**Tech Interview Knowledge**

# Good video links prior to finding this

## An example leetcode problem solving

### https://www.youtube.com/watch?v=vssbwPkarPQ

## Playlist of problem solving

### https://www.youtube.com/watch?v=U6-X\_QOwPcs&list=PLU\_sdQYzUj2keVENTP0a5rdykRSgg9Wp-

## His process

### <https://youtu.be/nmf-oObylnk>

## |\*.\*|SUMMARY OF HIS PROCESS|\*.\*|

## Did a TOOOOON of Pramp

### Scheduled multiples per day

### Watched MIT D&S too

#### Focus on Eric Demain

### Eventually you feel it in your brain that you can solve these problems

#### You start to categorize problems

#### At first you might not understand

##### Next day you’ll get it

#### Remember to go back and do old problems in case you forget

### He topped at 8 mock interviews per day for a month lmao

### Can do some hackerrank tutorials for python

## Doing problems in ORDER OF PRIORITY

### Binary Search

### Linked List

### Tree problems

### BFS AND DFS

### Stacks

### Do easy problems first, THEN move to medium

### Dynamic programming, heap sorting, backtracking, heaps is last

# |\*:\*||\*:\*|Video 1: Peak Finding|\*:\*| |\*:\*|

# https://courses.csail.mit.edu/6.006/fall10/lectures/lec02.pdf

# https://www.youtube.com/watch?v=HtSuA80QTyo&list=PLUl4u3cNGP61Oq3tWYp6V\_F-5jb5L2iHb

# There is a peak where given a, b ,c

## A <= b =<c

## Find it

# Normal brute force search is O(n)

# How do you shorten that

## Go binary search

## O(log n)

### Because of n/2

# As an example of binary search

## Start n/2

## If n/2-1 > n/2

### Search left half

## If n/2+1 > n/2

### Search right half

## Else, you’re at the peak

### Gj you did it

# Greedy 2d algorithm for finding ANY 2 peak

## Always look for the best local solution

## I.E. a 2d hill, where it’s the highest around

### Follow a value that’s greater than your current val no matter what

## Worst case scenario of an n\*m matrix

### O(n\*m), or O(n\*\*2) if m = n

### You could loop all the way around and find it at the very end

# Cannot just use O(log n) 1d search to solve for 2d

## The 2d peak may not exist in the row you just performed the 1d search in

# For i and j for N and M dimensions, row column

## Picking the starting with middle column, find the maximum 1d value in that column

### j = m/2

### O(n)

## But comparing (I, j-1) aka left, (I, j) aka middle, (I, j+1) aka right

### Pick left if (I, j-1) > (I,J)

### Pick right if (I,j+2) > (I,J)

### Otherwise you have found your 2d peak

## If you go left/right

### You solve with half the number of potential columns

## Big O

### T(N,M) = T(n, m/2) + O(n)

#### Recursively scanning for half of the columns, each time times n at worst

### = log2m(O(n)

### = nlog2(m)

### = n log (m)

# |\*.\*|Video 2: Models of Computation, Document Distance|\*.\*|

# Algorithm

## Takes an input, runs through the algo, returns the output

# Model of computation

## What operations of an alg is allowed

### Also cost, time, of each operation

# Random Access Machine functionality

## It’s a giant array, represented by RAM (random access memory) chip in your computer

## In O(1), you can

### Load

### Compute

### Store

### All in O(1)

# Pointer Machine

## Dynamically allocated objects

## An object has a constant number of fields

## The fields = word/pointer

## Pointer points/Reference

### To another object

### Or Null

# Pointer -> Linked list

## Prev/next pointers

# In Python, what’s going on (in terms of time)

## Not all operations are the same

## Some are exponential, some are costant

# Python Operation Runtimes

## List

### Aka an Array

## Concatentating to a list = O(1)

### L[i] = L[j] + 5

## Setting a next attribute in a list = O(1)

### x = x.next

## List append = O(1)

### Python does table doubling

### l.append(x)

## Concatenating two lists = O(n)

### N+m

## For x in L = O(n)

### Could scour the entire list

## Length of list O(1)

### Length of list is stored at the beginning

## Sorting a list = O(n log n)

### Python uses a comparison sort

## Dictionaries/Hashmaps, most operations EXCEPT UPDATE are O(1)

## Long int

### X+y = O(x+y)

### X\*y = O(x+y)\*\*log3 or 1.6, which is better than O(x+y)\*\*2

## Heapq

# Problem: Document Distance

## Given 2 docs, d1, d2, where a doc is a sequence of words

### And a word is a string

## What is the diff between 2 docs

### Are they cheating?

## They’re similar IF they have a lot of shared words

## Imagine that you have a starting dictionary with words, and their value equals the frequency in which the word appears

## Or if they’re vectors, you can think of it as Vector Calculus

### If the angle between the two vectors is like 90 degrees, they’re very different

## So you’ll want to

### Divide the doc into words

### Calc the freq of those said words

### Find the dot product the resulting two vector arrays/dicts

## Divide the doc into words using a dictionary

### For word in doc

#### Dict[word] +=1

### O(n)

# Leaving off from BST

## <https://www.youtube.com/watch?v=9Jry5-82I68&list=PLUl4u3cNGP61Oq3tWYp6V_F-5jb5L2iHb&index=5>

# BST search trees have some cool traits

## Versus a heap

### A heap is an array that can be visualized as a tree

### But a tree actually has pointers

## Pointers

### Parent

### Left

### Right

## If you’re in the left subtree

### Your values are less than those of the right subtree

### And vice versa

## It’s essentially a sorted list + sorted array

## During the insertion phase

### You can choose to make checks

### THIS IS THE MOST FLEXIBLE THING OF A BST

# Insertion

## O(h) where h is the height of the tree

# Find Min

## Keep going to the left

## O(h)

# Find Max

## Keep going to the right

## O(h)

# Augmented Binary Search Trees

## You can put more than one value in

## I.e. you can add how many children are underneath

# Toy Problem, find all values <= x

## Assuming you have an augmented BST with size associated with it

## Every time you look arrive at a new node

### +1

## If you see something is higher and you go to it

### +children in left node

#### Will cover everything in remaining left branch

## When you finally arrive at your node, do one last check

### +children in left node as necessary

## Or keep going until you reach double null while still being less than your desired endpoint

# |\*.\*||\*.\*|BFS|\*.\*||\*.\*|

# |\*.\*||\*.\*|DFS|\*.\*||\*.\*|

## HARD PROBLEM

### https://leetcode.com/problems/critical-connections-in-a-network/discuss/494896/Python-DFS

## Edges

### In an Undirected graph

#### Each edge is visited twice

### In a Directed graph

#### Each edge visited once

### Tree edge

#### When you visit a NEW vertex via an edge

#### Because they form a directed tree

### Back edge

#### Directed or reaches an ancestor

## Back edge

### There’s a tree edge