Final

1 True/False - Pre-MT2

(3 points each)

(a) $10000n^2 = O(n^2 \log n)$.

Solution: True. This is a freebie but $\lim_{n\to\infty} \frac{n^2 \log n}{10000n^2} = \lim_{n\to\infty} \frac{\log n}{10000}$ grows faster than n^2.

(b) e^{cn} is $O(e^n)$ for all $c > 0$.

Solution: False, because $e^{cn} = e^n e^{(c-1)n}$ and $e^{(c-1)n}$ is not $O(1)$ for $c > 1$.

(c) For $T(n) = 16T(n/9) + n^{3/2} \log n$, $T(n) = \Theta(n^{3/2} \log n)$

Solution: True. Using a master theorem-like analysis, the first level of the recursion tree asymptotically dominates the work (the work done in the subsequent levels decreases by a factor of at least $16/27$ per level).

(d) It suffices to choose $1 + \max(\deg(f(x)), \deg(g(x)))$ points if we use Fast Fourier Transform to get the coefficients of $p(x)$ where $p(x)=f(x) \cdot g(x)$.

Solution: False. We need at least $1+deg(f(x)) + deg(g(x))$ points for FFT (and better to choose a power of 2).

(e) The node with the highest **post**-order number in a depth first search of a directed graph must be in a **source** SCC.

Solution: True. This is the basis for the strongly connected components algorithm.

(f) The node with the highest **pre**-order number in a depth first search of a directed graph must be in a **sink** SCC.

Solution: False. Consider the dag on A, B, C, D consisting of edges $(A, B), (A, C), (B, C), (B, D)$. Depth first search starting at A would have either B or C have the highest pre-order number.

(g) Any connected undirected graph where depth first search does not find a back edge is a tree.

Solution: True. A connected graph with no cycles is a tree, and any cycle induces a back edge in a depth first search.

(h) Let G be an undirected graph with edge weights w, $f(\cdot)$ be some strictly increasing function, $F(G)$ be the graph G with edge weights w replaced with $f(w)$, and $W(G)$ denote the total weight of a graph G.

(i) Let T be an MST of a graph G. $F(T)$ is an MST of $F(G)$.

Solution: True. Appeal to Prim's or Kruskal's and the fact that the transformation preserves edge orderings.

(ii) Let T_1 and T_2 be spanning trees of G. If $W(T_1) > W(T_2)$ then $W(F(T_1)) > W(F(T_2))$.

Solution: False. Consider $f(w) = 2^w$ and spanning trees with weights $(1,3,3)$ and $(1,1,4)$.

(iii) Let T be a maximum spanning tree of a graph G. $F(T)$ is a maximum spanning tree of $F(G)$.

Solution: True. Appeal to Prim's or Kruskal's and the fact that the transformation preserves edge orderings.

(i) Recall that in a two person zero sum game with payoff matrix A, the entry $a_{i,j}$ in A is the payoff if the row player plays strategy i and the column player plays strategy j. We say a column j is dominating if for each row i, $a_{i,j} < a_{i,j'}$ for $j' \neq j$. (i.e., for any option the row player picks, the payoff is minimized by picking column j.)

(i) If there is a dominating column, then there is a pure strategy which is optimal for the column player.

Solution: True. This is clearly the best offense against any row strategy. And for every row payoff, adding any other column to the mixture, can only make it better for that row.

(ii) If there is a dominating column, there is a pair of pure strategies which are simultaneously optimal for the row and column player.

Solution: True. The column player can choose its dominating strategy regardless of the row strategy, thus the row can just pick the largest payoff in the dominating column strategy.

2 True/False - Post-MT2

(3 points each)

(a) If problem A reduces to problem B, then B can be solved in polynomial time **only if** A can be solved in polynomial time.

Solution: True. "P **only if** Q" means $P \implies Q$, and if A reduces to B, then if B has a polynomial time algorithm we can construct a polynomial time algorithm for A using the reduction.

(b) If we can prove an NP-hard problem is in P, then P = NP.

Solution: True. NP-hard problems are at least as hard as all NP problems, so a P problem would be at least as hard as all NP problems.

(c) If P $\neq$ NP, Vertex Cover can be reduced to Bipartite Matching.

Solution: False. Independent Set is NP-Complete, while Bipartite Matching is P.

(d) Notice that for any Travelling Salesman Instance, we can view the input as a weighted complete graph where edge (u, v) has weight $d(u, v) \geq 0$. The weight of the minimum spanning tree in this graph is a lower bound on the cost of the optimal Travelling Salesman Tour.

Solution: True. The tour forms a connected graph where the MST is the minimum cost connected graph.

(e) If for some $a \neq 0 \pmod N$, $a^{N-1} \neq 1 \pmod N$, then N is not prime.

Solution: True. This is the contrapositive of Fermat's theorem. This certificate for non-primality is useful in devising procedures to test for primality.

(f) Any two-qubit quantum state can be decomposed into two one-qubit states.

Solution: False. For example, $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$.

