Hacking a Google Interview – Handout 1

Course Description

Instructors: Bill Jacobs and Curtis Fonger
Time: January 12 – 15, 5:00 – 6:30 PM in 32-124
Website: http://courses.csail.mit.edu/iap/interview

Classic Question #1: Coin Puzzle

You have 8 coins which are all the same weight, except for one which is slightly heavier than the others (you don't know which coin is heavier). You also have an old-style balance, which allows you to weigh two piles of coins to see which one is heavier (or if they are of equal weight). What is the fewest number of weighings that you can make which will tell you which coin is the heavier one?

Good answer: Weigh 3 coins against 3 coins. If one of the groups is heavier, weigh one of its coins against another one of its coins; this allows you to identify the heavy coin. If the two groups balance, weigh the two leftover coins.

Not-so-good answer: Weigh 4 coins against 4 coins. Discard the lighter coins, and weigh 2 coins against 2 coins. Discard the lighter coins and weigh the remaining two coins.

Interview tips

When asked a question, open a dialog with the interviewer. Let them know what you are thinking. You might, for example, suggest a slow or partial solution (let them know that the solution is not ideal), mention some observations about the problem, or say any ideas you have that might lead to a solution. Often, interviewers will give hints if you appear to be stuck.

Often, you will be asked to write a program during an interview. For some reason, interviewers usually have people write programs on a blackboard or on a sheet of paper rather than on a computer. It is good to get practice with writing code on the board in order to be prepared for this.

Here is a list of "do's" and "don't's" when doing a programming interview:

Do's

- Ask for clarification on a problem if you didn't understand something or if there is any ambiguity

- Let the interviewer know what you are thinking
- Suggest multiple approaches to the problem
- Bounce ideas off the interviewer (such as ideas for data structures or algorithms)
- If you get stuck, don't be afraid to let them know and politely ask for a hint

Don't's

- Never give up! This says nothing good about your problem solving skills.
- Don't just sit in silence while thinking. The interviewer has limited time to find out as much as possible about you, and not talking with them tells them nothing, except that you can sit there silently.
- If you already know the answer, don't just blurt it out! They will suspect that you already knew the answer and didn't tell them you've seen the question before. At least pretend to be thinking though the problem before you give the answer!

Big O Notation

Big O notation is a way that programmers use to determine how the running speed of an algorithm is affected as the input size is increased. We say that an algorithm is O(n) if increasing the input size results in a linear increase in running time. For example, if we have an algorithm that takes an array of integers and increments each integer by 1, that algorithm will take twice as long to run on an array of size 200 than on an array of size 100.

Now let's look at an algorithm of running time O(n^2). Consider the following Java code:

```
boolean hasDuplicate(int[] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array.length; j++) {
            if (array[i] == array[j] && i != j) {
                return true;
            }
        }
    }
    return false;
}
```

This algorithm takes in an array of integers and compares each integer to every other integer, returning true if two integers are equal, otherwise returning false. This array takes O(n^2) running time because each element has to be compared with n elements (where n is the length of the array). Therefore, if we double the input size, we quadruple the running time.

There is also a more formal definition of big O notation, but we prefer the intuitive approach for the purposes of programming interviews.

Question: Searching through an array

Given a sorted array of integers, how can you find the location of a particular integer x?

Good answer: Use binary search. Compare the number in the middle of the array with x. If it is equal, we are done. If the number is greater, we know to look in the second half of the array. If it is smaller, we know to look in the first half. We can repeat the search on the appropriate half of the array by comparing the middle element of that array with x, once again narrowing our search by a factor of 2. We repeat this process until we find x. This algorithm takes O(log n) time.

Not-so-good answer: Go through each number in order and compare it to x. This algorithm takes O(n) time.

Parallelism

Threads and processes:

A computer will often appear to be doing many things simultaneously, such as checking for new e-mail messages, saving a Word document, and loading a website. Each program is a separate "process". Each process has one or more "threads." If a process has several threads, they appear to run simultaneously. For example, an e-mail client may have one thread that checks for new e-mail messages and one thread for the GUI so that it can show a button being pressed. In fact, only one thread is being run at any given time. The processor switches between threads so quickly that they appear to be running simultaneously.

Multiple threads in a single process have access to the same memory. By contrast, multiple processes have separate regions of memory and can only communicate by special mechanisms. The processor loads and saves a separate set of registers for each thread.

Remember, each process has one or more threads, and the processor switches between threads.

Mutexes and semaphores:

A mutex is like a lock. Mutexes are used in parallel programming to ensure that only one thread can access a shared resource at a time. For example, say one thread is modifying an array. When it has gotten halfway through the array, the processor switches to another thread. If we were not using mutexes, the thread might try to modify the array as well, which is probably not what we want.

To prevent this, we could use a mutex. Conceptually, a mutex is an integer that starts at 1. Whenever a thread needs to alter the array, it "locks" the mutex. This causes the thread to wait until the number is positive and then decreases it by one. When the thread is done modifying the array, it "unlocks" the mutex, causing the number to increase by 1. If we are sure to lock the mutex before modifying the array and to unlock it when we are done, then we know that no two threads will modify the array at the same time.

Semaphores are more general than mutexes. They differ only in that a semaphore's integer may start at a number greater than 1. The number at which a semaphore starts is the number of threads that may access the resource at once. Semaphores support "wait" and "signal" operations, which are analogous to the "lock" and "unlock" operations of mutexes.

Synchronized methods (in Java):

Another favorite question of interviewers is, "What is a synchronized method in Java?" Each object in Java has its own mutex. Whenever a synchronized method is called, the mutex is locked. When the method is finished, the mutex is unlocked. This ensures that only one synchronized method is called at a time on a given object.

