# Cracking the Coding Interview

with tony comanzo
(and lots of help from the rest of the e board)

# Itinerary:

Introduction: 10 minutes

Interview practice: 40 - 70 minutes depending on turnout

Conclusion: 10 minutes

# So what is a coding interview?

# Coding Interview

- Typically conducted in-person on a whiteboard
- Sometimes it’s virtual on codewars, leetcode, google docs
- Concept: solve a coding challenge without the help of an IDE
- You might have multiple rounds with multiple interviewers

# What’s the hype?

- Very common
- They test your knowledge of algorithms and data structures
- Oftentimes the “challenges” are CS trivia or riddles
- They’re infamous

Time to code!

# Some Solutions Follow

# Test for palindrome (java)

- Have variables that represent the first and last indices of the string
- If the characters at those indices match, continue. Otherwise, return false
- Increment the first index, decrement the second until they overlap
- If the loop finishes then the string must be a palindrome
- O(n) time complexity

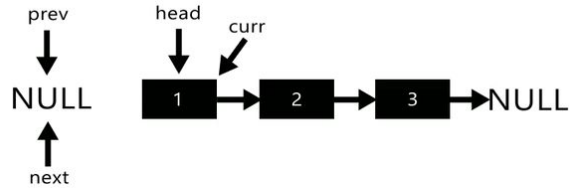
```java
// Function that returns true if
    // str is a palindrome
    static boolean isPalindrome(String str)
    {

        // Pointers pointing to the beginning
        // and the end of the string
        int i = 0, j = str.length() - 1;

        // While there are characters toc compare
        while (i < j) {

            // If there is a mismatch
            if (str.charAt(i) != str.charAt(j))
                return false;

            // Increment first pointer and
            // decrement the other
            i++;
            j--;
        }

        // Given string is a palindrome
        return true;
    }
```

# Convert to pig latin (javascript)

- Get a list of each word in the sentence
- Modify each item
  - Move first letter to the end
  - Append “ay”
  - Skip if it’s punctuation
- Make a new string
- O(n) time complexity

```
function pigIt(str){
  //Code here
  var words = str.split(" ");
  for (var i = 0; i < words.length; i++) {
    var word = words[i];
    if (word !== "!" && word !== "?" && word !== "." && word !== ",") {
      words[i] = word.substring(1) + word.charAt(0) + "ay";
    }
  }
  return words.join(" ");
}
```

# Reversing a linked list (python)




```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

```
# Function to reverse the linked list
def reverse(self):
    prev = None
    current = self.head
    while(current is not None):
        next = current.next
        current.next = prev
        prev = current
        current = next
    self.head = prev
```

- O(n) time complexity

# Removing a node from a doubly linked list (python)

- O(1) time complexity

```
# Function to delete a node in a Doubly Linked List.
# head_ref --> pointer to head node pointer.
# dele --> pointer to node to be deleted

 def deleteNode(self, dele):

     # Base Case
     if self.head is None or dele is None:
         return

     # If node to be deleted is head node
     if self.head == dele:
         self.head = dele.next

     # Change next only if node to be deleted is NOT
     # the last node
     if dele.next is not None:
         dele.next.prev = dele.prev

     # Change prev only if node to be deleted is NOT
     # the first node
     if dele.prev is not None:
         dele.prev.next = dele.next
```

# Reverse a stack without any external data structures (python)

- Trick is to use recursion because function calls will remember the item you popped
- Pop everything in stack order (last in first out)
- Insert everything you just popped
- Kicker: if there is already something on the stack, pop everything till it's empty, then push the new item, and then push everything you popped
- O(n) time complexity

```
  # Below is a recursive function
# that inserts an element
# at the bottom of a stack.
def insertAtBottom(stack, item):
    if isEmpty(stack):
        push(stack, item)
    else:
        temp = pop(stack)
        insertAtBottom(stack, item)
        push(stack, temp)

# Below is the function that
# reverses the given stack
# using insertAtBottom()
def reverse(stack):
    if not isEmpty(stack):
        temp = pop(stack)
        reverse(stack)
        insertAtBottom(stack, temp)
```

# See if there's a path between two nodes in a tree (java)

- Can be implemented with depth-first or breadth-first search
- If you have two nodes and their graph, traverse the graph starting at the first node and see if you find the second node, marking all nodes in between as "already visited" to avoid cycles and repetition
- What're the tradeoffs between breadth-first and depth-first?

