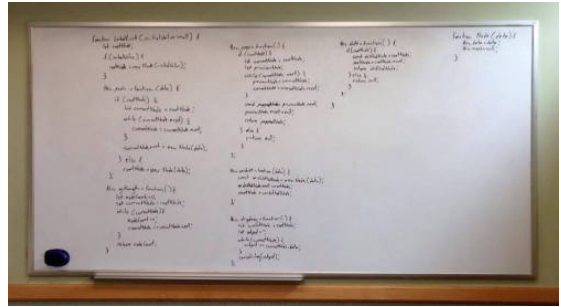
Write Beautiful Code

- Be methodical. Don't try to rush.
- Reasonably Bug Free
  - Thorough testing (and careful fixing)
  - Check for error conditions
- Clean coding
  - Use other functions
  - Good use of data structures (define own if useful)
  - Concise and readable
- Refactor

Idaho State University
Computer Science

How to Write Whiteboard Code

- Write straight
- Top-left corner
- Use arrows if needed
- Error cases / TODOs
- Good variables
- Modularized



Language choice is up to you!

Practice! Practice! Practice!

ROAR

Step 7 Testing

Test in this order:

1. **Conceptual test:**
   - Walk through you code like you would for a detailed code review.
2. **Unusual** or non-standard code
3. **Hot spots**
   - like arithmetic and null nodes
4. **Small test cases:**
   - It's much faster than a big test case and just as effective
5. **Special cases and edge cases**

- When you find bugs, **fix them carefully!**

Final Thoughts

Idaho State University
Computer Science

After Your Interview

- Follow-up with your recruiter/interviewer
  - No response != rejection
- You have **no idea** how well/poorly you did.
  - Seriously. I know you think you do. But you don't
- Lots of randomness
  - So if you fail, get up and try again.

ROAR

Are there any questions?

References

- The content of this came from Gayle Laakmann McDowell's books "Cracking the Coding Interview"
- And the handout "Cracking the Coding Skills" also by Gayle Laakmann McDowell
- The content of the slides are borrowed from her accompanying slides. I take no credit for this information!
- I suggest that you go out and buy a copy of her book and read it.

Are there any questions?