Technical Interviews

Tips and Tricks for Succeeding at Technical Interviews

Cody Watson, Assistant Professor of Computer Science

What to be prepared for

- Companies conducting interviews for a technical position are looking for two main components
  - Strong technical skills
    - Critical thinking
    - Coding
    - Numerical analysis
    - Making sense of numbers
  - Strong communication skills
    - Ability to communicate problems
    - Ability to communicate solutions
    - Ability to work in a group setting

Numbers

- Be prepared and comfortable to work in a numerical setting
- Remember the bottom line
- Saving time and resources through meaningful software
- Be comfortable talking metrics

Critical thinking

- Ability to critically think
- Ability to communicate
- Talk through your process,
  - data structures
  - algorithmic process
  - resources taken
- Ask yourself questions about edge cases and about efficiency of your code.

Interviews are like boxing

- You will have multiple rounds
- A bad round, its ok you can bounce back
- Preconceived notions of other interviewees and interviewers.

Keep these in mind

- DEEP BREATH, your nerves will be on edge.
- Do not force a solution, talk through outlines and processes
- Do not freak out if you get stuck
  - New perspective
  - 30 second walk
  - Write on a piece of scratch paper

Run time, time complexity, efficiency

- Run time measures the wall time needed to run the program
  - Empirically determined
  - Not theoretical
- Time complexity speaks to algorithms and uses Big O
  - Theoretical
  - Speaks to the quality of the algorithm
  - Wasting cpu cycles
- Efficiency can consider time complexity and memory used
  - Extra variables
  - Extra data structures
  - Time complexity
  - Run time

Run time in python

```
import time

start_time = time.time()
main()
print("---- %s seconds ----" % (time.time() - start_time))
```

Time complexity

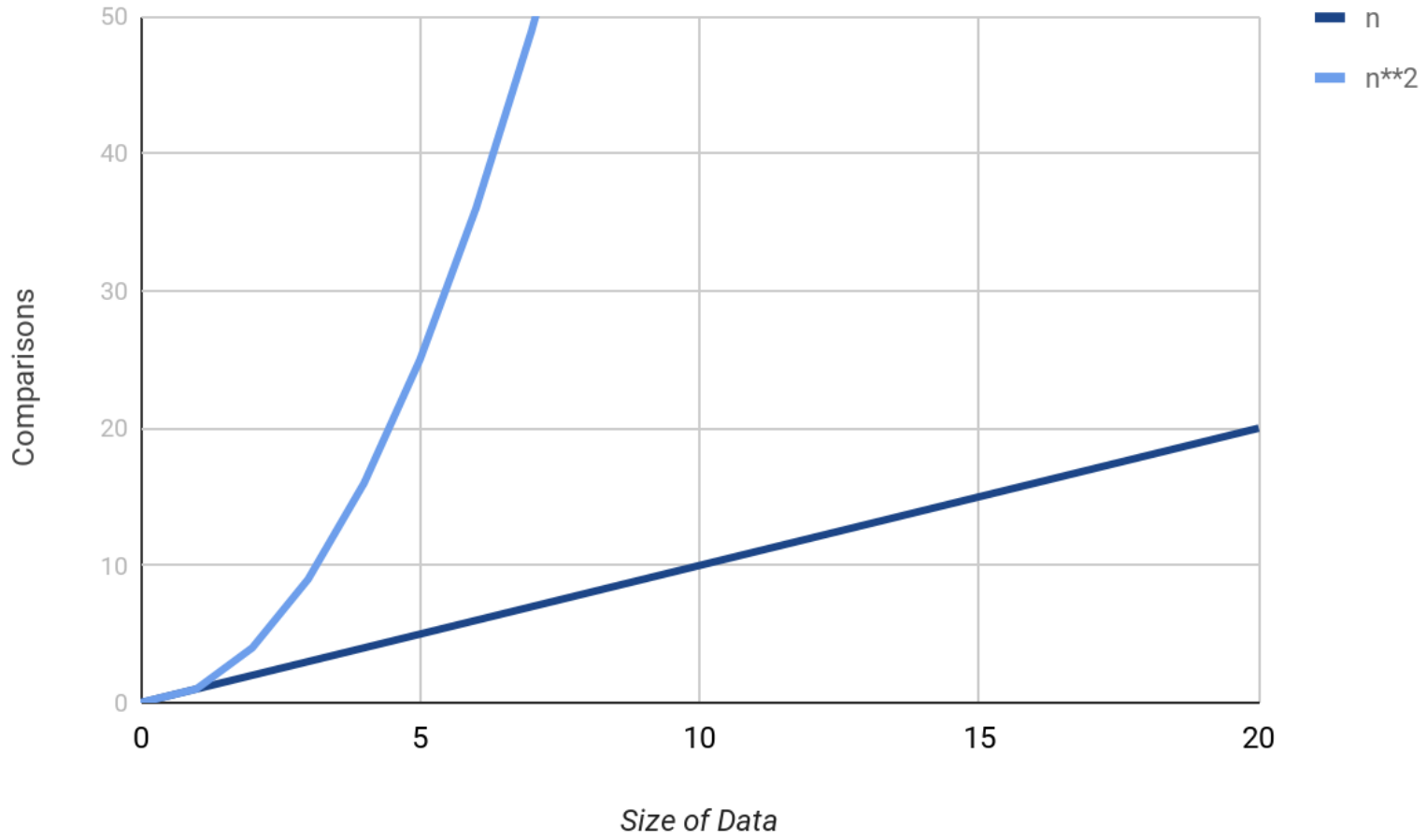
- Big-O - worse case scenario for an algorithm
  - Looks at order of magnitude

