Direct onsite because I interviewed in the past and did well that time. From the time I sent my resume to interview day: 2 weeks. From interview day to offer over the phone: 2 weeks. The syllabus for the interviews is very clear and simple: 1) Dynamic Programming 2) Super recursion (permutation, combination,...2^n, m^n, n!...etc. type of program. (NP hard, NP programs) 3) Probability related programs 4) Graphs: BFS/DFS are usually enough 5) All basic data structures from Arrays/Lists to circular queues, BSTs, Hash tables, B-Trees, and Red-Black trees, and all basic algorithms like sorting, binary search, median,... 6) Problem solving ability at a level similar to TopCoder Division 1, 250 points. If you can consistently solve these, then you are almost sure to get in with 2-weeks brush up. 7) Review all old interview questions in Glassdoor to get a feel. If you can solve 95% of them at home (including coding them up quickly and testing them out in a debugger + editor setup), you are in good shape. 8) Practice coding--write often and write a lot. If you can think of a solution, you should be able to code it easily...without much thought. 9) Very good to have for design interview: distributed systems knowledge and practical experience. 10) Good understanding of basic discrete math, computer architecture, basic math. 11) Coursera courses and assignments give a lot of what you need to know. 12) Note that all the above except the first 2 are useful in "real life" programming too! Interview 1: Graph related question and super recursion Interview 2: Design discussion involving a distributed system with writes/reads going on at different sites in parallel. Interview 3: Array and Tree related questions Interview 4: Designing a simple class to do something. Not hard, but not easy either. You need to know basic data structures very well to consider different designs and trade-offs. Interview 5: Dynamic programming, Computer architecture and low level perf. enhancement question which requires knowledge of Trees, binary search, etc. At the end, I wasn't tired and rather enjoyed the discussions. I think the key was long term preparation and time spent doing topcoder for several years (on and off as I enjoy solving the problems). Conclusion: "It's not the best who win the race; it's the best prepared who win it."

Show Less

Interview Questions

. Tow questions: 1) A string consists of ‘0’, ‘1’ and '?'. The question mark can be either '0' or '1'. Find all possible combinations for a string. 2) Give you a text file, remove duplicated lines. Follow up: If the file is very large, general hash map takes too much spaces, come up with a better solution

Research:

