Advanced Algorithms
Test 3: Solutions

Problem 1 (Computational Geometry).
We say that a line (not a line segment: an infinite line) is a skyline *if there exists some abscissa (x-coordinate) x it has the largest ordinate (y-coordinate) of all given lines. Given n lines in general position (i.e., no three lines intersect in the same point), design and analyze an $O(n \log n)$ algorithm to identify all skylines.*

Solution:
We can solve this problem using a geometric divide-and-conquer. We can compute the skyline itself, that is, the polygonal line, made of pieces of (some of) the various input lines, that, at every abscissa, gives the highest ordinate among all input lines. This is just an intersection of halfplanes, with each line defining a halfplane—the only trick is to realize that these halfplanes all contain the point at $(O, +\infty)$, i.e., each is "above" their defining line. (If one of the lines is vertical, we obviously have a problem. We treat those separately, removing them from the input collection; if any exist, then we have a discontinuity at their abscissa that can be noted in the answer.) Otherwise we have a collection of n halfplanes and we want to compute their intersection, which we know will be a (half-infinite) convex polygon (half-infinite because they all contain the point at infinity). We have seen in class how to compute the intersection of two convex polygons in linear time. So all we need to do here is to set up a binary tree of computations—the D&C does not need to be geometric. The total cost is $O(n \log n)$, since we have $\log n$ levels of computation and each level takes $O(n)$ total work (at first computing $n/2$ intersections of two halfplanes; at the root computing a single intersection of two convex polygon of at most $n/2$ vertices each).

Problem 2 (Dynamic Programming).
Given an English text without any spaces nor punctuation marks, we want to partition it into words. We are given an oracle that scores any substring x, returning a Quality(x) score in just one call. Quality(x) is a value between 0 and 1 that measures the likelihood that x is a word.

Design and analyze an algorithm that takes $O(n^2)$ time to partition a string of n characters into "words" so as to maximize the overall quality score, that is, the product (because this is also a probability) of the scores of the words in the partition.

Solution:
This is a DP problem in more or less "standard" form. We will look at successively increasing prefixes of the string. So let $f(k)$ be the optimal quality score for the first k characters. We can write

$$f(k) = \max_{1 \le i < k} \left(f(i-1) \cdot \text{Quality}(s[i \cdots k]) \right)$$

Initialization is just $f(0) = 1$, a neutral multiplier before we start making choices. Thus computing the kth step requires $\Theta(k)$ time (assuming the oracle has constant cost) and the overall cost is the sum over all steps, $\sum_{k=2}^{n} \Theta(k)$, which is just $\Theta(n^2)$. Correctness comes from the fact that we look at all substrings decompositions through the maximization over index i: we treat the substring from i to k and add this to the best decomposition for the prefix of size $i-1$, and retain the best choice of i. Thus every possible substring suffix is explored and the prefix for that substring is already optimized and stored.

Problem 3 (Dynamic Programming).
There are n courses and m classrooms, $m \geq n$; s_i students are taking the ith course; and the jth classroom has c_j chairs.

Design and analyze an algorithm to assign each course i to a distinct classroom a_i so as to minimize $\sum_{i=1}^{n} |s_i - c_{a_i}|$. For full credit, your algorithm should run in $O(nm)$ time; an algorithm that runs in $O(nm^2)$ time will receive 3/4 credit.

Solution:
We begin with an important observation. Assume that the courses and the classrooms are sorted in increasing order by size and consider courses i and j, with $i < j$. Then every optimal solution must assign course i to a classroom of index smaller than that assigned to course j: otherwise we could exchange the two to get a better assignment. As a special case, if we have the same number of classrooms and courses, then the optimal solution simply sorts both lists and assigns the kth course to the k classroom. (This simplified version is sometimes known as the *ski instructor* problem: match pupils to skis so that the sum of the differences in height between pupils and assigned skis is minimized.)