```java
enum State { Unvisited, Visited, Visting; }
boolean search(Graph g, Node start, Node end) {
  if (start == end) return true;

  // operates as Queue
  LinkedList<Node> q = new LinkedList<Node>();

  for (Node u : g.getNodes()) {
    u.state = State.unvisited;
  }
  start.state = State.Visiting;
  q.add(start);
  Node u;
  while (!q.isEmpty()) {
    u = q.removeFirst(); // i.e. dequeue
    if (u != null) {
      for (Node v : u.getAdjacent()) {
        if (v.state == State.Unvisited) {
          if (v == end) {
            return true;
          }
          else {
            v.state = State.Visiting;
            q.add(v);
          }
        }
      }
      u.state = State.Visited;
    }
  }
  return false;
}
```

*taken from Cracking the Coding Interview 6th edition by Gayle Laakmann McDowell

# Check if a binary tree is balanced (java)

- A balanced tree is defined to be a tree such that the heights of the two subtrees of any node never differ by more than one.
- Naive solution: recurse through the entire tree, and for each node, compute the heights of its subtrees
- This works but it's not efficient
  - On each node, we recurse through its entire subtree
- This solution is O(NlogN) b/c each node is "touched" once per node above it

```java
int getHeight(TreeNode root) {
    if (root == null) return -1; // base case
    return Math.max(getHeight(root.left), getHeight(root.right)) + 1;
}

boolean isBalanced(TreeNode root) {
    if (root == null) return true; // base case
    int heightDiff = getHeight(root.left) - getHeight(root.right);
    if (Math.abs(heightDiff) > 1) {
        return false;
    } else {
        // recurse
        return isBalanced(root.left) && isBalanced(root.right);
    }
}
```

*taken from Cracking the Coding Interview 6th edition by Gayle Laakmann McDowell

# Check if a binary tree is balanced (cont.)

- You might notice that getHeight could actually check if the tree is balanced at the same time as it's checking heights
- This version checks the height of each subtree as we recurse down the root
- On each node, we recursively get the heights of the left and right subtrees through checkHeight
- If the subtree is balanced, then checkHeight will return the actual height of the subtree, else an "error"
- This code runs in O(N) time and O(H) space, H is the height of the tree

```java
int checkHeight(TreeNode root) {
    if (root == null) return -1;

    int leftHeight = checkHeight(root.left);
    // pass error up
    if (leftHeight == Integer.MIN_VALUE) return Integer.MIN_VALUE;

    int rightHeight = checkHeight(root.right);
    // pass error up
    if (rightHeight == Integer.MIN_VALUE) return Integer.MIN_VALUE;

    int heightDiff = leftHeight - rightHeight;
    if (Math.abs(heightDiff) > 1) {
        return Integer.MIN_VALUE; // found error -> pass it back
    } else {
        return Math.max(leftHeight, rightHeight) + 1;
    }
}

boolean isBalanced(TreeNode root) {
    return checkHeight(root) != Integer.MIN_VALUE;
}
```

# Invert a binary tree (C++)

- Recursive approach: each node is visited only once, so the time complexity is O(n), where n is the number of nodes in the tree. Because of the recursion, O(h) function calls will be put on the stack in the worst case, where h is the height of the tree
- The iterative solution is identical in terms of time and space complexity

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        if (root == NULL) {
            return NULL;
        }
        TreeNode* temp = root->left;
        root->left = invertTree(root->right);
        root->right = invertTree(temp);
        return root;
    }
};
```

# Where do we go from here?

# Coding interviews are only a piece of the puzzle

- Some companies are moving away from putting a lot of emphasis on coding interviews
- For some technical interviews you'll need to answer questions about architecture
- Now, some companies give you a take-home test or quiz you on a platform that lets you debug, like Codewars or Codepad, that let you use test cases

Do ***NOT*** discount the behavioral portion!

- Why you?
- What would you do if you were asked to work on a task but weren’t sure how to do it?
- Do you enjoy working solo or on a team?
- Tell me about a time when you failed.
- Why do you want to work for us?
- Tell me about a time you were under a lot of pressure. What was going on and how did you get through it?

# What’re your goals?

- FAANG?
- Local tech company?
- Big company?
- Startup?
- Grad school or research?
- Consulting?
- Get out of tech?
- Live off the land?
- iOS/android development?
- Web development?
- Game development?
- systems/hardware engineering?
- Data analysis?
- External validation?
- Improve society?
- Be genuinely happy?
- Do something that helps people?
- Personal growth?
- Become rich and famous?
- Pursue something you’re really passionate about?
- Be healthy?
- Learn to accept yourself?
- Software engineering?
- UI/UX design?
- Data science?

# How’re you going to realize your goals?

1. Get a degree in computer science
2. Use a local tech company as a stepping stone to gain experience
3. Work for a faang company

1. Be the first person in your family to get a college degree
2. Become financially independent
3. Meet lots of people

1. Become financially independent
2. Raise a family
3. Be the best parent possible for your children

1. Practice interview questions
2. Work on killer personal projects
3. Get a job that makes you happy

# Why is talking about goals relevant?

# Tools

Useful general resources

UCAN

Handshake

Leetcode

Codewars

LinkedIn

Indeed

Glassdoor

Hacker News: who’s hiring? (August 2020)

apply.fyi

# Interesting Advice

# More Interesting Advice

*“It is important to draw wisdom from different places. If you take it from only one place, it becomes rigid and stale.”*
*-Uncle Iroh*

# Companies only interview applicants.

*“Failure is only the opportunity to begin again. Only this time, more wisely.” -Uncle Iroh*

*“Luck is what happens when preparation meets opportunity.”*

*-Seneca, Roman philosopher*