2. Strategies for a Great Interview

Approaching the problem

- clarify the question
- work on concrete examples
- spell out the brute force solution
- apply patterns

Presenting the solution

- use libraries if possible
- focus on the top-level algorithm
- manage the whiteboard
- assume valid inputs
- test for corner cases
- check syntax
- check memory management (especially in C and C++)
- remember the interviewer cannot process a large amount of code
    - keep your solution simple if possible

Know your interviewers and the company

General conversation

- can the candidate clearly communicate a complex idea?
- is the candidate passionate about his/her work?
- is there a potential interest match with some project?

Other advice

- be honest
- keep a positive spirit
- don't apologize
- keep money and perks out of the interview
- appearance
- be aware of your body language

Stress interviews

- may inject stress into situation to test candidate (especially in finance industry)
- testing to see how interviewee responds
  - do not fall apart
  - do not allow yourself to become belligerent
  - do not allow yourself to be easily swayed

Learning from bad outcomes

Negotiating an offer

- stick to email
- try to speak directly with hiring manager
- remember that long term career is more important than compensation

3. Conducting an Interview

Objective

- determine if candidate will be a successful employee
- do not be indecisive about candidate - this wastes interview
- turn candidate into brand ambassador for recruiting
- make interviewee feel positive about experience and company

What to ask

- how much training time does the work environment allow?
- for startups
  - can make sense to test on domain specific knowledge
- for larger organizations
  - data structures
  - algorithms
  - system design
  - problem solving techniques
- remember to choose problems that
  - have no single point of failure - do not ask a single question based around a single insight
  - have multiple possible solutions
  - cover multiple areas

    - calibrate on the problems that will be covered by colleagues
    - do not require unnecessary domain knowledge

Conducting the interview

- prepare hints beforehand
- watch for
    - a candidate that get stuck
    - a verbose candidate
    - an overconfident candidate

Scoring and reporting

4. Problem Solving

Data Structure Review

- particular method for storing or organizing data that allows for efficient manipulation (read, write, modify)
- primitive types
    - int
    - char
    - double
    - know representation in memory and operations on them
- arrays
    - constant access by index
    - slow lookups (unless sorted)
    - slow insertions
    - know
        - iteration
        - resizing
        - merging
        - partitioning
- strings
    - know
        - representation in memory
        - comparison
        - copy
        - matching
        - joining

        - splitting
- lists
    - know (singly and doubly-linked)
        - trade-offs with respect to arrays
        - iteration
        - insertion
        - deletion
        - dynamic allocation (e.g. new with nodes)
        - arrays
- stacks and queues
    - know
        - LIFO semantics and usage
        - FIFO semantics and usage
        - array implementation
        - linked list implementation
- binary trees
    - used to represent hierarchical data
    - know
        - depth
        - height
        - leaves
        - search path
        - traversal sequences
        - successor/predecessor operations
- heaps
    - benefits
        - O(1) lookup of max
        - log n insertion
        - log n deletion of max
    - know
        - node implementation
        - array implementation
        - min-heap variant
- hash tables
    - benefits
        - O(1) insertion
        - O(1) deletion
        - O(1) lookup
    - disadvantages
        - bad for order-related queries
        - resizing (consider amortized overhead)

        - poor worst-case performance
    - know
        - separate chaining
        - linear probing
        - hash functions
            - integers
            - strings
            - objects
- binary search trees
    - benefits
        - when height balanced
            - log n insertions
            - log n deletions
            - log n lookup
            - log n find min
            - log n find max
            - log n successor
            - log n predecessor
    - know
        - various node fields
        - pointer implementation
        - balance
        - operations to maintain balance

Primitive Data Types

- types
    - char
    - int
    - double
    - unsigned and long variant
- types differ among languages
    - Java has no unsigned int
    - int width is machine and compiler dependent in C
- example problems
    - find set bits in int
        - use x & ~(x - 1)
    - right propagate the rightmost set bit
    - x modulo a power of 2
    - check if power of 2
- remember to use lookup tables of nibbles

Arrays

- maps indexes in range 0 to n - 1 to objects of a given type (n is number of objects in array)
- lookup and replacement is constant time
- many common errors exist
    - bounds/fence errors
    - sizeof errors
- example problems
    - partitioning (related to quicksort)
        - maintain two regions on either end of array and expand one element at a time

Strings

- generally a character array
- many common operations are generally not used for arrays
    - comparison
    - joining
    - splitting
    - substring search
    - replacement
    - parsing
- example problems
    - look-and-say

Lists

- collection of values (implicitly ordered by relationship between elements) which may include repetitions
- generally collection of nodes
    - singly-linked
    - doubly-linked
- often used as components of larger data structures
- example problems
    - zip of a list

Stacks and queues

- stack - LIFO
- queue - FIFO
- example problems
    - reverse polish notation
    - parsing
    - order of evaluation

Binary trees

- search trees - keys in a sorted order via hierarchical relationship
- commonly node-based
- example problems
  - process/system resource locking

Heaps (aka priority queue)

- based on binary tree
- each element has a priority associated with it and delete removes element with highest priority
- min and max variants
- example problems
  - merging separate stock trade transactions sorted by timestamp
    - trivial when working with heaps

Hash tables

- used for storing keys (and optionally values)
- O(1) (on average)
  - inserts
  - deletes
  - lookups
- requires
  - a good hash function
  - resizing in certain cases
    - may result in O(n) updates
- example problems
  - check if string can be permuted into palindrome
    - if and only if characters occur only twice with an optional single character
  - check for plagiarism
    - suffix arrays in hash table form

Binary search trees

- used to store comparable objects
  - nodes satisfy BST property, e.g. left children are null or less than, right children are null or greater than
- log n
  - lookup
  - insertion
  - deletion
- supported by balanced types (AVL and red-black)

Algorithm patterns

- analysis patterns
    - concrete examples
        - manually solve concrete instances of the problem and then build a general solution
    - case-analysis
        - split the input/execution into a number of cases and solve each case in isolation
    - iterative refinement
        - find brute force solution and improve upon it
    - reduction
        - use a well-known solution to some other related problem as a subroutine
    - graph modeling
        - describe the problem as a graph and use graph algorithm to solve
- algorithm design patterns
    - sorting
        - uncover structure of problem by sorting input
    - recursion
        - if input is structured in recursive pattern, design recursive algorithm to follow
    - divide-and-conquer
        - divide into two or more independent subproblems and solve original using subproblem solutions
    - dynamic programming
        - compute and cache solutions to smaller subproblems and solve original using subproblem solutions
    - greedy algorithms
        - compute in stages making choices at each stage that are locally optimal
    - invariants
        - identify invariant and use it to remove potential solutions that are suboptimal or contained within other solutions

Intractability

- some problems may not have an efficient solution
- must prove intractability
- NP complete
    - must be NP and NP-hard
    - NP
        - nondeterministic polynomial time
    - NP-hard
        - at least as hard as the hardest problems in NP

- a problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H
  - assuming a solution for H takes 1 unit time, we can use H's solution to solve L in polynomial time
- finding a polynomial algorithm to solve any NP-hard problem would give polynomial algorithms for all the problems in NP, which is unlikely as many of them are considered hard