Thanks to this observation, we can decompose the problem into subproblems (i, j) consisting of the first i courses and the first j classrooms, where courses and classrooms are both sorted in increasing order by size. (We can do this sorting within the required time bounds for the full-credit solution as long as we have $\log m \in O(n)$, a reasonable assumption to make—few school systems have a superexponential number of classrooms to choose from!) Naturally, such a decomposition makes sense only for $i \leq j$: if we have more courses than classrooms, no solution can exist. Thus we will set $f(i, j) = +\infty$ for $i > j$. (The other initialization is to set all $(0, j)$ entries to 0, since, without any courses to assign, there is no cost.)

Now the more obvious, if more expensive, formulation for the calculation step is to look at all choices between i and j for the next assignment:

$$f(i, j) = \min_{i \leq k \leq j} (f(i-1, k-1) + |s_i - c_k|)$$

The running time immediately follows: we have to fill in every (i, j) entry, for a total of mn entries and the cost of filling in an entry is $\Theta(m)$. Correctness follows from the fact that the subproblem (i, j) can be solved without looking to larger index values thanks to our observation and then from the usual DP consideration: we look at every possible subproblem and compute its optimal value.

To get a more efficient dynamic program, we attack the optimization more directly:

$$f(i, j) = \min\{f(i-1.j-1) + |s_i - c_j|, f(i, j-1)\}$$

The running time follows immediately: we fill in everyone of the $n \times m$ entries and each new entry takes constant time to compute. The first term in the minimization is clear: one choice for the first i courses and the first j classrooms (for $i \leq j$) is to carry over the best assignment made for the first $i-1$ courses and the first $j-1$ classrooms and simply assign new course i to new classroom j, incurring an incremental cost of $|s_i - c_j|$. The second term in the minimization is trickier: it corresponds to *not* choosing to assign new course i to new classroom j, but instead looking for an optimal assignment of the first i courses to the first $j-1$ classrooms. In other words, the optimization here is based on whether or not to grab an extra classroom as the collection of courses to be scheduled increases—a simple Boolean decision, as opposed to *choosing* a particular classroom k in the previous formulation. Correctness follows from the same considerations as in the previous solution.

Problem 4 (Randomized Algorithms).
Assume that G is a graph of n vertices where every vertex has degree d.

Prove that the number of connected subgraphs of G of size k is at most nd^{2k} (a rather loose bound, but sufficient for the purpose). Now, assume that each vertex of G is removed, independently at random, with probability $1-\frac{1}{2d^2}$; show that, with probability $1-n^{-c}$ (for some suitable $c>0$), no connected component of size exceeding $\log_a n$ survives, for some constant a, $1<a<2$.

Solution:
Note first that the number of connected subgraphs of size 1 is just the number of vertices, n, while the the number of connected subgraphs of size 2 is just the number of edges, $nd/2$; both values clearly obey the bound and in both cases the bound is very loose. We can approach the problem by noting that a connected subgraph must be at least a tree (it may have additional edges beyond those of the tree—in fact, because of the regular degree, it usually does—but we analyze it as if it were just a tree), so that the number of connected subgraphs of size r cannot exceed the number of subtrees of size r. (The number is usually much less, unless d is less than 2; for instance, for $k=n$, there is a single connected "subgraph" of size $k=n$, the graph itself, but, assuming $d>1$, there are many distinct subtrees on these $k=n$ vertices.)

How can we count the number of distinct subtrees of size $r>1$ in a d-regular graph? A standard result about trees is that a tree is completely characterized by a walk of all of its edges that returns to its starting point (think of a DFS process where we print edges as we walk down and also as we walk back up; the printout defines a unique tree). Since the tree has $r-1$ edges, such a walk has $2r-2$ edges—but note that we do not need to print the last edge, as there is only one choice left at that stage, so we can use $2r-3$. At every step along the walk, we have up to d choices for the next edge (since we could retraverse the edge just traversed, as when we reach a leaf). The actual number is generally less, since, if we return to the same vertex for the kth time, we will have only $d-k$ choices left, but d is a safe (amnesiac) upper bound. With up to d choices at every step and $2r-3$ steps, there are up to d^{2r-3} distinct subtrees of size r. Since we could start the walk (root the subtree) at any of the n vertices of the graph, we conclude that there are fewer (for $n>1$, $r>1$) than nd^{2r-3} distinct connected subgraphs of size r in a d-regular graph on n vertices. However, simply multiplying by n overcounts, as the same tree can be viewed as rooted at any of its r vertices, so we can divide the bound by r, to obtain our final upper bound of nd^{2r-3}/r. This upper bound is tighter than that proposed in the statement of the problem (for instance, it is now tight for $r=2$), but remains loose, since, as observed, we are overcounting the number of subtrees as soon as vertices have degrees larger than 1.

