Technical Interview Notes 2012
Software Engineering

Compiled by: Michael Lossos

Books and Resources

Cracking the Coding Interview
Programming Interviews Exposed
The Algorithm Design Manual

Introduction to Algorithms by Cormen et al
The Art of Computer Programming by Knuth
Introduction to Information Retrieval by Manning et al

Code: https://github.com/michaellossos/coding_interview

Maths focused: http://projecteuler.net/

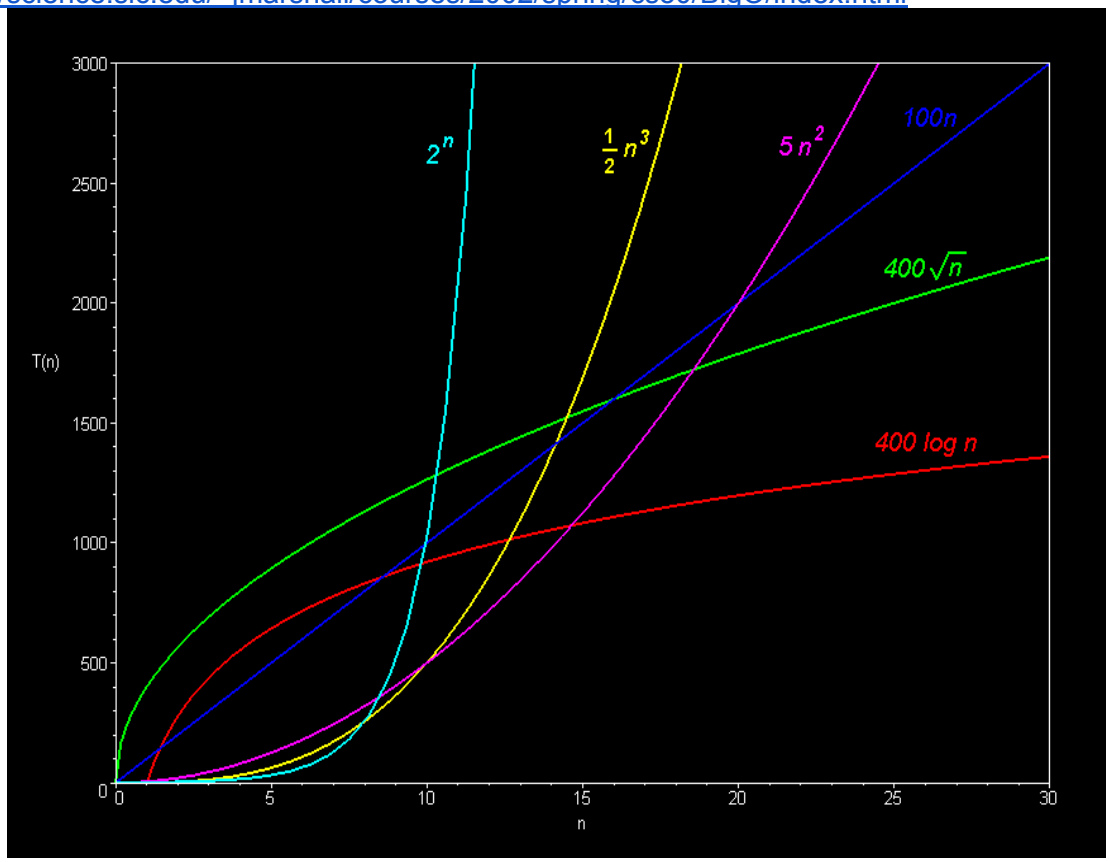
http://www.mytechinterviews.com/
http://www.careercup.com/page?pid=google-interview-questions
http://www.glassdoor.com/Interview/Google-Interview-Questions-E9079.htm

HN: http://www.google.com/search?ie=UTF-8&q=site%3Aycombinator.com+technical+interview+questions

Algorithm Complexity

Constant O(1)
Logarithmic -- O(log n)
Square root -- O(sqrt n)
Linear -- O(n)
? O(n log n)
Quadratic O(n^2)
Cubic -- O(n^3)
Exponential -- O(2^n)

Visualization of Big O:
http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/index.html



Data Structures

Binary Search Tree (BST)

wikipedia
In computer science, a binary search tree (BST), which may sometimes also be called an ordered or sorted binary tree, is a node-based binary tree data structure which has the following properties:[1]

The left subtree of a node contains only nodes with keys less than the node's key.
The right subtree of a node contains only nodes with keys greater than or equal to the node's key.
Both the left and right subtrees must also be binary search trees.