Deadlock:

Deadlock is a problem that sometimes arises in parallel programming. It is typified by the following, which is supposedly a law that came before the Kansas legislature:

"When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone."

Strange as this sounds, a similar situation can occur when using mutexes. Say we have two threads running the following code:

Thread 1:

```
acquire(lock1);
acquire(lock2);
[do stuff]
release(lock1);
release(lock2);
```

Thread 2:

```
acquire(lock2);
acquire(lock1);
[do stuff]
```

release(lock2);
release(lock1);

Suppose that thread 1 is executed to just after the first statement. Then, the processor switches to thread 2 and executes both statements. Then, the processor switches back to thread 1 and executes the second statement. In this situation, thread 1 will be waiting for thread 2 to release lock1, and thread 2 will be waiting for thread 1 to release lock2. Both threads will be stuck indefinitely. This is called deadlock.

Classic Question #2: Preventing Deadlock

How can we ensure that deadlock does not occur?

Answer: There are many possible answers to this problem, but the answer the interviewer will be looking for is this: we can prevent deadlock if we assign an order to our locks and require that locks always be acquired in order. For example, if a thread needs to acquire locks 1, 5, and 2, it must acquire lock 1, followed by lock 2, followed by lock 5. That way we prevent one thread trying to acquire lock 1 then lock 2, and another thread trying to acquire lock 2 then lock 1, which could cause deadlock. (Note that this approach is not used very often in practice.)

Some Other Topics

What is polymorphism?

Interviewers love to ask people this question point-blank, and there are several possible answers. For a full discussion of all the types of polymorphism, we recommend looking at its Wikipedia page. However, we believe that a good answer to this question is that polymorphism is the ability of one method to have different behavior depending on the type of object it is being called on or the type of object being passed as a parameter. For example, if we defined our own "MyInteger" class and wanted to define an "add" method for it (to add that integer with another number), we would want the following code to work:

```
MyInteger int1 = new MyInteger(5);
MyInteger int2 = new MyInteger(7);
MyFloat float1 = new MyFloat(3.14);
MyDouble doub1 = new MyDouble(2.71);
print(int1.add(int2));
print(int1.add(float1));
print(int1.add(doub1));
```

In this example, calling "add" will result in different behavior depending on the type of the input.

What is a virtual function/method? (in C++)

Out of all the possible questions interviewers could ask about C++, this one seems to be a strange favorite. A method's being "virtual" simply describes its behavior when working with superclasses and subclasses. Assume class B is a subclass of class A. Also assume both classes A and B have a method "bar()". Let's say we have the following code in C++:

```
A *foo = new B();
foo->bar();
```

If the method "bar()" is declared to be virtual, then when we call foo->bar(), the method found in class B will be run. This is how Java always handles methods and it's usually what we want to happen. However, if the method bar() is not declared to be virtual, then this code will run the method found in class A when we call foo->bar().

Classic Question #3: A to I

Write a function to convert a string into an integer. (This function is called A to I (or atoi()) because we are converting an ASCII string into an integer.)

Good answer: Go through the string from beginning to end. If the first character is a negative sign, remember this fact. Keep a running total, which starts at 0. Each time you reach a new digit, multiply the total by 10 and add the new digit. When you reach the end, return the current total, or, if there was a negative sign, the inverse of the number.

Okay answer: Another approach is to go through the string from end to beginning, again keeping a running total. Also, remember a number x representing which digit you are currently on; x is initially 1. For each character, add the current digit times x to the running total, and multiply x by 10. When you reach the beginning, return the current total, or, if there was a negative sign, the inverse of the number.

Note: The interviewer is likely to ask you about the limitations of your approach. You should mention that it only works if the string consists of an optional negative sign followed by digits. Also, mention that if the number is too big, the result will be incorrect due to overflow.

Hacking a Google Interview – Handout 2

Course Description

Instructors: Bill Jacobs and Curtis Fonger
Time: January 12 – 15, 5:00 – 6:30 PM in 32-124
Website: http://courses.csail.mit.edu/iap/interview

Classic Question #4: Reversing the words in a string

Write a function to reverse the order of words in a string in place.

Answer: Reverse the string by swapping the first character with the last character, the second character with the second-to-last character, and so on. Then, go through the string looking for spaces, so that you find where each of the words is. Reverse each of the words you encounter by again swapping the first character with the last character, the second character with the second-to-last character, and so on.

Sorting

Often, as part of a solution to a question, you will need to sort a collection of elements. The most important thing to remember about sorting is that it takes O(n log n) time. (That is, the fastest sorting algorithm for arbitrary data takes O(n log n) time.)

Merge Sort:

Merge sort is a recursive way to sort an array. First, you divide the array in half and recursively sort each half of the array. Then, you combine the two halves into a sorted array. So a merge sort function would look something like this:

```
int[] mergeSort(int[] array) {
    if (array.length <= 1)
        return array;
    int middle = array.length / 2;
    int firstHalf = mergeSort(array[0..middle - 1]);
    int secondHalf = mergeSort(
                        array[middle..array.length - 1]);
    return merge(firstHalf, secondHalf);
}
```

The algorithm relies on the fact that one can quickly combine two sorted arrays into a single sorted array. One can do so by keeping two pointers into the two sorted

arrays. One repeatedly adds the smaller of the two numbers pointed to to the new array and advances the pointer.

Quicksort:

Quicksort is another sorting algorithm. It takes O(n^2) time in the worst case and O(n log n) expected time.