In the second part, we simply use the bound affirmed in the first part—there is no need to use a tighter bound and thus the two parts of the problem can be solved independently. A connected subgraph of size r survives the elimination process intact with probability $(1/(2d^2))^r = 2^{-r}d^{-2r}$, so that the expected number of connected subgraphs of size r after the elimination process is at most $nd^{-2}2^{-r}r^{-1}$. Thus the expected number of surviving connected subgraphs of size at least $\log_a n$ is

$$\sum_{r=\log_a n}^{n} \frac{n}{rd^2 2^r} < \frac{n}{d^2 n^{\log_a 2}} \cdot \sum_{i=0}^{n-\log_a n} 2^{-i} < \frac{2n}{d^2 n^{\log_a 2}}$$

which is $O(n^{-\alpha}$, $\alpha>0$, for $a<2$. By Markov's inequality, the probability that at least one connected subgraph of size at least $\log_a n$ survives the elimination is thus at most $\frac{2}{d^2}n^{1-\log_a 2}$. If we choose $a<2$, the probability bound has the desired form.

Problem 5 (Randomized Analysis).
(In two parts: each part is worth 10pts.)
Let π be a permutation on the set $\{1,2,\cdots,n\}$; denote by π_i the ith element of that permutation—for instance, if the set is $\{1,2,3,4,5\}$ and our permutation π is 54321, then we have $\pi_2 = 4$ and $\pi_5 = 1$.

1. *Define $f(\pi)$ as the number of different integers in the following n-tuples:*
   $$\{\pi_1, \min\{\pi_1,\pi_2\}, \min\{\pi_1,\pi_2,\pi_3\}, \cdots, \min\{\pi_1,\pi_2,\cdots,\pi_n\}\}$$
   Assuming that all permutations are equally distributed, calculate $E[f]$.
2. *Define $g(\pi)$ as the number of different pairs in the following n-tuples:*
   $$\{(\pi_1,\pi_1), (\min\{\pi_1,\pi_2\},\max\{\pi_1,\pi_2\}), (\min\{\pi_1,\pi_2,\pi_3\},\max\{\pi_1,\pi_2,\pi_3\}), \cdots, (\min\{\pi_1,\pi_2,\cdots,\pi_n\},\max\{\pi_1,\pi_2,\cdots,\pi_n\})\}$$
   Assuming that all permutations are equally distributed, calculate $E[g]$.

Solution:

1. Let X_i be the binary random variable indicating whether the ith element of the n-tuple is different from the previous $i-1$ elements. Clearly, the event of $X_i = 1$ occurs if and only if the ith integer of π is the smallest one among the first i integers. Thus, we have $\Pr(X_i = 1) = 1/i$. Finally, $\mathbb{E}[f(\pi)] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \Pr(X_i = 1) = \sum_{i=1}^{n} 1/i = H_n$, where H_n denotes the nth harmonic number, which is $\Theta(\log n)$.

2. Let X_i be the binary random variable indicating whether the ith pair of the n-tuple is different from the previous $i-1$ pairs. Clearly, the event of $X_i = 1$ occurs if and only if the ith integer of π is either the smallest or the biggest among the first i integers. Thus, we have $\Pr(X_1 = 1) = 1$ and $\Pr(X_i = 1) = 2/i, i \geq 2$. Finally, $\mathbb{E}[f(\pi)] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \Pr(X_i = 1) = 1 + \sum_{i=1}^{n} 2/i = 2H_n - 1$, which is also $\Theta(\log n)$.