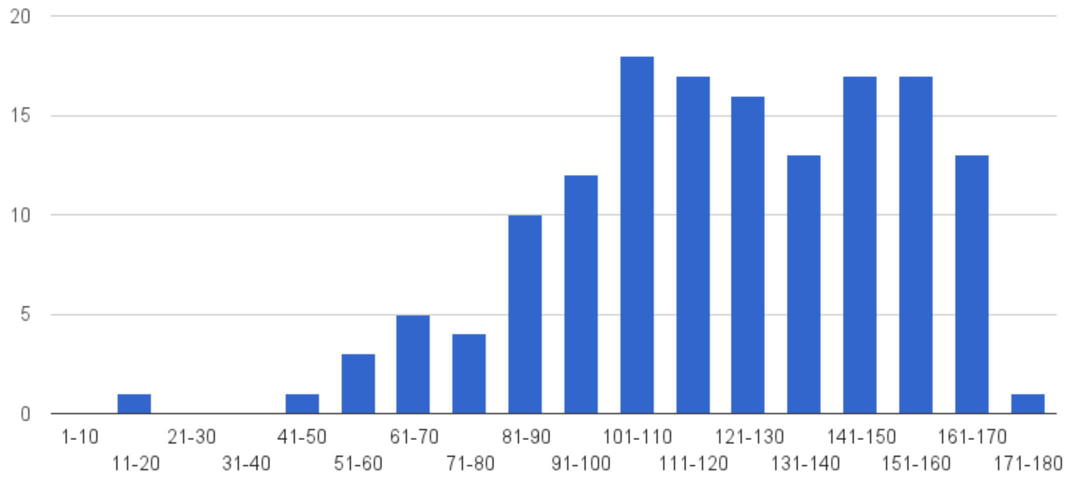
Final Exam



Mean: 121.3; Median: 122; Standard deviation: 30.8

Problem 1. True/False/Justify [40 points] (8 parts)

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, and briefly explain why. Your justification is worth more points than your true-or-false designation.

(a) T F If a problem has an algorithm that is correct for 2/3 fraction of the inputs, then it also has an algorithm that is correct for 99.9% of the inputs.

Solution: False. If an algorithm is correct for only 2/3 of the *inputs*, then it is not necessarily possible to amplify the results to make it correct for those inputs. (Consider, for example, a problem that is easy on 2/3 of the inputs but hard/undecidable for the remaining 1/3 of the inputs.) An algorithm must be correct for 2/3 of the possible sequences of random choices for amplification to apply.

(b) T F If a problem has a randomized algorithm that runs in time t and returns the correct answer with probability at least 2/3, then the problem also has a deterministic algorithm that runs in time t and always returns the correct answer.

Solution: False. Sublinear time algorithms are counterexamples, as a deterministic algorithm would not be able to read all of the input in the time it would take for the sublinear time algorithm to return an answer.

(c) T F A perfect hash table that is already built requires 2 hash functions, one for the first level hash, and another for the second level hash.

Solution: False. In the implementation of perfect hashing which we discussed in recitation, the second level hash table requires a different hash function for every bucket. Note that if we relax the requirement for taking only $O(n)$ space and allow the hash table to instead take $O(n^2)$ space, then we can use just one hash function.

(d) T F If Φ is a potential function associated with a data structure S, then 2Φ is also a potential function that can be associated with S. Moreover, the amortized running time of each operation with respect to 2Φ is at most twice the amortized running time of the operation with respect to Φ.

Solution: True. By definition, the amortized cost $\hat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$. If we use the potential function 2Φ, then because the actual cost is positive (and the same), the new amortized cost is bounded by 2 times the original.

(e) T F If we use van Emde Boas (vEB) to sort n elements, it is possible to achieve $O(n \lg \lg n)$ running time. Thus, whenever we need to use traditional $O(n \lg n)$ sorting, we can replace it with vEB sort and achieve a better asymptotic running time (ignore setup time).

Solution: False. In order to use van Emde Boas, we must have a restriction on the input (it is within a universe of integers size 1 through n).

(f) T F The "union-by-rank" and "path-compression" heuristics do not improve the running time of MAKE-SET in the union-find data structure.