To sort an array using quicksort, one first selects a random element of the array to be the "pivot". One then divides the array into two groups: a group of elements that are less than the pivot and a group of elements that are greater than the pivot. After this, there will be an array consisting of elements less than the pivot, followed by the pivot, followed by elements greater than the pivot. Then, one recursively sorts the portion of the array before the pivot and the portion of the array after the pivot. A quicksort function would look like this:

```
void quicksort(int[] array, int startIndex, int endIndex) {
    if (startIndex >= endIndex) {
        // Base case (array segment has 1 or 0 elements
    } else {
        int pivotIndex = partition(array,
                                   startIndex,
                                   endIndex);
        quicksort(array, startIndex, pivotIndex - 1);
        quicksort(array, pivotIndex + 1, endIndex);
    }
}
```

Quicksort is typically very fast in practice, but remember that it has O(n^2) worst-case running time, so be sure to mention another sorting algorithm, such as merge sort, if you need guaranteed O(n log n) running time.

Order Statistics:

Sometimes, an interviewer will ask you to describe an algorithm to identify the kth smallest element in an array of n elements. To do this, you select a random pivot and partition the array as you would in the quicksort algorithm. Then, based on the index of the pivot element, you know which half of the array the desired element lies in. For example, say k = 15 and n = 30, and after you select your pivot and partition the array, the first half has 10 elements (the half before the pivot). You know that the desired element is the 4th smallest element in the larger half. To identify the element, you partition the second half of the array and continue recursively. The reason that this is not O(n log n) is that the recursive partition call is only on one half of the array, so the expected running time is n + (n/2) + (n/4) + (n/8) + ... = O(n).

Note that finding the median of an array is a special case of this where k = n / 2. This is a very important point, as an interviewer will often ask you to find a way to get the median of an array of numbers.

Question: Nearest Neighbor

Say you have an array containing information regarding n people. Each person is described using a string (their name) and a number (their position along a number line). Each person has three friends, which are the three people whose number is nearest their own. Describe an algorithm to identify each person's three friends.

Good answer: Sort the array in ascending order of the people's number. For each person, check the three people immediately before and after them. Their three friends will be among these six people. This algorithm takes O(n log n) time, since sorting the people takes that much time.

Linked Lists

A linked list is a basic data structure. Each node in a linked list contains an element and a pointer to the next node in the linked list. The last node has a "null" pointer to indicate that there is no next node. A list may also be doubly linked, in which case each node also has a pointer to the previous node. It takes constant (O(1)) time to add a node to or remove a node from a linked list (if you already have a pointer to that node). It takes O(n) time to look up an element in a linked list if you don't already have a pointer to that node.

Classic Question #5: Cycle in a Linked List

How can one determine whether a singly linked list has a cycle?

Good answer: Keep track of two pointers in the linked list, and start them at the beginning of the linked list. At each iteration of the algorithm, advance the first pointer by one node and the second pointer by two nodes. If the two pointers are ever the same (other than at the beginning of the algorithm), then there is a cycle. If a pointer ever reaches the end of the linked list before the pointers are the same, then there is no cycle. Actually, the pointers need not move one and two nodes at a time; it is only necessary that the pointers move at different rates. This takes O(n) time. This is a tricky answer that interviewers really like for some reason.

Okay answer: For every node you encounter while going through the list one by one, put a pointer to that node into a O(1)-lookup time data structure, such as a hash set. Then, when you encounter a new node, see if a pointer to that node already exists in your hash set. This should take O(n) time, but also takes O(n) space.

Okay answer: Go through the elements of the list. "Mark" each node that you reach. If you reach a marked node before reaching the end, the list has a cycle; otherwise, it does not. This also takes O(n) time.

Note that this question is technically ill-posed. An ordinary linked list will have no cycles. What they actually mean is for you to determine whether you can reach a cycle from a node in a graph consisting of nodes that have at most one outgoing edge.

Stacks and Queues

An interviewer will probably expect you to know what queues and stacks are. Queues are abstract data types. A queue is just like a line of people at an amusement park. A queue typically has two operations: enqueue and dequeue. Enqueueing an element adds it to the queue. Dequeueing an element removes and returns the element that was added least recently. A queue is said to be FIFO (first-in, first-out).

A stack is another abstract data type with two common operations: push and pop. Pushing an element adds it to the stack. Popping an element removes and returns the element that was added most recently. A stack is said to be LIFO (last-in, first-out). A stack operates like a stack of cafeteria trays.

Hash Tables

A hash table is used to associate keys with values, so that each key is associated with one or zero values. Each key should be able to compute a "hash" function, which takes some or all of its information and digests it into a single integer. The hash table consists of an array of hash buckets. To add a key-value pair to a hash table, one computes the key's hash code and uses it to decide the hash bucket in which the mapping belongs. For example, if the hash value is 53 and there are 8 hash buckets, one might use the mod function to decide to put the mapping in bucket 53 mod 8, which is bucket 5. To lookup the value for a given key, one computes the bucket in which the key would reside and checks whether the key is there; if so, one can return the value stored in that bucket. To remove the mapping for a given key, one likewise locates the key's mapping and removes it from the appropriate bucket. Note that the hash function is generally decided on in advance.

A problem arises when two keys hash to the same bucket. This event is called a "collision". There are several ways to deal with this. One way is to store a linked list of key-value pairs for each bucket.

Insertion, removal, and lookup take expected O(1) time, provided that the hash function is sufficiently "random". In the worst-case, each key hashes to the same bucket, so each operation takes O(n) time. In practice, it is common to assume constant time.

