Advanced Algorithms, Fall 2011

Entrance Test: Solutions

Question 1 (function comparison) For each item, indicate whether the statement is true or false.

- $\log_2 n^{\log_2 n} = O(2^n)$ **true**
  take the logarithm on each side, or use l'Hopital's rule
- $n^{200} = O(2^n)$ **true**
  any polynomial grows much more slowy than any exponential (again, take logs both sides or use l'Hopital's rule)
- $n! = O(2^n)$ **false**
  while $n!$ is bounded by an exponential, that exponential does not have a constant radix; if you take logs both sides: $\log n! = \sum_{i=1}^{n} \log i$ and that sum is at least as large as its last $n/2$ terms, $\sum_{i=n/2}^{n} \log i$, which is in turn at least as large as $n/2$ times its smallest term, so $\log n! > n/2 \log n/2$, while $\log 2^n = n$.
- $lim_{n\to\infty}(1 - \frac{1}{n})^{2n} = 0$ **false**
  a very fundamental result from basic calculus states $\lim_{n\to\infty}(1 - \frac{1}{n})^n = \frac{1}{e}$; our expression is squared (raised to the power $2n$ instead of n) and so its limit is just $1/e^2$

Question 2. (set property) How many numbers from 1 to 1 million are not perfect squares or perfect cubes: 998,900, $>$ 998,900, or $<$ 998,900?
The correct answer is $>$ 998,900. There are exactly 1,000 perfect squares between 1 and 1,000,000, counting both 1 and 1,000,000; and there are exactly 100 perfect cubes. However some numbers are both perfect squares and perfect cubes: 1 and 1,000,000, but also 64 (8 square, 4 cube), 729 (27 square, 9 cube), 4096 (64 square, 16 cube), etc. If we just subtract 1,100 from 1,000,000, we count these square/cube numbers twice, so the answer is larger than 998,900.

Question 3. How many different shortest paths between (0,0) and (m,n) that pass only through the horizontal or vertical lines of the integer grid (circle one)?

- $\Theta(mn)$
- $\Theta(m^2n^2)$
- $\Theta(m^3n^3)$
- $\Theta(2^{m+n})$
- None of the above

In an $m \times n$ integer grid, every shortest path from the bottom left-hand corner to the top right-hand corner makes m moves up and n moves right. So we simply have all possible sequences of m "up" and n "right." How many distinct strings of m zeros and n ones are there? We can simply choose which of the $m+n$ bits are the m zeros, and there are $\binom{m+n}{m}$ such choices, which is not polynomial unless m is a constant, eliminating the first three answers, but is not always exponential, eliminating the fourth. Hence the correct answer is "None of the above."

Question 4. (reasoning by matching) Given a 10×10 grid $M[1 \ldots 10][1 \ldots 10]$, remove two squares $M[1][1]$(upper left) and $M[10][10]$ (lower right), Can the remaining 98 squares be completely covered by 49 non-overlapping 1×2 rectangles? Show how or indicate why not.

The answer is no. The simplest way to see it is to color squares as in a checkerboard. In such a coloring scheme, a 2×1 rectangle always has one white and one black square and so anything that can be tile with such tiles will always have an equal number of black and white tiles. The full 10×10 grid has that property, but the two squares removed are both black and so the remaining shape has 48 black squares and 50 white squares and so cannot be tiled with 2×1 rectangles.

Question 5. (binary search) Describe an algorithm that, given a sorted array of n distinct integers $A[1 \ldots n]$, checks in $O(logn)$ time whether there is an index i for which $A[i] = i$.

The answer is a plain binary search, but takes advantage of the fact that all numbers in the array are distinct. Specifically, we run a binary search; if, when testing location `i`, we find `A[i]=i`, we obviously stop and answer yes. If we find `A[i]>i`, we no longer need to search above `i`: because the numbers in the array are distinct and sorted in increasing order, we must have `A[i+1]>=A[i]+1` and thus `A[i]>i` implies `A[i+1]>i+1`. Symmetric reasoning applies if we find `A[i]<i`.