Solution: True. The running time of MAKE-SET is always $O(1)$.

(g) T F There is an NP-hard problem with a known polynomial time randomized algorithm that returns the correct answer with probability 2/3 on all inputs.

Solution: False. We currently don't know whether there are NP-hard problems that can be solved by polynomial time randomized algorithms, and conjecture that these do not exist.

(h) T F Every special case of an NP-hard problem is also NP-hard.

Solution: False. Consider the MST problem, which is a special case of the Steiner Tree problem.

Problem 2. Short Answer [20 points] (4 parts)

(a) Recall the forest-of-trees solution to the disjoint-set problem. Suppose that the only heuristic we use is a variant of the union-by-rank heuristic: when merging two roots u and v, we compare the number of descendants (rather than the rank), and make u a child of v if and only if u has fewer descendants. Is this asymptotically worse than the original union-by-rank heuristic? Explain why or why not.

Solution: No, it is not asymptotically worse. The way that heuristics affect the runtime is by affecting the height of the trees in the data structure. Consider some item x in the data structure. When will the path from x to the root increase in length? Well, it can only increase in length when the root of x changes. Let u be the old root of x, and let v be the new root of x. Let $d_{old}(\cdot)$ be the number of descendants before the merge, and let $d_{new}(\cdot)$ be the number of descendants after the merge.

Because of the union-by-descendants heuristic, the root could only change from u to v if $d_{old}(u) \leq d_{old}(v)$. This means that $d_{new}(v) = d_{old}(v) + d_{old}(u) \geq 2d_{old}(u)$. So every time the path from x to the root increases by 1, the number of descendants of the root at least doubles. This means that the depth of x is at most $O(\lg n)$, just as with the union-by-rank heuristic.

(b) Suppose we apply Huffman coding to an alphabet of size 4, and the resulting tree is a perfectly balanced binary tree (one root with two children, each of which has two children of its own). Find the maximum frequency of any letter.

Solution: The maximum frequency of any letter is $2/5$. To see why, we must prove that this is both feasible, and as large as possible.

1. Suppose that we have four characters a, b, c, d with frequencies $f_a = 2/5$ and $f_b = f_c = f_d = 1/5$. Without loss of generality, suppose that Huffman's algorithm merges c and d. Then $f_{cd} = f_c + f_d = 2/5$. This introduces a tie[1], which we may break by assuming that Huffman's algorithm chooses to merge b with a. This yields a perfectly balanced tree.
2. To see that any frequency greater than $2/5$ is impossible, let $f_a \geq f_b \geq f_c \geq f_d$ be the frequencies in decreasing order. We define these frequencies so that $f_a + f_b + f_c + f_d = 1$. For the sake of contradiction, suppose that $f_a > 2/5$. This means that $f_b + f_c + f_d < 3/5$. Because $f_b > f_c, f_d$, we can conclude that $f_c + f_d < 2/5$. But this means that when c and d are merged, their combined frequency will be strictly less than f_a, so we will not get a perfectly balanced tree.

[1]To avoid a tie, we must use $2/5 - \epsilon$ as the maximum frequency instead.

(c) In lecture, we saw min-radius clustering, in which the goal was to pick a subset of k points such that each point formed the center of a cluster of radius r. Suppose instead that the center of the cluster can be a point not in the original set.

Give an example set of points where it is possible to find k clusters of radius r centered around arbitrary points, but impossible to find k clusters of radius r centered around points in the set.

Solution: Given arbitrary values of k and r, we can set up a counterexample in $\mathbb{R}^2$ as follows. For each $i \in \{1, \ldots, k\}$, create two points: $p_{i,1} = (4ri, 0.9r)$ and $p_{i,2} = (4ri, -0.9r)$. For any pair $p_{i,1}$ and $p_{i,2}$ the distance is equal to $1.8r$, so any cluster centered on one of the points cannot contain the other. For all other pairs of points, their x-coordinates must differ by at least $4r$, and so they cannot belong to the same cluster. So there's no way to create k clusters centered on k points in the set. However, there is a way to create k clusters centered on k arbitrary points: for each $i \in \{1, \ldots, k\}$, the point $(4ri, 0)$ has distance $\leq r$ to both $p_{i,1}$ and $p_{i,2}$.