Hash tables can often be used as smaller components of answers to questions. In our experience, some interviewers like hash tables and some don't. That is, some interviewers will allow you to assume constant time, while others will not. If you want to use a hash table, we recommend subtly trying to figure out which category your interviewer belongs to. You might, for example, say something like, "Well, I could used a hash table, but that would have bad worst-case performance." The interviewer might then indicate that he'll allow you to use a hash table.

Classic Question #6: Data structure for anagrams

Given an English word in the form of a string, how can you quickly find all valid anagrams for that string (all valid rearrangements of the letters that form valid English words)? You are allowed to pre-compute whatever you want to and store whatever you optionally pre-compute on disk.

Answer: We want to use a hash table! If your interviewer really hates hash tables (which they sometimes do for some reason), you can use a tree instead. But let's assume you can use a hash table. Then for the pre-computing step, go through each word in the dictionary, sort the letters of the word in alphabetical order (so "hacking" would become "acghikn") and add the sorted letters as a key in the table and the original word as one of the values in a list of values for that key. For example, the entry for "opst" would be the list ["opts", "post", "stop", "pots", "tops", "spot"]. Then, whenever you get a string, you simply sort the letters of the string and look up the value in the hash table. The running time is O(n log n) for sorting the string (which is relatively small) and approximately O(1) for the lookup in the hash table.

There are several other possible answers to this question, but we feel that the answer above is considered an optimal solution.

Question: Factorial Zeros

Without using a calculator, how many zeros are at the end of "100!"? (that's 100*99*98*...*3*2*1)

Answer: What you don't want to do is start multiplying it all out! The trick is remembering that the number of zeros at the end of a number is equal to the number of times "10" (or "2*5") appears when you factor the number. Therefore think about the prime factorization of 100! and how many 2s and 5s there are. There are a bunch more 2s than 5s, so the number of 5s is also the number of 10s in the factorization. There is one 5 for every factor of 5 in our factorial multiplication (1*2*...*5*...*10*...*15*...) and an extra 5 for 25, 50, 75, and 100. Therefore we have 20+4 = 24 zeros at the end of 100!.

Hacking a Google Interview – Handout 3

Course Description

Instructors: Bill Jacobs and Curtis Fonger
Time: January 12 – 15, 5:00 – 6:30 PM in 32-124
Website: http://courses.csail.mit.edu/iap/interview

Question: Deck Shuffling

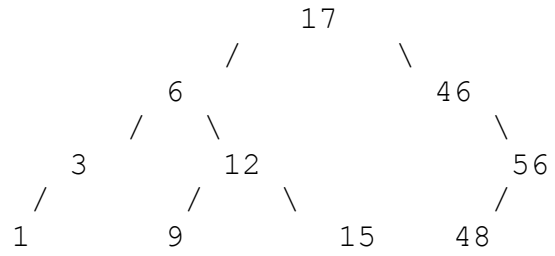
Given an array of distinct integers, give an algorithm to randomly reorder the integers so that each possible reordering is equally likely. In other words, given a deck of cards, how can you shuffle them such that any permutation of cards is equally likely?

Good answer: Go through the elements in order, swapping each element with a random element in the array that does not appear earlier than the element. This takes O(n) time.

Note that there are several possible solutions to this problem, as well as several good-looking answers that are incorrect. For example, a slight modification to the above algorithm whereby one switches each element with any element in the array does not give each reordering with equally probability. The answer given here is, in our opinion, the best solution. If you want to see other solutions, check the "Shuffling" page on Wikipedia.

Binary Search Trees

A binary search tree is a data structure that keeps items in sorted order. It consists of a binary tree. Each node has a pointer to two children (one or both of which may be null), an optional pointer to its parent (which may be null), and one element that is being stored in the tree (perhaps a string or an integer). For a binary search tree to be valid, each node's element must be greater than every element in its left subtree and less than every element in its right subtree. For example, a binary tree might look like this:



To check whether an element appears in a binary search tree, one need only follow the appropriate links from parent to child. For example, if we want to search for 15 in the above tree, we start at the root, 17. Since 15 < 17, we move to the left child, 6. Since 15 > 6, we move to the right child, 12. Since 15 > 12, we move to the right child again, 15. Now we have found the number we were looking for, so we're done.

To add an element to a binary search tree, we begin as if we were searching for the element, following the appropriate links from parent to child. When the desired child is null, we add the element as a new child node. For example, if we were to add 14 to the above tree, we would go down the tree. Once we reached 15, we would see that the node has no left child, so we would add 14 as a left child.

To remove an element from a binary search tree, we first find the node containing that element. If the node has zero children, we simply remove it. If it has one child, we replace the node with its child. If it has two children, we identify the next-smaller or next-larger element in the tree (it doesn't matter which), using an algorithm which we do not describe here for the sake of brevity. We set the element stored in the node to this value. Then, we splice the node that contained the value from the tree. This will be relatively easy, since the node will have at most one child. For example, to remove 6 from the tree, we first change the node to have the value 3. Then, we remove the node that used to have the value 3, and we make 1 the left child of the node that used to have the value 6.

A small modification to a binary search tree allows it to be used to associate keys with values, as in a hash table. Instead of storing a single value in each node, one could store a key-value pair in each node. The tree would be ordered based on the nodes' keys.

Interviewers sometimes ask about binary search trees. In addition, binary search trees are often useful as a component of an answer to interview questions. The important thing to remember is that insertion, removal, and lookup take O(log n) time (where n is the number of elements in the tree), since the height of a well-balanced binary search tree is O(log n). Although in the worst case, a binary search tree might have a height of O(n), there are "self-balancing" binary search trees that periodically reorganize a BST to ensure a height of O(log n). Many self-balancing BST's guarantee that operations take O(log n) time. If you want to learn more about particular types binary search trees, such as red-black trees, we recommend looking them up.

