Hacking the Coding Interview

Career Fair 2016: January 24-25

Goals

- Basics of interviewing
- Technical Interviews
- General topics
- Practice Questions

The Non-Coding Part

- Boring, but INCREDIBLY IMPORTANT – You can be the best coder, but that won't matter if you don't fit in with the company culture
- "Getting to know you questions"
- Walk through your resume
- Behavioral Questions

Prep Work

- Do research
  - Look at Glassdoor, go to company website
  - This will also help narrow down which technical questions to practice more
  - What is the company culture like?
- Tailor your resume
- Make a list of questions to ask the interviewer

- Questions they ask you:
  - Tell me about yourself.
  - What is a current project you are working on?
  - Why do you want to work here?
  - Where do you see yourself in 5 years?
  - What are your strengths and weaknesses?
  - Give me an example of a situation where you disagreed with someone on their approach
  - How do you manage time when working in a fast paced work environment

- Questions you ask them:
  - What is a current project that you're working on?
  - How is this project different from projects at other companies?
  - What is in your opinion the coolest thing you've worked on (If talking to an engineer)

Any Questions?

The Technical Interview

- Topics to absolutely know:
  - Big-O Notation
  - Algorithms
  - Data Structures
  - Conceptual questions
- Programming Languages: C/C++, JavaScript, Python
  - More specific languages: Swift, HTML, CSS, R, STATA, etc.

Topics

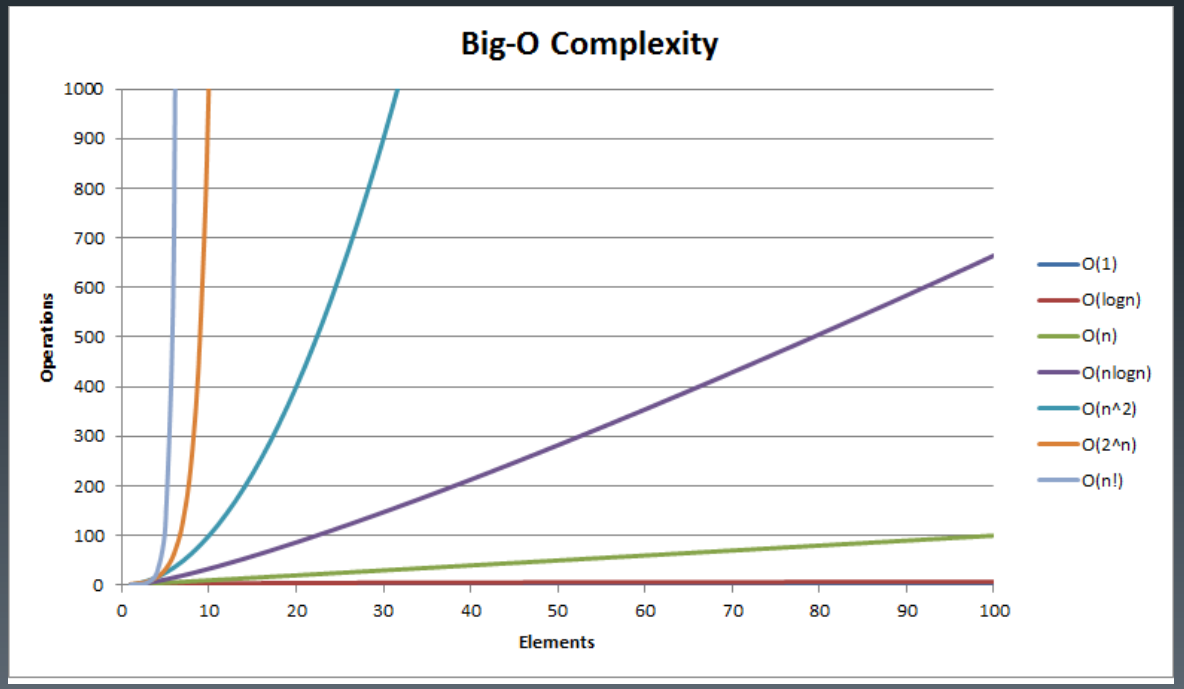
- Big O Notation
- Data structures
- Algorithms
- Other

Big O Notation

- A way of stating the overall complexity of a solution
- I.e.) How much time will this take to run
- Based on the size of input “n”
- Ex) Given an unsorted array of n doubles, find out if for each item, the double = item / 2 is also in the array
- We need to go through every item, and for each item we have to look at every other item.
- Therefore our solution is $O(n^2)$

Big O Notation

- Common complexities…
- Constant = O(1)
- Logarethmic = O(log n)
- Linear = O(n)
- Quadratic = $O(n^2)$
- Exponential = $O(2^n)$
- Factorial = O(n!)