(d) Consider the following algorithm, which is intended to compute the shortest distance among a collection of points in the plane:

```
1  Sort all points by their $y$-coordinate.
2  for each point in the sorted list:
3      Compute the distance to the next 7 points in the list.
4  return the smallest distance found.
```

Give an example where this algorithm will return a distance that is not in fact the overall shortest distance.

Solution: To construct a counterexample, it is sufficient to construct an example where the shortest distance is between two points that have at least 7 points between them in the ordering of y-coordinates. The example we use is as follows:

$$(0, -1) \quad (0, 5) \quad (0, 10) \quad (0, 15) \quad (0, 20) \quad (0, 25) \quad (0, 30) \quad (0, 35) \quad (0, 1)$$

In any ordering of these points by y-coordinate, the seven points with y-coordinate 0 will lie between $(0, -1)$ and $(0, 1)$. Hence, the algorithm will never compute the Euclidean distance between $(0, -1)$ and $(0, 1)$, and so it cannot find the true shortest distance.

Problem 3. Estate Showing. [30 points] (3 parts)

Trip Trillionaire is planning to give potential buyers private showings of his estate, which can be modeled as a weighted, directed graph G containing locations V connected by one-way roads E. To save time, he decides to do k of these showings at the same time, but because they were supposed to be private, he doesn't want any of his clients to see each other as they are being driven through the estate.

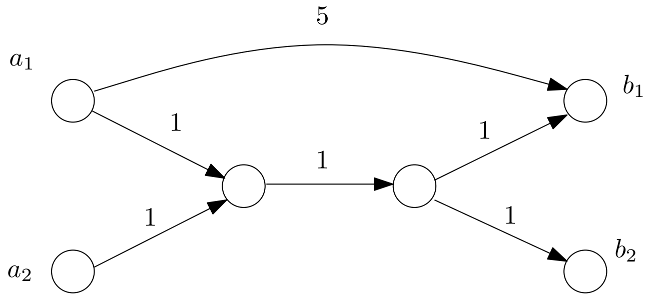
Trip has k grand entrances to his estate, $A = \{a_1, a_2, \ldots, a_k\} \subset V$. He wants to take each of his buyers on a path through G from their starting location a_i to some ending location in $B = \{b_1, b_2, \ldots, b_k\} \subset V$, where there are spectacular views of his private mountain range.

Because of your prowess with algorithms, he hires you to help him plan his buyers' visits. His goal is to find a path for each buyer i from the entrance they take, a_i, to any ending location b_j such that no two paths share any edges, and no two buyers end in the same location b_j.

(a) Trip tells you his idea: find all-pairs shortest paths on G, and then try to select k of those shortest paths $a_i \leadsto b_j$ such that all k paths start and end at different vertices and no two paths share any edges.

Give a graph where there exists a set of paths satisfying the requirements, but where Trip's strategy won't find it.

Solution: Consider this graph:



The all-pairs shortest paths algorithm would find all shortest paths from a_i to b_j ($a_1 \leadsto b_1$, $a_1 \leadsto b_2$, $a_2 \leadsto b_1$, and $a_2 \leadsto b_2$), which all go through the same edge. Trip's algorithm, which considers only those paths, would find no solution. There are two completely disjoint paths using the shortest path $a_2 \leadsto b_2$ and the direct edge (a_1, b_1), but Trip's algorithm would not find this combination because one of them is not a shortest path.

(b) Rather than using shortest paths, you think that perhaps you can formulate this as a flow problem. Find an algorithm to find k paths $a_i \leadsto b_j$ that start and end at different vertices and that share no edges, and briefly justify the correctness and running time of your algorithm.

Solution: The algorithm is as follows:

1. Create a flow network G' containing all vertices in V, all directed edges in E with capacity 1, and additionally a source vertex s and a sink vertex t. Connect the source to each starting location with a directed edge (s, a_i) and each ending location to the sink with a directed edge (b_i, t), all with capacity 1.
2. Run Ford-Fulkerson on this network to get a maximum flow f on this network. If $|f| = k$, then there is a solution; if $|f| < k$, then there is no solution, so we return FALSE. We will later show that it is always the case that $|f| \leq k$.
3. To extract the paths from a_i to b_j (as well as which starting location ultimately connects to which ending location), run a depth-first search on the returned max flow f starting from s, tracing a path to t. Remove these edges and repeat k times until we have the k disjoint paths.