Question 6. (simple DP/optimal structure) Describe a linear-time algorithm that, given an array of n integers (positive or negative), finds the largest sum among all contiguous intervals. For example, with array $2, 1, -5, 4, 6, -3, 4, -7$, the largest sum is 11 $(= 4 + 6 + (-3) + 4)$.

We traverse the array twice, once to compute sums and the second to find the largest one. (We could do both in one traversal, but it is conceptually cleaner and simpler to separate the two phases.) The crucial observation is that we can reset a count to zero when it becomes zero or less, but otherwise should let it keep increasing—that it, there is no reason to chop off the head of an interval that is still contributing some positive value. In the first traversal, we will store a sum and a pointer to the beginning of the corresponding interval at each index location. If the sum at the next index is negative or zero, we reset the sum to zero and the beginning index to the next index value. In the second traversal, we look for the largest value and simply read off the current index (right endpoint of interval) and the stored beginning index (left endpoint of interval).

Question 7. (random numbers) Alice and Bob are tossing coins independently—they are in different rooms and they can only see their own results. If each makes a random prediction (head or tail) of the other's coin toss, show that, in expectation, one of them makes the correct prediction.

Alice has probability 0.5 of getting the right answer; so does Bob. Since the two are completely independent of each other, the probability of the union (one or the other, or even both, is right) is just the sum, which is 1.

Question 8. (random numbers) Refer to Question 7 above. If Alice and Bob are allowed to discuss strategy to make predictions before they are placed in separate rooms, can they make sure that, at each turn, at least one of them will correctly predict the other's toss? Justify your yes/no answer.

This is a bit tricky. Alice and Bob agree to do the following: at each step, Alice tosses her coin and predicts that Bob will get the same result that she just got, while Bob tosses his coin and predicts that Alice will get the opposite result from the one he just got. Thus, if Bob's and Alice's tosses give the same outcome, then Alice will be right (and Bob will be wrong); and if they are different, then Bob will be right (and Alice will be wrong)—and there is no other possibility.

Question 9. (geometric reasoning) Given n blue points and n red points in the plane, all points distincts, no three collinear, is it always possible to find a (straight, infinite) line and a k, $1 \leq k < n$, such that the line leaves k red points and k blues points on one side, and $n-k$ red points and $n-k$ blue points on the other side? Give a proof or a disproof (it may help to draw a picture or two).

The answer is yes, but proving it is a bit frustrating. Consider sweeping the points from left to right, tallying red and blue points encountered. Ignoring sweeps that encounter more than one point at once, we see that we have found a suitable partition as soon as the tallies of red and blue points match, since at this point we have separated the $2n$ red and blue points into a collection of $2k$ ($k > 0$) red and blue points on the left of the current sweep line and $2(n-k)$ red and blue points on the right of the sweep line—two collections that thus cannot interfere with each other. The only problem arises when $k = n$, i.e., when the tallies only match after the sweep has covered all points; this situation arises, for instance, when all blue points lie to the "left" (with respect to the sweep line) of any red point. But we can sweep from any direction whatsoever; in particular, note that the tallies are exactly inverted when we sweep in the opposite direction. Thus, as we progressively rotate the direction of sweep, we move from a constantly positive difference between blue and red tallies to a constantly negative difference between the two. Hence there is a point in the rotation at which the difference between the tallies changes sign; at that point we are guaranteed to have a solution. The only possible problem arises when *all* points are collinear, so that the change in sign occurs on a sweep line that coincide with the line on which all points lie; but we have at least 4 points, since we have $2n$ points and $n \geq 2$, so that our hypothesis (no 3 points collinear) forbids that case.