	Average	Worst
Space	O(n)	O(n)
Search	**O(log n)**	**O(n)**
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)

Search and tree traversal
http://en.wikipedia.org/wiki/Tree_traversal
http://en.wikipedia.org/wiki/Breadth-first_search

Depth first search
- Preorder: root, left, right
- Inorder (symmetric): left, root, right
- Postorder: left, right root

Breadth first search
- Level order: visit every node on each level before going lower (deeper).

Binary search:
"This lookup is a fast operation because you eliminate half the nodes from your search on each iteration by choosing to follow the left subtree or the right subtree. In the worst case, you will know whether the lookup was successful by the time there is only one node left to search. Therefore, the running time of the lookup is equal to the number of times that you can halve *n* nodes before you get to 1. This number, *x,* is the same as the number of times you can double 1 before reaching *n,* and it can be expressed as

2x = *n.* You can find *x* using a logarithm. For example, log2 8 = 3 because 23 = 8, and so the running time of the lookup operation is O(log2(n)). It is common to omit the base 2 and call this O(log(n)). log(n) is very fast. Consider that log21,000,000,000 (30. Logarithms with different bases differ by a constant factor, so the base is ignored in big-O notation. *Lookup is an O(log(n)) operation in a binary search tree.*
Note one caveat to saying that lookup is O(log(n)) in a BST. Lookup is only O(log(n)) if you can guaran- tee that the number of nodes remaining to be searched will be halved or nearly halved on each iteration. In the worst case, each node has only one child. In such a case, you have a linked list because each node points to only one other node. Lookup then becomes an O(n) operation just as in a linked list. The good news is that there are ways to guarantee that every node has approximately the same number of nodes on its left side as its right. A tree with approximately the same number of nodes on each side is called a *balanced tree."*
- Programming Interviews Exposed

Red Black Tree

“A type of self balancing binary search tree.” (wiki)
Balanced! (mostly)
“the path from the root to the furthest leaf is no more than twice as long as the path from the root to the nearest leaf.”

Space	O(n)	O(n)
Search	**O(log n)**	**O(log n)**
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

Prefix Tries

wiki
“Unlike most other algorithms, tries have the peculiar feature that the code path, and hence the time required, is almost identical for insert, delete, and find operations. As a result, for situations where code is inserting, deleting and finding in equal measure, tries can handily beat binary search trees, as well as provide a better basis for the CPU's instruction and branch caches.”

There are numerous other uses for prefix tries (see wiki and google). Suffix trees!

Skip Lists

O(log n) search, insertion, deletion.
http://cg.scs.carleton.ca/~morin/teaching/5408/refs/p90b.pdf
http://en.wikipedia.org/wiki/Skip_list

Hash Table / Hash Map

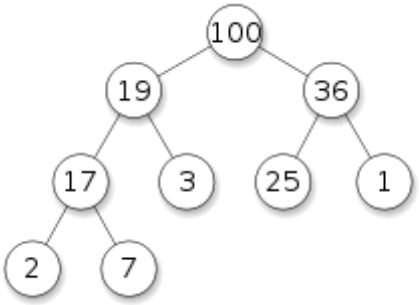
http://en.wikipedia.org/wiki/Hash_table
k keys
n entries

Space	O(*n*)	O(*n*)

Search	O(1 + *n*/*k*) [2]	O(*n*)
Insert	O(1)	O(1)
Delete	O(1 + *n*/*k*)	O(*n*)

[2] O(1) contant time retrieval (get) with a hash function that evenly distributes over buckets. javadoc

Heap



wiki
tree-based data structure that satisfies the heap property: if B is a child node of A, then key(A) ≥ key(B). T

Uses: Heapsort, graph algorithms, selection (find min/max/median of kth element)

Linked List

wiki

Insert / delete at ends O(1)
Insert / delete in middle O(n)
Indexing (“search”) O(n)

Stack

LIFO

methods:
push: add item to end
pop: remove item from end

Queue

FIFO

methods:
enqueue (append): add the item to the end
dequeue (popleft): remove and return first item

Deque - double ended queue

Less Common Trees and Things

Usually too complicated for interview questions.