The running time for this algorithm is $O(k|E|)$: in the modified graph G', we have $|E| + 2k = O(|E|)$ edges, and the maximum possible flow value is k; therefore, running Ford-Fulkerson takes $O(k|E|)$ time. Extracting paths takes at most $O(|E|)$ time, as we traverse each edge at most once. In total, then, the algorithm takes $O(k|E|)$ time.

Some students used Edmonds-Karp, which runs in $O(VE^2)$ time. In this case, because $k = O(|V|)$, Ford-Fulkerson provides a better bound.

To show correctness, we must show that there is a flow of size k if and only if there are k disjoint paths.

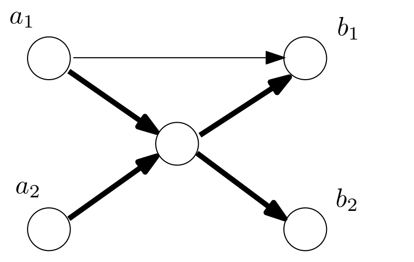
$(\rightarrow)$ **Claim.** An integral flow of size f in a network with unit capacities can be decomposed into a unit flow over a simple path $p : s \leadsto t$ and a flow of size $f - 1$ which does not use p.

Proof. Pick any simple path $p : s \leadsto t$ over edges with non-zero flow. One must exist (otherwise, we would have a cut with no flow). In the residual graph, that path has capacity 1; send an augmenting flow of -1 over that path. We then have a resulting flow of size $f - 1$, and the flow on each of the edges of p is 0.

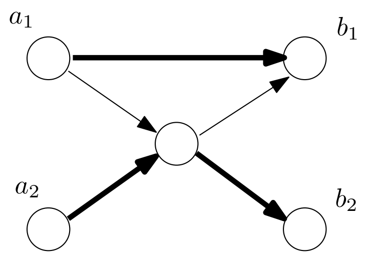
$(\leftarrow)$If there is a set of k disjoint paths from a_i to b_j, then the flow with one unit of flow on each of the edges contained in those paths defines a flow of size k. Because the cut $(s, V' - s)$ has capacity k, this is a maximum flow on the network.

A common incorrect solution involved using the augmenting paths found by Ford-Fulkerson as the k paths for the buyers to use. This approach doesn't work because the augmenting paths are not guaranteed to be the final paths along which there are units of flow.

(c) Trip, after trying out the paths found by your algorithm, realizes that making sure the k paths don't share edges isn't enough: it's possible that some paths will share vertices, and his buyers might run into each other where their paths intersect.



$a_1 \rightsquigarrow b_1$ and $a_2 \rightsquigarrow b_2$ share a vertex



$a_1 \rightsquigarrow b_1$ and $a_2 \rightsquigarrow b_2$ share neither vertices nor edges

Modify your algorithm to find k paths $a_i \rightsquigarrow b_j$ that start and end in different locations, and that share neither vertices nor edges.

Solution: Duplicate each vertex v into two vertices v_{in} and v_{out}, with a directed edge between them. All edges (u, v) now become (u, v_{in}); all edges (v, w) now become (v_{out}, w). Assign the edge (v_{in}, v_{out}) capacity 1.

With this transformation, we now have a graph in which there is a single edge corresponding to each vertex, and thus any paths that formerly shared vertices would be forced to share this edge.

Now, we can use the same algorithm as in part (b) on the modified graph to find k disjoint paths sharing neither edges nor vertices, if they exist.

The transformation of the graph takes $O(|E|)$ time, as we are adding that many new vertices and edges to the graph. The new graph has $O(|E| + |V| + 2k) = O(|E|)$ edges, and therefore the running time is unchanged.

Many students excluded the starting and ending locations a_i and b_i from the vertices duplicated. This approach, however, would overlook any paths that might have gone through some a_i or b_i before reaching their final destinations. (There was nothing in the previous problem statement preventing this from occurring.)