Question: Path Between Nodes in a Binary Tree

Design an algorithm to find a path from one node in a binary tree to another.

Good Answer: There will always be exactly one path: from the starting node to the lowest common ancestor of the nodes to the second node. The goal is to identify the lowest common ancestor.

For each node, keep track of a set of nodes in the binary tree (using a hash table or a BST) as well as a current node. At each iteration, for each of the two current nodes, change the current node to be its parent and add it to the appropriate set. The first element that is added to one set when it is already present in the other set is the lowest common ancestor. This algorithm takes O(n) time, where n is the length of the path. For example, if we were finding the lowest common ancestor of 3 and 15 in the above tree, our algorithm would do the following:

Current node 1	Current node 2	Set 1	Set 2
3	15	3	15
6	12	3, 6	15, 12
17	6	3, 6, 17	15, 12, 6

To improve the solution, we actually only need to use one set instead of two.

Bitwise Operations

Integers are represented in a computer using base two, using only 0's and 1's. Each place in a binary number represents a power of two. The rightmost bit corresponds to 2^0, the second digit from the right corresponds to 2^1, and so on. For example, the number 11000101 in binary is equal to 2^0 + 2^2 + 2^6 + 2^7 = 197. Negative integers can also be represented in binary; look up "two's complement" on Wikipedia for more details.

There are a few operations that a computer can perform quickly on one or two integers. The first is "bitwise and", which takes two integers and returns an integer that has a 1 only in places where both of the inputs had a 1. For example:

```
  00101011
& 10110010
----------
  00100010
```

Another operation is "bitwise or", which takes two integers and returns an integer that has a 0 only in places where both of the inputs had a 0. For example:

```
  00101011
| 10110010
----------
  10111011
```

"Bitwise xor" has a 1 in each place where the bits in the two integers is different. For example:

```
  00101011
^ 10110010
----------
  10011001
```

"Bitwise negation" takes a number and inverts each of the bits. For example:

```
~ 00101011
----------
  11010100
```

"Left shifting" takes a binary number, adds a certain number of zeros to the end, and removes the same number of bits from the beginning. For example, 00101011 << 4 is equal to 10110000. Likewise, right shifting takes a binary number, adds a certain number of zeros to the beginning, and removes the same number of bits from the end. For instance, 00101011 >>> 4 = 00000010. Actually, there is a more common form of right shifting (the "arithmetic right shift") that replaces the first few bits with a copy of the first bit instead of with zeros. For example, 10110010 >> 4 = 11111011.

Interviewers like to ask questions related to bitwise logic. Often, there is a tricky way to solve these problems. Bitwise xor can often be used in a tricky way because two identical numbers in an expression involving xor will "cancel out". For example, 15 ^ 12 ^ 15 = 12.

Question: Compute 2^x

How can you quickly compute 2^x?

Good answer: 1 << x (1 left-shifted by x)

Question: Is Power of 2

How can you quickly determine whether a number is a power of 2?

Good answer: Check whether x & (x - 1) is 0. If x is not an even power of 2, the highest position of x with a 1 will also have a 1 in x - 1; otherwise, x will be 100...0 and x - 1 will be 011...1; and'ing them together will return 0.

Question: Beating the Stock Market

Say you have an array for which the ith element is the price of a given stock on day i. If you were only permitted to buy one share of the stock and sell one share of the stock, design an algorithm to find the best times to buy and sell.

Good answer: Go through the array in order, keeping track of the lowest stock price and the best deal you've seen so far. Whenever the current stock price minus the current lowest stock price is better than the current best deal, update the best deal to this new value.

Program Design

Although it sometimes may seem like it, interviewers aren't always interested in little programming tricks and puzzles. Sometimes they may ask you to design a program or a system. For example, it's not uncommon to be asked to sketch out what classes you would need if you were to write a poker game program or a simulation of car traffic at an intersection. There are many different questions the interviewer could ask about design, so just keep in mind that if you need to design the classes for a program, try to keep your design simple and at the same time allow for future extensions on your design.

For example, if you were designing a five-card draw poker game program, you could have a GameMode interface or superclass and have a FiveCardDraw subclass to encapsulate the particular rules for that version of the game. Therefore if the interviewer then asks you how you could extend the system to allow a Texas hold 'em game, you could simply again make a TexasHoldEm subclass of GameMode.

Design Patterns

A design pattern is a useful design technique that programmers have discovered to be so useful over the years that they give a specific name to it. Interviewers sometimes ask you to list some design patters you know, and may even ask you to describe how a particular one works. But besides questions that directly test your knowledge of them, design patters are very useful as building blocks for solving other questions, especially the ones that focus on program design. There are several design patterns that exist, and we recommend that you take a look at the "Design Pattern" page on Wikipedia to get an idea of several of the best-know ones, but there are a few that truly stand out in their popularity and usefulness.

Listener/Observer Pattern:

This may be the most popular design pattern out there. The idea is this: suppose there were an e-mail list for Hacking a Google Interview (unfortunately there isn't one, but if we had been a bit more forward-thinking, perhaps we would have made one). This list would allow for important announcements to be sent to anyone who

cared about the class. Every student who put themselves on the list would be a "listener" (or "observer"). The teacher would be the "announcer" (or "subject" in some texts). Every time the teacher wanted to let the students know something, they would go though the e-mail list and send an announcement e-mail to each listener.