Bitwise Trie
http://en.wikipedia.org/wiki/Trie

Treap

http://en.wikipedia.org/wiki/Treap

Splay Tree

http://en.wikipedia.org/wiki/Splay_tree

AA Tree
http://en.wikipedia.org/wiki/AA_tree

Sorting

- Insertion sort - iterate over values and swap. O(n^2) average. Stable.
- Merge sort - average and worst case time O(n log n), worst case space O(n). Fewer comparisons than quicksort. Stable.
- Heap sort - average and worst case time O(n log n), slower in practice than quicksort. Not stable.
- Quicksort - time complexity average O(n log n), worst O(n^2) (rare). Worst case space O(log n). Not stable.
- Timsort - average and worst case time O(n log n) (actually Theta), best case O(n). Worst case space O(n). Stable. A merge sort variant that identifies already sorted runs.

Nearly useless: selection sort, bubble sort.

Stable sorts maintain relative position of “equal” keys.

Visualizing sorting:
http://corte.si//posts/code/visualisingsorting/index.html
http://corte.si//posts/code/timsort/index.html

```
def quicksort_in_place(values):
	"""
	O(n log n) with constant additional memory. (Other than the recursion stacks!)

	Reference: Introduction to Algorithms section 7.1
	"""
	def qs(start, end):
	if start >= end:
		return
	# print values  # Uncomment this to see the divide and conquer progression.
	q = partition(start, end)
	qs(start, q-1)
	qs(q+1, end)

	def partition(start, end):
	# Illustration of regions in the partition:
	#
	# start i|             |j     |end
	# |________|__________|_____|pivot
	#  <= pivot |  > pivot |     |
	#
	pivot = values[end]
	i = start - 1
	for j in xrange(start, end):
		# Order is determined by this comparison.
#<= for ascending smallest to largest, > for descending
		if values[j] <= pivot:
		i += 1
		# Swap to move values into the region <= pivot.
		values[i], values[j] = values[j], values[i]

	# Finally, swap the pivot value[end] to the end of the <= pivot region at i+1.
	values[i+1], values[end] = values[end], values[i+1]
	return i+1

	qs(0, len(values)-1)
	return values
```

```
def merge_sort(values):
    # From: http://rosettacode.org/wiki/Sorting_algorithms/Merge_sort#Python

    # Note: you could reuse Python's heapq.merge()
    def merge(left, right):
        result = []

        while left and right:
                if left[0] <= right[0]:
                result.append(left.pop(0))
                else:
                result.append(right.pop(0))

        if right:
                result.extend(right)
        if left:
                result.extend(left)
        return result

    def msort(m):
        if len(m) <= 1:
                return m

        middle = len(m) / 2
        left = m[:middle]
        right = m[middle:]

        left = msort(left)
        right = msort(right)
        return list(merge(left, right))

    return msort(values)
```

Object Oriented Programming

OOP / OOD

http://en.wikipedia.org/wiki/Object-oriented_programming

- Data abstraction
- Encapsulation
- Polymorphism
- Inheritance
- Messaging
- Modularity

Knowledge / Trivia

How does garbage collection work? (Python, Ruby, Java)

- Hotspot JVM : generational GC http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html
- Python (CPython) : reference counted GC with cycle detection http://stackoverflow.com/questions/21934/why-java-and-python-garbage-collection-methods-are-different
- Pypy has a bunch of GC's http://doc.pypy.org/en/latest/garbage_collection.html
- Ruby MRI 1.8.x mark and sweep (pauses), 1.9.x lazy sweep, 2.x bitmap marking http://engineering.twitter.com/2011/03/building-faster-ruby-garbage-collector.html

"Garbage collection is a form of automatic memory management." Garbage is memory occupied by objects that are no longer referenced. The garbage collector frees memory occupied by garbage.
Generational garbage collector(s). Generations (memory pools) from: young (Eden, survivors), to tenured, to permanent (objects describing classes, et al).