As in part (b), some students also tried to modify the operation of Ford-Fulkerson by removing all edges incident to vertices in the augmenting paths found, thus not allowing any other augmenting paths to go through those vertices. Again, this approach doesn't work because it is too greedy: the augmenting paths found by Ford-Fulkerson are not necessarily the final paths along which there are units of flow, and therefore this approach may miss valid solutions.

Problem 4. Credit Card Optimization [30 points] (3 parts)

Zelda Zillionaire is planning on making a sequence of purchases with costs $x_1, \ldots, x_n$. Ideally, she would like to make all of these purchases on one of her many credit cards. Each credit card has credit limit ℓ. Zelda wants to minimize the number of credit cards that she needs to use to make these purchases, without exceeding the credit limit on any card. More formally, she wants to know the smallest k such that there exists $\pi : \{1, \ldots, n\} \to \{1, \ldots, k\}$ assigning each purchase i to a credit card j, where π must satisfy the following constraint:

$$\forall j \in \{1, \ldots, k\} : \sum_{\substack{i \in \{1,\ldots,n\} \\ \text{s.t. } \pi(i)=j}} x_i \leq \ell.$$

Zelda is thinking of using the following algorithm to distribute her purchases:

```
Create an empty list of credit cards $L$.
for each purchase $x_i$:
    for each credit card $j \leq |L|$:
        if $L[j] + x_i \leq \ell$:
            Purchase $x_i$ with credit card $j$.
            Set the balance for card $j$ to $L[j] = L[j] + x_i$.
            Skip to the next purchase.
    if no existing credit card has enough credit left:
        Purchase $x_i$ with a new credit card.
        Append a new credit card to $L$.
        Set the balance of the new credit card to $x_i$.
return $k = |L|$
```

(a) Give an example where Zelda's algorithm will not use the optimal number of credit cards.

Solution:

Let the credit limit l be 10. Consider the following sequence of purchases as input to Zelda's algorithm: $2, 1, 9, 8$ The optimal solution would only use 2 credit cards by grouping purchases $\{2, 8\}$ and $\{9, 1\}$. Zelda's algorithm would split the charges into 3 cards: $\{2, 1\}$, $\{9\}$, $\{8\}$, which is suboptimal.

This problem part was worth 5 points. Almost everyone got this problem right. Some students didn't read carefully the pseudocode and inverted the order of the two loops.

(b) Show that Zelda's algorithm gives a 2-approximation for the number of credit cards.

HINT: Let OPT be the optimal number of credit cards. In order for the algorithm to add a new credit card $j > OPT$, it must reach x_i such that $x_i + \min_{j' \in \{1,\ldots,OPT\}} L[j'] > \ell$. It will then set $L[j] = x_i$.

Solution:

First, we show that the following loop invariant holds. At no point are there two distinct cards i and j, where $i < j$ such that the balance on each card $L[j] \leq \frac{l}{2}$ and $L[i] \leq \frac{l}{2}$. Suppose to the contrary that there are two such cards. Then the remaining credit on card i was sufficient to transfer all of the purchases x_i from the new credit card to the old credit card j, instead of opening up a new credit card.

We prove that Zelda's algorithm gives a 2-approximation by contradiction. Namely, we assume that there is a case where at least $2OPT + 1$ cards are needed. By the loop invariant we know that no 2 credit cards have less than $\frac{l}{2}$ as a balance. So, assuming that there are no zero purchase charges, the total amount of purchases must be more than $2OPT \cdot \frac{l}{2}$, which is the maximum amount that the OPT cards in the optimal solution can hold. This is a contradiction. So, we can always find at most $2OPT$ cards to pay for all of the purchases.

This problem part was worth 10 points.

(c) Show that minimizing the number of credit cards used is NP-hard.

Solution:

We prove that minimizing the number of credit cards is NP-hard by reducing SET-PARTITION to the Zelda's problem in polynomial time.

First, we recast Zelda's optimization problem as a decision problem. Let the decision problem be formulated as follows: For a given target number of cards k, we are able to come up with an assignment of purchases $x_1 \dots x_n$ to k credit cards, where the cumulative balance on each credit card can not exceed the limit l.

