

Speed Networking Spectacular

Sample Technical Interview

Computer Science Community*
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology

March 24, 2008

1 General Info

When applying for many co-ops and most full time software development jobs, you should be prepared for a technical interview. The three main areas that you should focus on are: algorithms, data structures, and complexity analysis.

Many of the questions will require you to *actually write code!* Be sure you have a clear understanding of the problem and don't be afraid to ask questions about the problem before you begin.

When solving a problem, make sure to always keep talking (verbalize your thought process). This shows the interviewer that you aren't just "blanking" and that you are considering many options. When you make important design decisions (i.e., a decision to use a linked list over an array), explain *why* you chose that data structure over others. This shows a broad understanding of data structures.

When you begin to write code, make sure to use proper coding techniques. Always use intuitive variable names instead of *foo* or *bar*. If your code is at all tricky, comment your code as you write it (it will help both you and the interviewer understand your algorithm). Also add error checking when applicable.

After completing your code, read through it a few times and perform a sanity check. Discuss any limitations of your code (i.e., an int has a max value of $2^{31} - 1$ in many languages). Also discuss the efficiency and complexity of your algorithm.

From my experience, most companies have stopped asking the "mind puzzlers", but it's still a good idea to be familiar with the creative processes needed to think through these (hint: draw a diagram).

2 Array Question

Question: Given a string, find the first non-repeated character. For example, the first non-repeated character in "total" is 'o' and in "sciences" is 'i'.

Answer: A naïve approach would be to take each character and compare it against every other character in the string to determine if it occurs again. This solution has a runtime of $O(n^2)$. However, this problem can be efficiently solved in $O(n)$ time with only two passes through the string:

Scan through the string and build a hashtable containing each character and the number of occurrences in the string thus far (if the hashtable contains no value for a character, store 1; otherwise, increment the value). Next, scan through the string a second time and perform a lookup into the hashtable. If the value is 1, return this character, otherwise proceed to the next character in the string. Note that this could also be implemented using an array (where the offset into the array is the integer value of the character). Both

*Examples from *Programming Interviews Exposed* by Mongan and Suojanen. Prepared by Kurt Alfred Kluever.

an array and a hashtable provide constant-time lookups, but an array would need to be as large as your character set. Therefore, if the incoming string was guaranteed to be in ASCII, it would require only 256 elements, but Unicode would require 65,000 elements. Arrays are more efficient for longer strings with a limited set of possible character values while hashtables are more efficient for shorter strings or when there are a large set of possible character values.

3 Linked List Questions

Question: Given a singly linked list, find the m th-to-last element of the list.

Answer: Finding the m th-from-beginning element would be a trivial task (maintain a counter and traverse the list). However, singly linked lists can only be traversed in the forward direction. However, this problem can be efficiently solved in $O(n)$ time with a single traversal while maintaining only 2 pointers:

Maintain two pointers: a current position pointer and an m -behind pointer. Start both pointers at the head of the linked list. Advance the current pointer m element (now the current pointer and the m -behind pointer are exactly m elements apart). Continue traversing the list with the current pointer, but now advance both pointers. When the current pointer hits the end of the list, the m -behind pointer will be exactly m elements from the end of the list.

Question: Given a singly linked list, determine if it contains a cycle.

Answer: At first glance, it may appear that this problem cannot be solved without marking the elements in the list or maintaining an additional data structure. However, it can be done in $O(n)$ time while maintaining only 2 pointers:

Maintain two pointers: a slow pointer and a fast pointer. Start both pointers at the head of the linked list. Loop indefinitely. If the fast pointer reaches the end of the list (a null reference), then return that the list is acyclic. If the fast pointer “catches” the slow pointer, then return that the list is cyclic. Advance the slow pointer 1 element, advance the fast pointer 2 elements, and loop.

4 Factorial Questions

Question: Write a recursive implementation of factorial.

Answer:

RECURSIVE-FACTORIAL(n)

```
1  if ( $n > 1$ )
2      then return  $n * \text{RECURSIVE-FACTORIAL}(n - 1)$ 
3      else return 1
```

Question: Write an iterative implementation of factorial.

Answer:

ITERATIVE-FACTORIAL(n)

```
1   $f \leftarrow 1$ 
2  while ( $n > 1$ )
3      then  $f \leftarrow (f * n)$ 
4           $n \leftarrow n - 1$ 
5  return  $f$ 
```

Question: Which implementation is faster? Why?

Answer: In practice, iterative solutions are almost always faster than recursive solutions. Iterative solutions do not require the overhead of function calls. Note that all recursive functions can be implemented iteratively. However, sometimes they require the use of a stack to mimic recursion. In fact, tail recursion can be easily (and commonly is) optimized into an iterative solution a compiler.

5 Mind Puzzler

Question: You are given two fuses and a lighter. When lit, each fuse takes exactly 1 hour to burn from one end to the other. The fuses do not burn at a constant rate, though, and they are not identical. In other words, you may make no assumptions about the relationship between the length of a section of fuse and the time it has taken or will take to burn. Two equal lengths of fuse will not necessarily take the same time to burn. Using only the fuses and the lighter, measure a period of exactly 45 minutes.

Answer: If you light both ends of the first fuse at the same time, they will burn towards each other until they meet and extinguish each other after exactly 30 minutes. If you also light one end of the second fuse at the same time as you light both ends of the first fuse, you will end up with 30 minutes of the second fuse left over after the first fuse goes out. Therefore, if you light the unlit end of the second fuse after the first fuse has gone out, the 30 minutes will be cut in half and the second fuse will go out in 15 minutes. Since the first fuse took 30 minutes to burn and the second fuse took an additional 15 minutes, we have measured exactly 45 minutes.