- Constant complexity: O(1)
- Logarithmic complexity: O(log n)
- Linear complexity: O(n)
- Quadratic Complexity: O(n^2)
- Cubic complexity: O(n^3)
- Exponential: O(2^n)

Time complexity

- Theta instead of Big-O
  - A tight bound
  - Average scenario
  - Θ
- We don’t frequently use bound
- Want to account for worst case scenario

Graph of O(n) vs O(n^2)



Simple way to identify complexity

- Nested for loops are almost always O(n^2)
- Loop within a loop within a loop is usually O(n^3)
- Try to break problems into instructions
- See how many times those instructions are executed
- Double input size -> resulting increase in operations

Constant complexity O(1)

```
def print_element(lst, index):
    print("This is element %d, %d\n", index, lst[i])
```

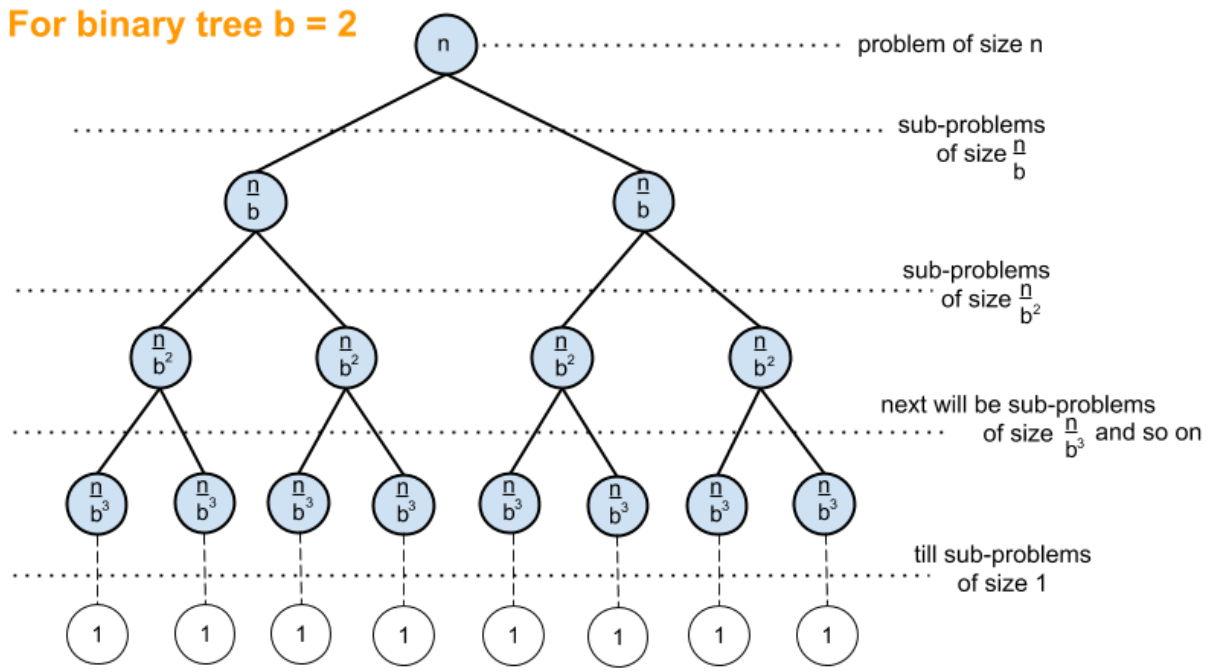
Linear Complexity O(n)

```
def print_list(lst):
    for i in range(len(lst)):
        print("This is element %d, %d\n", i, lst[i])
```