In a program, we would have a Listener interface that any class could implement. That Listener interface would have some sort of "update()" method that would be called whenever the announcer wanted to tell the listeners something. The announcer would store a list of all the listeners. If we wanted to add an object "foo" as a listener, we would call "announcer.addListener(foo)", which would cause the announcer to add foo to its list of listeners. Whenever the announcer did something important that it wanted to tell the listeners about, it would go though its list of listeners and call "update()" on each of those objects.

Going back to the poker game program, you might mention to the interviewer that you could use the listener design pattern for several things. For example, you could have the GUI be a listener to several objects in the game (such as player hands, the pot, etc.) for any changes in game state for which it would need to update the display.

Singleton Pattern:

The singleton pattern is used when you want to make sure there is exactly one instance of something in your program. If you were making a Lord of the Rings game, you would want to make sure that the One Ring was only instantiated once! We have to give Frodo a chance!

In Java, for instance, to make sure there is only one of something, you can do something like this:

```
public class OnlyOneOfMe {
    private static OnlyOneOfMe theOneInstance = null;

    private OnlyOneOfMe() {
        // do stuff to make the object
    }

    public static OnlyOneOfMe getInstance() {
        if (theOneInstance == null) {
            theOneInstance = new OnlyOneOfMe();
        }
        return theOneInstance;
    }
}
```

Notice that there is no public constructor. If you want an instance of this class, you have to call "getInstance()", which ensures that only one instance of the class is ever made.

Model-View-Controller:

Model-view-controller (MVC) is a design pattern commonly used in user interfaces. Its goal is to keep the "data" separate from the user interface. For example, when designing a program that displays stock information, the code that downloads the stock information should not depend on the code that displays the information.

The exact meaning of model-view-controller is a bit ambiguous. Essentially, a program that uses model-view-controller uses separate programming entities to store the data (the "model"), to display the data (the "view"), and to modify the data (the "controller"). In model-view-controller, the view usually makes heavy use of listeners to listen to changes and events in the model or the controller.

Model-view-controller is a good buzzword to whip out when you're asked a design question relating to a user interface.

Classic Question #7: Ransom Note

Let's say you've just kidnapped Alyssa Hacker you want to leave a ransom note for Ben Bitdiddle, saying that if he ever wants to see her again, he needs to swear to never use Scheme again. You don't want to write the note by hand, since they would be able to trace your handwriting. You're standing in Alyssa's apartment and you see a million computer magazines. You need to write your note by cutting letters out of the magazines and pasting them together to form a letter. Here's the question: given an arbitrary ransom note string and another string containing all the contents of all the magazines, write a function that will return true if the ransom note can be made from the magazines; otherwise, it will return false. Remember, every letter found in the magazine string can only be used once in your ransom note.

For example, if the ransom note string was "no scheme" and the magazine string was "programming interviews are weird", you would return false since you can't form the first string by grabbing and rearranging letters from the second string.

Pretty-good answer: Make a data structure to store a count of each letter in the magazine string. If you're programming in C, you can make an array of length 256 and simply use the ASCII value for each character as its spot in the array, since characters are 1 byte. If you're programming in Java, you can just use a hash table instead (since characters are 2 bytes in Java). Then go through the magazine string and for each character, increment the value for that letter in your data structure. After you go though the whole magazine string, you should have an exact count of how many times each character appears in the magazine string. Then go through each character in the ransom note string and for every character you encounter,

decrement the value for that letter in your data structure. If you ever find that after you decrement a count to something less than 0, you know you can't make the ransom note, so you immediately return false. If however you get though the entire ransom note without running out of available letters, you return true.

Even better answer: Because the magazine string may be very large, we want to reduce the time we spend going through the magazine string. We use the same idea as above, except we go through the ransom note and the magazine string at the same time. Keep one pointer for our current character in the ransom note and another pointer for our current character in our magazine string. First, check to see if the count in our data structure for our current ransom note character is greater than 0. If it is, decrement it and advance the pointer in our ransom note. If it isn't, start going through the characters in the magazine string, updating the counts in the data structure for each character encountered, until we reach the character we need for our ransom note. Then stop advancing the magazine string pointer and start advancing the ransom note pointer again. If we get to the end of the ransom note, we return true. If we get to the end of the magazine string (meaning we didn't find enough letters for our ransom note), we return false.

Hacking a Google Interview Practice Questions – Person A

Question: Substring

Write a program to determine whether an input string x is a substring of another input string y. (For example, "bat" is a substring of "abate", but not of "beat".) You may use any language you like.

Sample Answer (in C++):

```
bool hasSubstring(const char *str, const char *find) {
    if (str[0] == '\0' && find[0] == '\0')
        return true;

    for(int i = 0; str[i] != '\0'; i++) {
        bool foundNonMatch = false;
        for(int j = 0; find[j] != '\0'; j++) {
            if (str[i + j] != find[j]) {
                foundNonMatch = true;
                break;
            }
        }
        if (!foundNonMatch)
            return true;
    }
    return false;
}
```

Question: Text Editor

Describe a design for a text editor. Describe the classes, interfaces, and so on that you would use and how you would organize them.

Answer: There are so many possible answers to this problem that it would be difficult to say that one answer is the best. Look to make sure that they make classes to set up a text editor (classes for the GUI, formatting, saving/loading files, handling input, etc.). Using inheritance (subclassing in object-oriented programming) where it makes sense is also good for reusability and extendability. Using design patters (such as Model-View-Controller, Listener/Observer, or the Singleton pattern) is also a good thing. The main point is for them to get used to thinking about how they would design a system. Most importantly, they need to think about simplicity, reusability, and extendability in their design.

