Advanced Algorithms
Test 1: Solutions

Problem 1 (amortized analysis). We need to start by defining a potential function. Since we are again looking at a number representation scheme, we will use the same potential function we used for binary representation, suitably rephrased: we will count the number of nonzero entries in the vector. Now consider the two operations. In each there are some pieces that do constant work and cause at most a constant change in potential and so can be safely ignored. The important part of INCREMENT is the while loop; if the algorithm goes through that loop j times, then the real cost is j, but note that the potential then also decreases (in that loop) by j, so the net effect of the loop in amortized terms is zero. The important part of DECREMENT is also its while loop, which is essentially a symmetric version, and for which the same reasoning holds. So each operation has a constant amortized cost per operation. As we started with a potential of zero (the number 0 is the only one to have a unique representation in this system) and the final potential cannot exceed m, everything works out.

Problem 2 (priority queues). The statement of the problem is very general; it applies to all our meldable PQs, both the amortized ones such as skew heaps and the worst-case ones such as leftist trees and binomial queues. (It applies to Fibonacci heaps as well, but there it is not required to preserve the amortized performance of DecreaseKey.) Most of the problem is solved in the statement, but we need to show it all amortizes properly. Since unification can take linear time (after a linear number of insertions), we will need a potential that reflects not just what happens with the original meldable PQ, but also with the linked list. This is simpler than it may sound: we just add (a multiple of) the size of the linked list to the potential of the original meldable PQ.

Now Insert runs in constant real time and only adds one to the potential (the linked list grows by one item), so its amortized time is constant. Unification takes time linear in the size of the linked list to build the new meldable PQ, but in the process zeroes out the contribution to the potential from the linked list, so that the two cancel out (pick the right coefficient for the potential contribution of the size of the linked list). Meld and DeleteMin now are the regular Meld and DeleteMin for the original meldable PQ, preceded by unification on each data structure. But we have seen that the net effect of unification in amortized terms is zero, so we are in fact left with the analysis of the original Meld and DeleteMin and we already know that these run in logarithmic amortized time. The total additional contribution to the potential from the linked list cannot exceed the total number of items in the data structure, so that the overal amortization works. Hence we are done.

Problem 3 (competitive analysis). The answer here is deceptively simple—there is not even any need to define a potential. If the algorithm uses only one bin, it is obviously found an optimal solution. If it uses more than one bin, consider two consecutive bins: the algorithm opened the second bin to place into it some item because that item could not have fit into the previous bin, so that the size of that item plus the occupied capacity of the previous bin must sum to a value larger than 1. Hence our online algorithm uses more than half of the total capacity of that pair of bins; the optimal algorithm cannot do better than use the total capacity, or strictly less than twice what our online algorithm used.

Problem 4 (competitive analysis). Our algorithm uses a standard logarithmic search: we start by taking one step in one direction, then, if no chest is found, return to our starting point and take one step in the other direction, at which point, if we still have not found the chest, we return again to our starting position and repeat the whole thing, but after doubling the number of steps. Let i be such that we have $2^{i-1} \leq x \leq 2^i$; then we will have to iterate i times before we reach the iteration in which we will find the chest, so that we will have walked, before that iteration, a total distance of $4\sum_{j=0}^{i-1} 2^j = 4 \cdot (2^i - 1)$ steps. At the last iteration, in the worst case, we move first in the wrong direction and so waste another $2 \cdot 2^i$ steps, before finally taking x more

steps and finding the chest. Overall, then, we will have taken $6 \cdot 2^i + x - 4$ steps; but we have $2^{i-1} \leq x$, so we can write

$$6 \cdot 2^i + x - 4 = 12 \cdot 2^{i-1} + x - 4 \leq 13x - 4 \leq 13x$$

and thus our algorithm is 13-competitive.

If we have more than 2 choices of directions because we stand at an m-way intersection, we can use exactly the same strategy, but now we try each of the m roads with 1 step, then with 2 steps, etc. The total distance walked will be, as per our previous analysis, $2m \cdot (2^i - 1) + 2(m-1) \cdot 2^i + x$ (because we try all $m-1$ wrong choices in the last iteration before hitting the correct road). Hence our competitive ratio is $O(m)$, the desired upper bound. For a lower bound, note that any algorithm whatsoever, after using $2(m-1)x$ steps, can have explored at most $m-1$ of the m roads to a distance of x each; an adversary thus can place the chest at distance x on the last road explored to that distance by the algorithm, forcing the algorithm to take at least $(2m-1)x$ steps and thus showing that the competitive ratio of any algorithm must be $\Omega(m)$. Note that the size of x does not matter, as long as it is nonzero: distances are discrete, since they are measured in steps, and x cannot be smaller than 1 (or we would already have found the chest).

Problem 5 (randomized analysis). Note that all k values tested will be distinct, since we draw from separate partitions of the set of values. Assume the number of possible roots of the polynomial in the ith subset is r_i; since each subset has size k, we have $r_i \leq k$. The probability of error per subset is r_i/k, which may seem worse than for the whole set (the denominator is much smaller), but now we must take an intersection of events, since we require acceptance for each subset. Thus the probability of error is

$$\prod_{i=1}^{k} \frac{r_i}{k} = (\frac{1}{k})^k \prod_{i=1}^{k} r_i$$

Since we have both $0 \leq r_i \leq k$ and $\sum_{i=1}^{k} r_i \leq n < k^2$, we can conclude

$$\prod_{i=1}^{k} r_i < (\frac{n}{k})^k$$

and thus the probability of error is strictly less than $(n/k^2)^k$, much as we would get by analyzing the "normal" version of Freivald's algorithm (pick a value from among the k^2 values, test it, reject if the result is nonzero, otherwise repeat; after k successful trials, answer yes). This is an upper bound on the probability of error under the assumption that the k^2 values that can be tested contain every root of the polynomial.

If we have $k > n$, then the probability of error is zero, because there must exist an i with $r_i = 0$ and in that ith subset our test cannot err. This is now in contrast with the "normal" version, in which the probability of error decreases, but does not become zero.