Another way to prove it is to sort all points along any axis and look at the two extreme points. If they are the same color, then we can just go from left to right, keeping track of the number of points of each color found so far, and stop as soon as the two counts are equal: there is some k, $1 < k < 2n$, where this will happen. If the two endpoints have different colors, then construct their convex hull (if you know what that is); the two extreme points are on the hull, so, as one traverses from one to the other along the hull (on either arc), there will be at least one edge with one red endpoint and one blue endpoint. A line parallel to this edge and infinitesimally to the side of it (that is, on the side where the line will cross the convex hull) will isolate exactly this pair.

Question 10. (recurrences) Derive an exact solution to the (rather weird) recurrence $f(n + f(n - 1)) = f(n-1) + 4n - 2$, with $f(0) = 0$.

Try something simple, like $f(n) = an$, hoping it will work... We get $f(n + a(n-1)) = f(an + n - a) = a(an + n - a)$ and $f(n-1) + 4n - 2 = a(n-1) + 4n - 2$; setting the two equal yields $a(an+n-a) = a(n-1)+4n-2$, and thus $a = 2$. So the answer is $f(n) = 2n$. Note that the series is defined only for some values of n, since not every integer can be written as $n + f(n-1) = 3n-2$.

Question 11. (recurrences) Derive an exact solution to the (almost as weird) recurrence $f(n+1) = \frac{1}{2}(3f^2(n) + 1) - f(n)$, with $f(1) = 3$.

On this one, calculating a few values can be useful. We have $f(1) = 3$, then $f(2) = \frac{1}{2}(3(3^2) + 1) - 3 = 11$, then $f(3) = \frac{1}{2}(3(11^2)+1) - 11 = 171$, etc. This grows like a double exponential. Trying to unroll the recurrence to detect a pattern leads only to a confusion of terms. Can we transform the recurrence to put it in an easier form? Yes: set $g(n) = 3f(n) - 1$ and rewrite the recurrence in terms of g. Now we have $g(n) = \frac{1}{2}g^2(n)$, with $g(1) = 8$. This one can in turn be reduced by setting $g(n) = 2^{h(n)}$, leading to $h(n+1) = 2h(n) - 1$, with $h(1) = 3$, and that is easily solved to yield $h(n) = 2^n + 1$, from which we recover $f(n) = \frac{2^{2^n}+1}{3}$.

(The exam sheet had a typo—it was missing the $-f(n)$ term—and so an exact answer could not be derived. An answer of $\Omega(2^{2^n})$ would be all that one could expect.)

Question 12. (data structures) Design an algorithm that, given two sorted arrays A (of size n) and B (of size m), both sorted in increasing order, and given some positive integer k, finds the kth smallest sum of the form $a_i + b_j$ in time dependent only on k.

First note that there is no need to investigate elements of either array past the kth one, so there is a trivial algorithm that runs in $\Theta(k^2 \log k)$ time: generate all pairs $a_i + b_j$ for $1 \leq i, j \leq k$, sort the values, and return the kth one. Can we do better? Think of using a min-heap in which we keep adding new sums; the first sum is $a_1 + b_1$. Now, to generate new sums, we first remove the smallest element from the heap, say the sum $a_i + b_j$; we then generate the two new sums $a_{i+1} + b_j$ and $a_i + b_{j+1}$ and add them both to the heap. Both new sums are larger than the sum removed, but neither one need be the next larger sum: there could be some sum $a_{i-k} + b_{j+1}$ (or vice versa) that, while larger than $a_i + b_j$, is smaller than both new sums. However, notice two properties. First, that "intermediate" sum is already in the heap, since its generating parent is $a_{i-k} + b_j$ and that is smaller than $a_i + b_j$ (due to sorting) and so already removed from the heap. Second, the sum removed from the heap is smaller than *any* new sum that could be added to it—because it is smaller than anything in the heap and thus than anything that will be added to the heap when one of its elements is removed. Thus the kth element we remove from the heap is the desired sum. The time required is $O(k \log k)$.