A text editor design question is slightly different from other design questions in that programmers often have strong feelings about how a text editor should work. Programmers often want the ability to greatly modify the behavior of their editor and want to be able to write extensions that add functionality to it. The major text editors used by programmers today, such as Emacs, Vim, Eclipse, and Visual Studio have this ability. A discussion about how their text editor would accomplish this (especially with how the design would include a place for extensions and how input would be handled) would be good.

Question: Axis-Aligned Rectangles

Describe an algorithm that takes an unsorted array of axis-aligned rectangles and returns any pair of rectangles that overlaps, if there is such a pair. Axis-aligned means that all the rectangle sides are either parallel or perpendicular to the x- and y-axis. You can assume that each rectangle object has two variables in it: the x-y coordinates of the upper-left corner and the bottom-right corner.

Good Answer: Create a sorted array of the x coordinates of the left and right edges of the rectangles. Then, use a "scanline" to move from left to right through the rectangles. Keep a binary search tree containing the y coordinates of the top and bottom edges of the rectangles that overlap the scanline. For each element of the array, check whether it is a left or right edge. If it is a right edge, remove the corresponding top and bottom edges from the BST. If it is a left edge, search the BST for rectangles that overlap the current rectangle; if there is one, return the overlap. Then, add the y coordinates of the top and bottom edges of the rectangle to the BST. The search takes O(n log n) time, since it takes O(n log n) time to sort the rectangles and each of the 2n iterations takes O(log n) time.

Question: Doubly Linked List

Write a function to remove a single occurrence of an integer from a doubly linked list if it is present. You may use any language you like.

Sample Answer (in Java):

```
void remove(Node head, int value) {
    Node cur = head;
    while (cur != null) {
        if (cur.value == value) {
            if (curr.prev != null)
                cur.prev.next = cur.next;
            if (cur.next != null)
                cur.next.prev = cur.prev;
            break;
        }
        cur = cur.next;
```

```
        }
}
```

Question: Minimum Stack

Describe a stack data structure that supports "push", "pop", and "find minimum" operations. "Find minimum" returns the smallest element in the stack.

Good Answer: Store two stacks, one of which contains all of the items in the stack and one of which is a stack of minima. To push an element, push it onto the first stack. Check whether it is smaller than the top item on the second stack; if so, push it onto the second stack. To pop an item, pop it from the first stack. If it is the top element of the second stack, pop it from the second stack. To find the minimum element, simply return the element on the top of the second stack. Each operation takes O(1) time.

Question: Hash Tables

Describe how a hash table works.

Answer: You can refer to handout 2 for a description of hash tables.

Question: Coin Flipping and Die Rolls

Describe an algorithm to output a die roll (a random number from 1 to 6), given a function that outputs a coin toss (a random number from 1 to 2). Each possible outcome should be equally likely.

Sample Answer: Flip the coin three times, and use the three coin flips as the bits of a three-bit number. If the number is in the range 1 to 6, output the number. Otherwise, repeat. Note that many other answers are possible.

Question: Target Sum

Given an integer x and an unsorted array of integers, describe an algorithm to determine whether two of the numbers add up to x. (In this case, say that the interviewer hates hash tables.)

Good Answer: Sort the array. Then, keep track of two pointers in the array, one at the beginning and one at the end. Whenever the sum of the current two integers is less than x, move the first pointer forwards, and whenever the sum is greater than x, move the second pointer backwards. If you cannot find two numbers that add to x before one of the pointers meet, then there is no pair of integers that sum to x. This solution takes O(n log n) time because we sort the numbers.

Another Good Answer: Create a binary search tree containing x minus each element in the array. Then, check whether any element of the array appears in the BST. It takes O(n log n) time to create a binary search tree from an array, since it takes O(log n) time to insert something into a BST, and it takes O(n log n) time to see if any element in an array is in a BST, since the lookup time for each element in the array takes O(log n). Therefore step one takes O(n log n) time and step two takes O(n log n) time, so our total running time is O(n log n).

Question: Debugging

Describe a good strategy to find a bug in a program.

Answer: This question has many possible answers, and is the sort of open-ended question that interviewers occasionally ask. A good answer to this question might include identifying the portion of the program in which the bug appears to be occurring based on its behavior, as well as using breakpoints and a stepper to step through the program. Any answers that involve thinking about possible sources of the problem and finding ways to limit the search scope of the bug are good answers.

Hacking a Google Interview Practice Questions – Person B

Question: Binary Search Tree Validity

Write a function to determine whether a given binary tree of distinct integers is a valid binary search tree. Assume that each node contains a pointer to its left child, a pointer to its right child, and an integer, but not a pointer to its parent. You may use any language you like.

Good Answer: Note that it's not enough to write a recursive function that just checks if the left and right nodes of each node are less than and greater than the current node (and calls that recursively). You need to make sure that all the nodes of the subtree starting at your current node are within the valid range of values allowed by the current node's ancestors. Therefore you can solve this recursively by writing a helper function that accepts a current node, the smallest allowed value, and the largest allowed value for that subtree. An example of this is the following (in Java):

```
boolean isValid(Node root) {
    return isValidHelper(root, Integer.MIN_VALUE,
                         Integer.MAX_VALUE);
}

boolean isValidHelper(Node curr, int min, int max) {
    if (curr.left != null) {
        if (curr.left.value < min ||
               !isValidHelper(curr.left, min, curr.value))
            return false;
    }
    if (curr.right != null) {
        if (curr.right.value > max ||
               !isValidHelper(curr.right, curr.value, max))
            return false;
    }
    return true;
}
```

The running time of this algorithm is O(n).