- GC detects (object) survivors and promotes (copies) them to the next generation during a copy collection. When a young generation is full there is a minor collection (fast, time proportional to objects collected). When the tenured collection is full there is a major collection (entire heap collected, takes much longer).

Multiple garbage collectors available in the Hotspot JVM: serial GC, parallel GC, concurrent GC.

- "Ergonomics" is the selection of GC strategy, heap size, and runtime compiler based on machine, environment, command line options.
- There are trade offs between throughput (amount of time not spent in GC) and responsiveness (avoiding GC pauses). "Techniques used to minimize pauses can reduce application performance." (Concurrent GC avoids pauses but negatively impacts overall throughput.) [source]

Teach me multithreaded programming.

Multithreading is the simultaneous execution of multiple threads ("threads of execution"), or the simultaneous execution of multiple processes as in multiprocessing. Multi core processors can execute multiple processes simultaneously. Multithreading on a single processor

- Kernel threads are created by the kernel but do not have a user context. Also known as native threads.
- Lightweight processes (LWP) bind to kernel threads with a user context.
- User threads are managed in user space (as opposed to kernel space) and map 1:1 or N:1 to kernel threads.
- Green threads are scheduled by a virtual machine.

Also: what's a deadlock? Livelock? Starvation?

“A deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does.” http://en.wikipedia.org/wiki/Deadlock

“A deadlock situation can arise if and only if all of the following conditions hold simultaneously in a system:[1]

1. **Mutual Exclusion:** At least one resource must be non-shareable.[1] Only one process can use the resource at any given instant of time.
2. **Hold and Wait** or **Resource Holding:** A process is currently holding at least one resource and requesting additional resources which are being held by other processes.
3. **No Preemption:** The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process voluntarily.
4. **Circular Wait:** A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, P = {P1, P2, ..., PN}, such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3 and so on till PN is waiting for a resource held by P1.[1][7]

“A **livelock** is similar to a deadlock, except that the states of the processes involved in the livelock constantly change with regard to one another, none progressing.[10] Livelock is a special case of resource starvation; the general definition only states that a specific process is not progressing.

Canonical concurrency problems:
Producer / consumer problem
http://en.wikipedia.org/wiki/Producers-consumers_problem
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/Condition.html

Dining philosophers problem

http://stackoverflow.com/questions/1385843/simple-deadlock-examples

```
/**
 * Adapted from The Java Tutorial
 * Second Edition by Campione, M. and
 * Walrath, K.Addison-Wesley 1998
 */

// ...
public class Deadlock {
  public static void main(String[] args){
    //These are the two resource objects
    //we'll try to get locks for
    final Object resource1 = "resource1";
    final Object resource2 = "resource2";
    //Here's the first thread.
    //It tries to lock resource1 then resource2
    Thread t1 = new Thread() {
      public void run() {
        //Lock resource 1
        synchronized(resource1){
          System.out.println("Thread 1: locked resource 1");
          //Pause for a bit, simulating some file I/O or
          //something. Basically, we just want to give the
          //other thread a chance to run. Threads and deadlock
          //are asynchronous things, but we're trying to force
          //deadlock to happen here...
          try{
            Thread.sleep(50);
          } catch (InterruptedException e) {}

          //Now wait 'till we can get a lock on resource 2
```

```
        synchronized(resource2){
          System.out.println("Thread 1: locked resource 2");
        }
      }
    }
  };

  //Here's the second thread.
  //It tries to lock resource2 then resource1
  Thread t2 = new Thread(){
    public void run(){
      //This thread locks resource 2 right away
      synchronized(resource2){
        System.out.println("Thread 2: locked resource 2");
        //Then it pauses, for the same reason as the first
        //thread does
        try{
              Thread.sleep(50);
        } catch (InterruptedException e){}

        //Then it tries to lock resource1.
        //But wait!  Thread 1 locked resource1, and
        //won't release it till it gets a lock on resource2.
        //This thread holds the lock on resource2, and won't
        //release it till it gets resource1.
        //We're at an impasse. Neither thread can run,
        //and the program freezes up.
        synchronized(resource1){
          System.out.println("Thread 2: locked resource 1");
        }
      }
    }
  };

  //Start the two threads.
  //If all goes as planned, deadlock will occur,
  //and the program will never exit.
  t1.start();
  t2.start();
 }
}
```