Data Structures

- Arrays/ Vectors
- Linked Lists (Focus on singly- linked)
- Stacks
- Queues
- Hash Tables (Hash Collisions)
- Trees
  - Binary tree
  - Binary search tree
  - AVL trees

Algorithms

- Brush up on the basic algorithms
  - Sorting: bubble, selection, insertion, mergesort, quicksort
  - Searching: binary search, breadth-first search, depth-first search
- Implementations
  - iteration
  - Recursion
- Algorithm analysis
  - Time and space complexity
  - Be able to list pros/ cons about different algorithms and approaches

Miscellaneous Topics

- String manipulation
- Conceptual, non-coding questions
  - Ex. You have 9 pool balls that are identical in every way, except one ball weighs more than the others. Describe an algorithm that returns the ball that weighs more.

What to do when you don’t know the answer:

- If you haven’t covered a topic in your course but are familiar with it, showcase that you know what the problem is asking
- Not all questions will click with everyone, this is expected. Just start with clarifying questions and never say “I don’t know”. Although this isn’t ideal, it’s a great chance to show you can communicate well in uncertainty.
- Don’t get caught up on syntax

IMPORTANT

- The most crucial language to use during an interview is not Java, Python, JavaScript, or C++
- It is English
- A huge part of interviews is showing that you can communicate well about a problem.
- Think out loud, start from a very basic solution and make it better.
- Ask clarifying questions
- Be prepared to explain what you're thinking and be confident. Some companies purposely ask why you aren't doing it another way (sometimes they are right, other times they are wrong) and you should be able to compare ideas. Don't give up on yours easily but also don't be stubborn

Any Questions?

Question #1

Write code to traverse a binary tree and print out the node values from left to right.

Question #1

Write code to traverse a binary tree and print out the node values from left to right.

- Which data structure would you use? Could you use a stack, a queue, or an array? Think outside of the box ☺
- How would you implement the algorithm? (Iteratively or recursively?)

Question #1 Solution and Notes

- Recursion:

```
void print_tree_recursive(Node* current){
    if(current != NULL){
        print_tree_recursive(current->left);
        std::cout << current->datum << " ";
        print_tree_recursive(current->right);
    }
    return;
}
```

- Iteratively:

```
void print_tree_iterative(Node* root){
    std::stack<Node*> s;
    Node* temp;
    s.push(root);

    while(!s.empty()){
        temp = s.top();
        s.pop();
        if(!temp->visited){
            temp->visited = true;
            if(temp->right !=  NULL){
                s.push(temp->right);
            }
            s.push(temp);
            if(temp->left !=  NULL){
                s.push(temp->left);
            }
        }
        else{
            std::cout << temp->datum << " ";
        }
    }
    return;

}
```

Question #2

You're a part of the security team for a company. Design a way to keep track of login locations (e.g. zipcode) of employers. If the location of login is different, flag it.

Assume: Logins are given in a pair<username, zipcode>, users can be flagged multiple times, keep a vector of flagged users.

Question #2

You're a part of the security team for a company. Design a way to keep track of login locations (e.g. zipcode) of employers. If the location of login is different, flag it.

- Which data structures would you use?
- Think about different approaches – which ones are better in terms of complexity and implementation?

Question #2 – Solution

- Use a hashmap of usernames -> zipcodes
- Iterate over vector of logins. For each login check our hashmap[username] to see if the stored zipcode is different than the zipcode of the login. If they are different, push the username onto our vector of flagged users.

Question #3

Find the intersection of two (singly) linked lists

What is important to realize here? What qualities can we assume?

Question #3 Solution

- Move through each linked list keeping track of the size. Find the end of each list and verify they are the same. If not, then return NULL;
- Otherwise, find the difference in the lengths and remove that number of nodes from the longer of the lists. I.e.) List A : Length = 8, List B: Length = 5. Therefore remove the first 3 elements of A.
- Now, move through both lists at the same time comparing the pointers until they are the same. Once they are, we have found the intersection.

Resources

- Cracking the Coding Interview by Gayle Laakman
- Visualgo.net
- Harvard CS50 videos
- Hack Night on Thursday! We will have some team members there to give mock interviews