Question: Odd Man Out

You're given an unsorted array of integers where every integer appears exactly twice, except for one integer which appears only once. Write an algorithm (in a language of your choice) that finds the integer that appears only once.

Good Answer: Set up a hash set that we will put the integers from the array into. Have a second variable that will keep a sum. Start going through the array and for each integer, check to see if it's already in the hash set. If it is not, add that integer to the sum and store that integer in the hash set. If it is in the hash set, subtract that integer from the sum. When the algorithm finishes going through the array, the sum variable should be equal to the integer we were looking for, since it is the only number we never subtracted from the sum. This takes O(n) time and O(n) space.

```
int oddManOut(int[] array) {
    HashSet<Integer> s = new HashSet<Integer>();
    int sum = 0;
    for (int i = 0; i < array.length; i++) {
        if (s.contains(array[i])) {
            sum = sum - array[i];
        } else {
            s.add(array[i]);
            sum = sum + array[i];
        }
    }
    return sum;
}
```

Really Awesome Answer: XOR all the values of the array together! Since XOR is commutative and is its own inverse, each integer in the array that appears twice will cancel itself out, and we'll be left with the integer we're looking for. This takes O(n) time and O(1) space. We told you bitwise stuff was handy!

```
int oddManOut(int[] array) {
    int val = 0;
    for (int i = 0; i < array.length; i++) {
        val ^= array[i];
    }
    return val;
}
```

Question: Design a Poker Game

(Don't ask all these questions at the same time; ask one after another, since they build upon each other.) Without writing any actual code, describe as much as possible how you would design a poker game program. What classes would you have? What relationships would they have with each other? What would be the basic flow of the program and how would those classes play a part? If you then wanted to add a new type of poker game (such as Texas Hold 'em), how would that fit into your design?

Answer: There are so many possible answers to this problem that it would be difficult to say that one answer is the best. Look to make sure that they make classes to simulate the basic parts of a poker game (perhaps a hand, the pot, a game type or rules, a round, the deck, etc.). Using inheritance (subclassing in object-oriented programming) where it makes sense is also good for reusability and extendibility. Using design patters (such as Model-View-Controller, Listener/Observer, or the Singleton pattern) is also a good thing. The main point is for them to get used to thinking about how they would design a system. Most importantly, they need to think about simplicity, reusability, and extendibility in their design.

Question: Leader Election

Describe a technique to identify a "leader" among a group of 10 identical servers that are all connected to every other server. There are no prior distinguishing characteristics of any of them and the same program to identify the leader starts running on all of them at the same time. After an answer is given, ask how much network traffic it requires and, if "ties" are possible, ask how you can break ties.

Good Answer: Have each server wait a random amount of time and then say "I'm it." The "I'm it" announcement is time-stamped, and the computer that time-stamped its announcement first is elected the leader. This approach requires sending out 9 messages. If there is a tie, the computers can repeat the procedure.

Note that other answers are possible, but every correct answer will use randomness in some way.

Question: Queue Using Stacks

Describe a queue data structure that is implemented using one or more stacks. Don't worry about running time. Write the enqueue and dequeue operations for the queue. You may use any language you like.

Good answer: You can use two stacks: an "incoming" stack and an "outgoing" stack. The enqueue and dequeue operations would look like this (in Java):

```
Stack in;
Stack out;

void enqueue(int value) {
    while (!out.isEmpty())
        in.push(out.pop());
    in.push(value);
}
```

```
int dequeue() {
    while (!in.isEmpty())
        out.push(in.pop());
    return out.pop();
}
```

Question: Instant Messaging

Describe a design for an instant messaging program where there are several servers, clients are connected to each server, and the servers communicate with each other. Describe the classes, interfaces, and so on that you would use and how you would organize them.

Answer: As in the previous design questions, there is no best answer. Good topics to discuss are how each client communicates with a server, how the servers maintain state with the other servers, how state information is communicated between servers and clients, and the speed/reliability of their design.

Question: Maximal Subarray

Given an array, describe an algorithm to identify the subarray with the maximum sum. For example, if the input is [1, -3, 5, -2, 9, -8, -6, 4], the output would be [5, -2, 9].

Good Answer: Observe that the sum of a subarray from element i to element j is equal to the sum of the subarray from element 1 to element j minus the subarray from element 1 to element i - 1. Our algorithm will iterate through the array. The algorithm keeps track of the sum x of the elements no later than the element. It will also keep track of the minimum sum y of the subarray from the first element to an element no later than the current element. Finally, It will also keep track of the subarray z with the maximum sum so far. At each step, we update x by adding the current element to it. We update y by checking whether x < y; if so, we set y to be x. We update z by checking whether y - x is greater than z; if so, we set z to be y - x.

For example, with the sample input, our algorithm would do the following:

Current element	x	y	z
1	1	0	1
-3	-2	-2	0
5	3	-2	5
-2	1	-2	5
9	10	-2	12
-8	2	-2	12
-6	-4	-4	12
4	0	-4	12

Surprisingly, this problem is equivalent to the stock market problem described in handout 3. Given an array a1, you can "convert" it to an array a2 for the stock market problem by setting each element a2[i] to be a1[0] + a1[1] + ... + a1[i].

Question: Obstacle Avoidance

Given an n x n grid with a person and obstacles, how would you find a path for the person to a particular destination? The person is permitted to move left, right, up, and down.

Good Answer: Use the A* algorithm or another fast path-finding algorithm. (It is described on Wikipedia.)