Recall that we defined SET-PARTITION as follows:

Given set $S = \{s_1, \dots s_n\}$ we can partition them into 2 disjoint subsets A and B such that $\max\{\sum_{i \in A} s_i, \sum_{j \in B} s_j\}$ is minimized.

The decision version of the SET-PARTITION problem can be reformulated as the question whether the numbers can be partitioned into two sets A and $B = S - A$ such that $\sum_{i \in A} s_i = \sum_{j \in B} s_j$.

Given an input set $S = \{s_1, \dots s_n\}$ to the SET-PARTITION problem, we can use the elements in the set as the purchases in Zelda's problem, set the number of cards to be 2 and the credit card limit $l = \frac{1}{2} \sum_{i \in S} s_i$. Clearly, if the answer to the Zelda's decision problem is in the affirmative, then each credit card will have balance of exactly $\frac{l}{2}$, and we can take the disjoint subsets of purchases on each card to be the two sets A and B that would satisfy SET-PARTITION. The reduction can be executed in linear time.

We also accepted the reduction from SUBSET-SUM to Zelda.

We gave 5 points for correctly identifying an NP-hard problem from which one could reduce to Zelda's problem to prove that the problem is NP-hard (the direction of the reduction needed to be correct to receive all the points). 10 points was given for a correct reduction and analysis.

Problem 5. Well-Separated Clusters [30 points] (3 parts)

Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a set of n points in the plane. You have been told that there is some k such that the points can be divided into k clusters with the following properties:

1. Each cluster has radius r.
2. The center of each cluster has distance at least $5r$ from the center of any other cluster.
3. Each cluster contains at least ϵn elements.

Your goal is to use this information to construct an algorithm for computing the number of clusters.

(a) Give an algorithm that takes as input r and ϵ, and outputs the exact value of k.

Solution:

Executive Summary. We find all points sufficiently close to a given point, remove all such points and increment the number of clusters, and repeat until we run out of points. This procedure is exact and takes $O(nk)$ time.

Algorithm Initialize the number of clusters $k \equiv 0$, and the set $S' \equiv S$. Choose a point $p \in S'$, and compute the distance from p to every other point in S'. Remove p and all points within distance $2r$ from S', and increment k. Repeat this procedure until S' is empty; at that point, return k.

Correctness We first note that if the point p chosen in an iteration lies in a cluster with center c, then all points within this cluster will be removed in the same iteration. Indeed, if q is any other point in this cluster, by the triangle inequality,

$$d(p, q) \leq d(p, c) + d(c, q) \leq r + r = 2r.$$

Furthermore, no points from other clusters will be removed in the same iteration. Indeed, if q' is in another cluster with center c', then by the triangle inequality

$$\begin{aligned} d(p, q') &= (d(c, p) + d(p, q') + d(q', c')) - d(c, p) - d(q', c') \\ &\geq d(c, c') - d(c, p) - d(q', c') \\ &\geq 5r - r - r = 3r. \end{aligned}$$

Runtime Analysis Each iteration computes at most n distances, and removes exactly 1 cluster. Hence there are k iterations total, so the overall runtime is $O(nk)$. Since each cluster contains at least ϵn points, there are at most $1/\epsilon$ clusters, so the runtime can also be phrased as $O(n/\epsilon)$.

Grading comments This part was worth 12 marks total, with 6 marks allotted for a correct and efficient algorithm, 4 for justification of correctness, and 2 for runtime analysis. Some students omitted the latter two parts and were duly penalized, since both are expected in this course.

Correct algorithms which were slightly inefficient (e.g $O(n^2)$ or similar) werededucted one mark independent of analysis; this often occurred if all pairs of distances were computed, or a disjoint-set structure used to find the actual clusters (both were unnecessary). More inefficient or incorrect algorithms were given at most 3 marks.

Several students noted that this was a variant of Hierarchical Agglomerative Clustering as discussed in class, which is true; however, justification of correctness was still necessary since the latter is an approximation algorithm. This observation was not required for full credit.

(b) Given a particular cluster C_i, if you sample t points uniformly at random (with replacement), give an upper bound in terms of ϵ for the probability that none of these points are in C_i.