Logarithmic complexity

```
def binary_search(lst, start, end, element):
    if end >= 1:
        midpoint = 1 + (end - 1) // 2
        if lst[midpoint] == element:
            return midpoint
        elif lst[midpoint] > x:
            return binarySearch(lst, start, midpoint - 1, element)
        else
            return binary_search(lst, midpoint + 1, end, element)
    else:
        return 0
```

Logarithmic complexity



Quadratic complexity

```
def compare_duplicates_quadratic(lst):
    duplicates = []
    for i in range(len(lst)):
        for j in range(len(lst)):
            if i != j and lst[i] == lst[j] and lst[i] not in duplicates:
                duplicates.append(lst[i])
    return duplicates
```

Exponential complexity

```
def fibonacci_exponential(n):
    if n <= 1:
        return n
    else:
        return fibonacci_exponential(n - 2) + fibonacci_exponential(n - 1)
```

You try

```
def sort_example(lst):
    for i in range(len(lst) - 1):
        for j in range(len(lst) - i - 1):
            if lst[j] > lst[j+1]:
                temp = lst[j]
                lst[j] = lst[j+1]
                lst[j+1] = temp
    return lst
```

- Best case? Worst case? Average case?

Questions?

Leet Code Examples

Problem

- Given an array of integers, find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers.

Brute Force $O(n^2)$

- Most simple solution
- Will likely come to you first
- What do you need
  - A list
  - A variable to hold the starting value
  - A variable to hold the ending value
  - A variable to hold the length

Algorithm

```
def longestSequence(lst):
    lst.sort()
    longest_run = 0
    for i in range(len(lst)):
        start = lst[i]
        end = start
        run = 0
        for j in range(i,len(lst)):
            if lst[j] == end + 1:
                run += 1
                end = lst[j]
            if run > longest_run:
                longest_run = run
    return longest_run + 1
```

O(n) Method

- Only traverse the list a single time
- Need to make use of a different idea
- 2 key changes from brute force
  - We save everything in a set (unique values only)
  - We only build the longest sequence out of numbers that are not already a part of our solution
- Key idea: ?

Algorithm

```
def longestSequence_fast(lst):
    longest_run = 0
    lst_set = set(lst)
    for element in lst:
        #THIS IS THE KEY
        if element - 1 not in lst_set:
            current_element = element
            current_run = 1
            while current_element + 1 in lst_set:
                current_element += 1
                current_run += 1
            longest_run = max(longest_run, current_run)
    return longest_run
```

Problem 2

- Given an array nums of n integers where n > 1, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].

Simple solution of brute force

- Iterative through the list
- Figure out which element not to include
- Append to list
- We need a list, a variable to hold the calculation for multiplication, an empty list to append the values

Algorithm

```
def getProductList(lst):
    productList = []
    ans = 1
    for i in range(len(lst)):
        for j in range(len(lst)):
            if j != i:
                ans *= lst[j]
        productList.append(ans)
        ans = 1
    return productList
```

Clever solution, use division

```
import functools
def getProductListDivision(lst):
    productList = []
    result = functools.reduce((lambda x, y: x*y), lst)
    for i in range(len(lst)):
        productList.append(result / lst[i])
    return productList
```

Best solution O(n)

- Split the problem into two separate problems
  - All the numbers on one side of the index
  - The remaining numbers on the other side

- Multiplying the two pieces will give you multiplication of all elements other than the current index

Algorithm

```
def getProductList_fast(lst):
        length = len(lst)
        Left, Right, answer = [0]*length, [0]*length, [0]*length
        Left[0] = 1
        for i in range(1, length):
            Left[i] = lst[i - 1] * Left[i - 1]
        Right[length - 1] = 1
        for i in reversed(range(length - 1)):
            Right[i] = lst[i + 1] * Right[i + 1]
        for i in range(length):
            answer[i] = Left[i] * Right[i]
        return answer
```