* https://medium.com/this-happened-to-me/four-steps-to-google-without-a-degree-8f381aa6bd5e
* design a Load Balancer
* Problem solving ability at a level similar to TopCoder Division 1, 250 points
* Review all old interview questions in Glassdoor to get a feel. If you can solve 95% of them at home (including coding them up quickly and testing them out in a debugger + editor setup), you are in good shape
* Design discussion involving a distributed system with writes/reads going on at different sites in parallel
* ;why are manhole covers round&#034; and &#034;
* if you and a friend were on the same flight what are the odds you&#039;d be sitting next to each other?
* What data structure would you use to represent a scenario where a person has friends and each friend has friends? (ANSWER: An Undirected Graph (~~why??~~ edit: I know why ☺))
* Given an array of integers, find the length of the shortest subsequence that sums up to over a certain threshold. ([ANSWER](#_Find_Shortest_Subsequence))
* Matrix - given a nxn matrix, print the matrix in diagonal from left to right.
* study leetcode
* Write a function to add a set {x, y} to a set of disjoint sets in a certain pattern.
* how to handle eventhandler in Actionscript
* Given a string which only contains lower case you need to delete only the repeated letters only leave one the resulting string should lexicography as smallest. i.e. bcabc delete the first b and first c
* Make sure you go in there knowing how the work and experience you have done could apply to Google
* but typical linked list, array, and string manipulations involving sorting and searching, and one question focusing on computer architecture. Asked to optimize for time and/or storage and to discuss the trade offs.
* Print every tree paths from root to leaf. Just use recursion way to do that. And the second one is to print good number.
* Asked about time and space complexity about tree paths question.
* Asked about algorithm design, classifier optimization, background, system design, DP etc. All interviewers were helpful and friendly. Algorithms, optimization of classifier, recursion, DP
* For the algo questions - just know your data structures and basic algorithms, practice solving problems for a few weeks and you should do just fine. System design questions focus on designs Google-scale distributed systems. I suggest reading up on MapReduce and similar things. Study or practice with modern NoSQL distributed storage engines, know where usual bottlenecks are and you should do fine. Also read up on the materials that Google recruiters recommend - they are relevant. The "anti-loop" thing can be real and you can fail the interview even if you are well qualified.
* The questions were what you expect of a standard programming interviews (DP, trees and other data structures). Make sure you have extensively reviewed algorithms before you attempt the interview.
* What products would you want to work on? I wasn&#039;t expecting this and just babbled off a list of all of their products instead of picking out the ones that most interested me.
* Return the max k numbers from an unsorted integer array. Each number in the array is in the range [0, 10000). ([ANSWER](#_Find_Max_Numbers))
* You are given a set of unsorted strings to work with. Construct a program that will suggest autocorrect words as they are typed. (the unsorted strings are the valid words).
* Wasn't sure how to go about it at first, so I talked through my thought process until I came up with a solution. He asked about the running time and what kind of tests I would write for it.
* how I could relate my project with google?
* Given a string &#039;aaabb&#039;, can you rearrange the string such that no 2 similar letters are adjacent to each other.
* Purely algorithmic. 5 interviews, 45 minutes each and a lunch interview. The first problem was a graph search problem solvable using DFS and knowing about disjoint sets. Second problem was a linked list problem. Third problem was a data structure problem. Fourth problem was a string problem. Fifth problem was geometry/area/collision detection.
* Design a locking mechanism for a distributed system
* Given a young tableaux. Figure out how to sort elements.
* Given that you have a with an even number of points, how do you find two points that equally subdivide the graph
* Given an integer array shuffle the elements in the array such that no two elements are in same place
* most questions required the use of a hashmap and knowing how to compare strings, substrings, and permuted strings. I implemented a flood fill to answer one of the q&#039;s. Changing bases is a good question to review.
* Given a list of ranges such as: [(6, 10), (-2, 4), (3, 5)], find the range with the highest number of overlaps. In this example, that would be (3, 4) which has 2 ranges overlapping.
* VA questions (http://www.va.gov/PBI/Questions.asp)
  + Tell me about two suggestions you have made to your supervisor in the past year. How did you come up with the ideas? What happened? How do you feel about the way things went?" (For creative thinking)
  + "In the past, how have you obtained and incorporated customer feedback into your organization's planning and service standards? Give specific examples." (For customer service)
  + "Describe a situation where you were responsible for getting others to make a change. What role did you play and what actions did you take? What was the outcome? If you had to do it again, would you do anything differently?" (For adaptability)
  + "How does the work you are currently doing affect your organization's ability to meet its mission and goals? Do you think your work is important? If yes, why? If no, why not?" (For systems thinking)
* Dynamic Programming
* Super Recursion
  + permutation,, combination,...2^n, m^n, n!...etc. type of program.
    - (NP hard, NP programs)
* Probability related programs
* Datastructures
  + circular queues
  + BSTs
  + Hash tables
  + Sets (Disjoined Sets)
  + Trees
    - B-Trees
    - Red-Black trees,
    - Trie (aka radix tree or prefix tree)
  + Graphs
    - BFS
    - DFS
    - Undirected Graph
* basic algorithms
  + sorting, binary search, median
  + manipulate/validate data using tree manipulation

ADD ME:

* Adjacency\_matrix
* double hashiing
* Dijkstra and A\*
* NP-complete problems, such as traveling salesman and the knapsack problem
  + You should find out what NP-complete means
* know how to sort. Don't do bubble-sort. You should know the details of at least one n\*log(n) sorting algorithm, preferably two (say, quicksort and merge sort). Merge sort can be highly useful in situations where quicksort is impractical, so take a look at it.
* red/black tree, a splay tree or an AVL tree
* combinatorics and probability. You should be familiar with n-choose-k problems and their ilk
* Know about locks and mutexes and semaphores and monitors and how they work. Know about deadlock and livelock and how to avoid them. Know what resources a processes needs, and a thread needs, and how context switching works, and how it's initiated by the operating system and underlying hardware. Know a little about scheduling. The world is rapidly moving towards multi-core, and you'll be a dinosaur in a real hurry if you don't understand the fundamentals of "modern" (which is to say, "kinda broken") concurrency constructs.
* servlet life cycle is
* dynamic programming
* design a concurrent queuing system
* map reduce
* know how the web works down to the level of IP packets.

## Find Shortest Subsequence

1. **function** findSubSum(nums, targetVal) {
2. **var** subNums = **null**;
4. **while** (nums.length && nums.reduce(sum) >= targetVal) {
5. subNums = nums
6. nums = (nums[0] > nums[nums.length - 1]) ? nums.slice(0, -1) : nums.slice(1);
7. }
8. **return** subNums;
10. **function** sum(a, b) {**return** a + b;}
11. }

## Find Max Numbers

1. **function** getMaxes(numArray, maxCount) {
2. **var** maxes = numArray.slice(0, maxCount).sort(**function**(a, b) {**return** a - b;});
3. **for** (**var** i = maxCount; i < numArray.length; i++) {
4. **var** j = -1;
5. **while** (j < numArray.length && maxes[j+1] <= numArray[i]) {j++;}
6. **if** (j !== -1) {
7. maxes = maxes.slice(1);
8. maxes.splice(j, 0, numArray[i]);
9. }
10. }
11. **return** maxes;
12. }