Solution:

Solution The number of points in C_i is at least ϵn, so the probability that a point sampled u.a.r. lies within C_i is at least $\epsilon n/n = \epsilon$. Therefore the probability that a point sampled u.a.r. *doesn't* lie within C_i is at most $1 - \epsilon$. Since the points are chosen independently, the probability that t points sampled u.a.r. don't lie within C_i is at most $\boxed{(1-\epsilon)^t}$.

Grading comments This problem was relatively straightforward; the full 8 marks were given for the correct answer. An alternative (slightly weaker) bound using Chernoff was also given full credit as long as it was computed correctly. When the answer was incorrect, part marks were allotted for similarity to the solution and evidence of work.

(c) Give a sublinear-time algorithm that takes as input r and ϵ, and outputs a number of clusters k that is correct with probability at least $2/3$.

Solution:

Executive Summary Use the algorithm from part (a) on a subset of the points chosen u.a.r. The size of the subset is chosen so that the algorithm is correct with the desired probability.

Algorithm Initialize S' to be a subset of S of size t (to be determined below), where the points are chosen uniformly at random. Run the same algorithm as in part (a) on this subset.

Correctness From part (b), the probability that t points sampled u.a.r. don't lie in a given cluster is $(1-\epsilon)^t$. Hence, by the union bound, the probability that t points sampled u.a.r. don't lie in *any* cluster is at most $k(1-\epsilon)^t$; since $k \le 1/\epsilon$, this can be upper bounded by $\frac{1}{\epsilon}(1-\epsilon)^t$. The algorithm is incorrect iff the t sampled points miss any cluster, so we want this probability to be at most one third:

$$\begin{aligned}
\frac{1}{\epsilon}(1-\epsilon)^t &\le \frac{1}{3} \\
(1-\epsilon)^t &\le \frac{\epsilon}{3} \\
t\log(1-\epsilon) &\le \log\frac{\epsilon}{3} \\
t &\ge \frac{\log\frac{\epsilon}{3}}{\log(1-\epsilon)}
\end{aligned}$$

(the sign is reversed in the last inequality since $\log(1-\epsilon)$ is negative). The rest of justification of correctness follows from part (a).

Runtime analysis The runtime of this algorithm is $O(tk)$ or $O(t/\epsilon)$; the analysis is similar to that for part (a). Since neither t nor ϵ depend on n, this algorithm is sublinear.

Grading comments This part was worth 10 marks total, with 6 marks for achieving the correct approach to the sublinear algorithm and 4 marks for computing the size of the subset correctly.

Some students failed to realize that part (b) gives the probability of missing a *fixed* cluster, whereas this part requires the probability of missing *any* cluster; this resulted in a deduction of 2 marks. Others erroneously assumed that the events of missing each cluster were independent; this resulted in a deduction of 1 mark. Incorrect algorithms were given partial credit depending on their similarity with the correct solution.

Several students noted that this was a variant of the sublinear Connected Components algorithm as discussed in class; again, while true, this was not necessary and a correct calculation of the subset size was still required.

Problem 6. Looking for a Bacterium [30 points] (2 parts)

Imagine you want to find a bacterium in a one dimensional region divided into n 1-μm regions. We want to find in which 1-μm region the bacterium lives. A microscope of resolution r lets you query regions 1 to r, $r+1$ to $2r$, etc. and tells you whether the bacteria is inside that region. Each time we operate a microscope to query one region, we need to pay (n/r) dollars for electricity (microscopes with more precise resolution take more electricity). We also need to pay an n-dollar fixed cost for each type of microscope we decide to use (independent of r).

(a) Suppose that you decide to purchase $\ell = \lg n$ microscopes, with resolutions $r_1, \ldots, r_\ell$ such that $r_i = 2^{i-1}$. Give an algorithm for using these microscopes to find the bacterium, and analyze the combined cost of the queries and microscopes. **Solution:**

Executive Summary We will use all the microscope to perform a binary search. Starting with the lowest resolution microscope, we use successively more powerful microscopes and eventually using the microscope of resolution 1 to locate exactly where the bacterium is.

Algorithm We will assume n is a power of 2, if not, we can imagine expanding to a larger region that is a power of 2. This will not affect the asymptotic cost. We first use a microscope of resolution $n/2$ to query the region $(1, n/2)$. Regardless of whether the answer is yes or no, we can eliminate a region of size $n/2$. We then do the same with a microscope of resolution $n/4$, and so on.