API Design

http://37signals.com/svn/posts/3018-api-design-for-humans
http://news.ycombinator.com/item?id=4019040

TODO Add more wonderful links, especially Apigee talks

Databases and SQL

Normal forms:

- first normal form: no repeating groups
    - (e.g. bad: columns that represent the same/similar data repeated per row)
- second normal form: no column should depend on part of the primary key
    - (e.g. bad: values in a column change purely based on one part of a multi-column primary key)
- third normal form: no column can depend on a non-key column
    - (e.g. bad: one column's values always vary according another column's values)
- fourth normal form, a record type should not contain two or more independent multi-valued facts about an entity.
- fifth normal form, stricter rules to prevent data redundancy

source: http://www.bkent.net/Doc/simple5.htm

ACID : atomic, consistent, isolated, durable (Atomicity, Consistency, Isolation, and Durability)
source: http://en.wikipedia.org/wiki/ACID

- The Consistency property ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not).
- Consistency states that only valid data will be written to the database.
- Isolation refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction.
- Durability refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone.

Transaction isolation levels

Sybase: http://manuals.sybase.com/onlinebooks/group-as/asg1250e/sqlug/@Generic__BookTextView/53713%3Bpt=52735/*

Level 0 - ensures that data written by one transaction represents the actual data. It prevents other transactions from changing data that has already been modified (through an insert, delete, update, and so on) by an uncommitted transaction. The other transactions are blocked from modifying that data until the transaction commits. However, other transactions can still read the uncommitted data, which results in dirty reads.

Level 1 - prevents dirty reads. Such reads occur when one transaction modifies a row, and a second transaction reads that row before the first transaction commits the change. If the first transaction rolls back the change, the information read by the second transaction becomes invalid. This is the default isolation level supported by Adaptive Server.

Level 2 - prevents nonrepeatable reads. Such reads occur when one transaction reads a row and a second transaction modifies that row. If the second transaction commits its change, subsequent reads by the first transaction yield different results than the original read.

Adaptive Server supports this level for data-only-locked tables. It is not supported for allpages-locked tables.

Level 3 - ensures that data read by one transaction is valid until the end of that transaction, hence preventing phantom rows. Adaptive Server supports this level through the holdlock keyword of the select statement, which applies a read-lock on the specified data. Phantom rows occur when one transaction reads a set of rows that satisfy a search condition, and then a second transaction modifies the data (through an insert, delete, update, and so on). If the first transaction repeats the read with the same search conditions, it obtains a different set of rows.

joins: inner, left outer, right outer, nested, self
http://dev.mysql.com/doc/refman/6.0/en/join.html
"having" clause sets conditions for the group by clause in the same way
"where" sets conditions for the select clause, except where
cannot include aggregates, while having often does
correlated subquery
http://dev.mysql.com/doc/refman/6.0/en/correlated-subqueries.html
might be optimized but, if not, likely to be slow
subqueries can create derived temporary tables
explain select ...
http://dev.mysql.com/doc/refman/6.0/en/using-explain.html
Equijoins: Joins based on equality (=) are called equijoins.
Equijoins compare the values in the columns being joined
for equality and then include all the columns in the tables
being joined in the results.
By definition, the results of an equijoin contain two identical columns.
Natural join: removes the duplicate columns from an equijoin.

http://manuals.sybase.com/onlinebooks/group-as/asg1250e/sqlug/@Generic__BookTextView;pt=12120;nh=1
http://dev.mysql.com/doc/refman/6.0/en/examples.htm l