Correctness This is essentially the same as a binary search, we use each type of microscope exactly once, and eventually we will pin point the location of the bacteria to a region of size 1.

Cost analysis We are only interested in the total electricity cost for this problem. Some students provided runtime analysis, but their correctness were not graded. We use each microscope exactly once, starting from the the one with the lowest resolution, it costs us 2, 4, 8, ... n to use each microscope. The last cost of n comes from using the microscope of resolution 1. This geometric series sums to a total of $2n - 2$, thus the electricity cost is $O(n)$. The total cost of purchasing the microscope is $n \lg n$, for a combined cost of $O(n \lg n)$.

Grading comments The maximum points for this part is 10. 3 points are awarded for people who can roughly describe the algorithm. 5 to 7 points are awarded for people who fail to correctly analyze the cost of electricity to various degrees. Points are also deducted if you simply assert the total cost of electricity is $O(n)$, we are looking for at least a mentioning of things such as this geometric sum converges.

Most students did pretty well on this part, some students attempted to use recurrence to solve for the cost of electricity, but most of them made mistakes.

(b) Give a set of microscopes and an algorithm to find the bacterium with total cost $O(n \lg \lg n)$.

Solution:

Executive Summary We borrow our idea from the vEB data structure. We will purchase microscopes of resolutions $n^{1/2}$,$n^{1/4}$,$n^{1/8}$, ..., $n^{1/2^i} = 2$. Clearly, $i = \lg \lg n$. So we spend $n \lg \lg n$ for buying microscopes. We do a linear search at each level, and we will show that the total cost of electricity is also $n \lg \lg n$.

Algorithm We perform a search for the bacterium as follows. First we use the microscope with resolution $\sqrt{n}$ to locate which region of size $\sqrt{n}$ the bacterium is in. This will take at most $\sqrt{n}$ applications of this microscope. We can then narrow our search in the region of size $\sqrt{n}$ and use microscopes of resolution $n^{1/4}$ to continue this process, eventually we can narrow down the location of the bacterium to a region of size 2. From there, it takes one simple application of a microscope of resolution 1 to find the bacterium.

Correctness This collection of microscopes allows us to search every region exhaustsively, thus we can always find the bacterium.

Cost analysis The total purchase of $\lg \lg n$ microscopes will cost us $n \lg \lg n$ dollars. At stage 1, we use the microscope of resolution $\sqrt{n}$ a total of $\sqrt{n}$ times. Each use of this microscope costs $\frac{n}{\sqrt{n}}$ dollars, for a total of n dollars. In fact, at every subsequent step, we always apply the microscope of resolution $\sqrt{r}$ a total of $\sqrt{r}$ times to completely scan a region of size r. The total cost of this is always n. Thus, the total electricity cost associated with using each type of microscope is at most n. Therefore, the total electricity cost os also $n \lg \lg n$. Our combined total cost is thus $O(n \lg \lg n)$.

Grading comments The maximum points for this part is 20. 2 points are given for some very vague mentioning of vEB, demonstrating some level of understanding. 5 points are given for the correct choice of microscope resolutions (but not how to use them to search for the bacterium). 10 points are given if the algorithm is correct (correct choice of resolutions as well as a description of how to use them to search for the bacterium). 13 points are awarded if the electricity cost is analyzed incorrectly, but has some correct ideas. 16 to 18 points are awarded if there are some minor mistakes in the electricity cost analysis. Many students attempted to analyze the electricity cost using recurrence, arriving at $C(n) = C(\sqrt{n}) + O(n)$. The idea is that the cost to search a region of size n is equal to spending $O(n)$ cost to apply the microscope of resolution $\sqrt{n}$ a total of $\sqrt{n}$ times, and then recurse on a region of size $\sqrt{n}$. This problem with this approach is that the $O(n)$ term is a constant. Every level costs at most n dollars, but this cost does not scale down with region size. Therefore, the correct way to setup the recurrence is to distinguish the variables from the constants, as $C(k) = C(\sqrt{k}) + n$. this indeed gives us an answer of $C(k) = n \lg \lg k$.