SYBASE SQL

```
-- _____________________________________________________________


table Person
- name
- address
- age

table Employee
- name
- position

-- _____________________________________________________________

-- inner join
select name, address, position
       from Person a
       inner join Employee b
       on a.name = b.name
       where age > 20

-- inner join with nested select
select s1.article, s1.dealer, s1.price
```

```
from shop s1
inner join (
    select article, max(price) as price
    from shop
    group by article) as s2                -- alias the temporary table
    on s1.article = s2.article and s1.price = s2.price;
-- ____________________________________________________________


-- nested select
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);
-- ____________________________________________________________


-- left [outer] join
---- all rows in left with nulls represent unmatched rows from the right table


-- nested left outer join
select *
from titles left join titleauthor
on titles.title_id=roysched.title_id /*join #1*/
left join roysched
on titleauthor.title_id=roysched.title_id   /*join #2*/
where titles.title_id != "PS7777"

-- ____________________________________________________________


-- insert

insert into Person (name,address,age)
       values ('Frank Lee', '500 Sansome', '38');
-- ____________________________________________________________


-- update

update Person set name = 'Frank A Lee'
       [from AnotherTable]
       where name = 'Frank Lee'
-- ____________________________________________________________


-- create index on stor_id from stores
create index stor_id_ind
on stores (stor_id)
____________________________________________________________


-- group by

SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article;

____________________________________________________________
```

```
-- correlated subquery (self join)

SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
   FROM CORPDATA.EMPLOYEE X
   WHERE EDLEVEL >
        (SELECT AVG(EDLEVEL)
        FROM CORPDATA.EMPLOYEE Y
        WHERE Y.WORKDEPT = X.WORKDEPT)   -- this is the correlation links the subquery to the
outer query



SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
  ON s1.article = s2.article AND s1.price = s2.price;
```

```
-- outer join to replace the above correlated subquery
-- note the use of s2.article is null to show only rows where s2.price is > s1.price
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL;
```

```
-- "having" clause
    ---- unverified (mysql): in the "having" you can only use columns used in the "group by"
This example is legal:

select title_id
from titles
where title_id like "PS%"
having avg(price) > $2.0     -- having allows aggregates

But this example is not:

select title_id
from titles
where avg(price) > $20       -- aggregate avg(price) not allowed in where
```

```
create table summy (
    id mediumint not null auto_increment,
    price double(8,6),
    primary key(id));

describe summy;

insert into summy (price) values ('2'),('3'),('4');

select * from summy;
```

```
select sum(price) from summy
```

Assertion

Your technical interviewing may be broken. Please ask yourself this: if the candidate knew exactly what you were going to ask throughout the entire interview and had a full day beforehand to prepare, would they ace your technical interview?

Using algorithms and trivia as technical interview tools for screening candidates is ineffective and has a high error rate. Candidates with years of experience will perform worse than those fresh out of school (having recently learned the material). The problems are not reflective of real world technical problems: if you're reimplementing quicksort or generating Pascal's triangle, you're working on the wrong thing. The answers to the problems are often something that can be googled for. The environment in which the problems are presented, with the candidate under pressure at a whiteboard with no computer nor internet, are not at all like the comfortable environment in which software engineers work. Worst of all, successfully solving algorithm problems is not indicative of the ability to work effectively as part of the team.

Coding alongside the candidate, at a computer, on a bit of code they are already familiar with, in a reassuring and supportive manner, is a more effective screening technique. While not a silver bullet, it's far more indicative of whether the interviewer will be able to work with the candidate. Successful screening can also be improved with behavioural interviewing techniques that explore how a candidate solved past technical problems and contributed to the success of their former teams. Another interviewing technique of asking the candidate to design solutions to the company's existing problems, or to talk about how to build applications, may also yield insights into how the candidate reasons about real world problems.

Of course, none of this helps when the tech industry as a whole stubbornly continues to use algorithms and trivia as primary screening tools in technical interviews.

- Michael Lossos (2012)

Related discussions:
http://37signals.com/svn/posts/3071-why-we-dont-hire-programmers-based-on-puzzles-api-quizzes-math-riddles-or-other-parlor-tricks
There's an interesting discussion on the differences between US and UK technical interviewing here on HN.
http://devinterviews.pen.io/
http://news.ycombinator.com/item?id=2385424
http://news.ycombinator.com/item?id=3428984
http://news.ycombinator.com/item?id=3369723
http://news.ycombinator.com/item?id=2854695
http://news.ycombinator.com/item?id=2855062
http://corner.squareup.com/2011/10/why-we-pair-interview.html