(g) In the experts problem with n experts, let A be *any* algorithm which only picks predictions made by an expert that has made no mistakes. A makes $O(\log n)$ mistakes if there is an expert who makes no mistakes.

Solution: False. An algorithm which picks an arbitrary expert that has made no mistakes could still make $n-1$ mistakes, if every round they pick the only expert to choose one option and that expert is wrong. In this case, the algorithm only eliminates one expert per day, so it takes $n-1$ days in the worst case to find the best expert.

(h) In the weighted majority algorithm, we multiply an expert's weight by $1-\epsilon$ anytime they make a mistake. To minimize the upper bound on our regret, we should set ϵ to be relatively small if the number of experts is small and the number of mistakes the best expert makes is large.

Solution: True. The number of mistakes made by this algorithm is bounded by $2(1+\epsilon)m + \frac{\ln n}{\epsilon}$ where n is the number of experts, m is the number of mistakes made by the best expert. When m is large, n is small, we want to make the $(1+\epsilon)$ multiplier small and are okay with making the $\frac{1}{\epsilon}$ multiplier large since it's attached to a tiny $\ln n$ term.

(i) Suppose we modify the weighted majority algorithm so that it multiplies an expert's weight by $1/(1-\epsilon)$ every time the expert is correct instead of multiplying their weight by $1-\epsilon$ every time the expert makes a mistake. Then the algorithm achieves the same guarantee.

Solution: True. Multiplying all the weights by the same number doesn't affect the algorithm, and this is the same as multiplying all weights by $1/(1-\epsilon)$ after each step in the algorithm.

3 Short Answer - Pre-MT2

(4 points each)

(a) Let ω be a primitive nth root of unity for an even number n, and let $S = \{\omega^0, \omega^1, \ldots, \omega^{n-1}\}$. How big is the set $\{x^2 \mid x \in S\}$?

Solution: $n/2$. Each image is an $n/2$th root of unity of which there are $n/2$. Moreover, every $n/2$th root of unity has a square root that is a nth root of unity. This is the key to the fast fourier transform working.

(b) We have an array of n integers A where n is a power of 2. We want to find $f(A) = \max_{j<k}(A[k] - A[j])$, i.e. the maximum of any element minus an element to its left, using a divide and conquer algorithm.

Let L be the left half of A, and R be the right half. Suppose our algorithm has recursively computed $f(L), f(R)$. The algorithm should finish by computing and outputting _____. Write your answer in terms of $f(L), f(R)$ and the elements of L and R.

Solution: $\max\{f(L), f(R), \max_i R[i] - \min_i L[i]\}$. There are three cases. Either i, j that define $f(A)$ are both on the left half, both on the right half, or i is in the left half and j is in the right half. In the first two cases, $f(A) = f(L)$ or $f(R)$. In the last case, $L[i]$ should be the minimum element in L and $R[i]$ should be the maximum element to maximize the difference.

(c) Let $d = 2^k - 1$ for positive integer k, and let F, G, and H be polynomials of degree at most d satisfying $F(x)/G(x) = H(x)$. Suppose for any 2^k-th root of unity z that $G(z) \neq 0$. Briefly explain how to compute the coefficients of H given the coefficients of F and G in $O(d \log d)$ time.

Solution:

Compute the FFT of F, G to get $F(\omega^i), G(\omega^i)$ for $i = 0$ to d for the $d+1$th root of unity ω. Compute the vector whose entries are $F(\omega^i)/G(\omega^i)$. Compute the IFFT of this vector to get the coefficients of H.

(d) We have an undirected graph $G = (V, E)$. With probability $0 \leq p_e < 1$, each edge e will independently be deleted from the graph. Given $s, t \in V$, we want to find a path from s to t with the maximum probability of existing in the graph after deletions. In particular, we want to do this by weighting the edges in G and running a shortest path algorithm.

(i) What should be the weight of edge e in terms of the deletion probability p_e?

Solution: $-\ln(1 - p_e)$. Then, the shortest path algorithm finds P that minimizes $\sum_{e \in P} -\ln(1 - p_e)$, i.e. maximizes $\prod_{e \in P}(1 - p_e)$, which is the probability no edge in the path is deleted.

(ii) Which shortest path algorithm should we run?

Solution: Dijkstra's. The edge weights $-\ln(1 - p_e)$ are non-negative because $0 < 1 - p_e \leq 1$, so both will return the shortest path, but Dijkstra's is faster in this instance.

○ Dijkstra's	○ Bellman-Ford

(e) We have an undirected weighted graph $G = (V, E)$. For sets of vertices $S, T \subseteq V$, we want to find the shortest path from any vertex in S to any vertex in T. In particular, we would like to do this by adding new vertices and edges to G to get a graph G' and running a shortest path algorithm on G'.

(i) What *new* vertices and edges should we add to G to get G'?

Vertices:

Solution: Two new vertices s and t

Edges:

Solution: An edge from s to every vertex in S, and from t to every vertex in T

(ii) What weight should the new edges have?

Solution: The intended answer is 0, but technically any non-negative value that is fixed for all the edges is correct. This way the algorithm doesn't care which vertex in S it starts from or which vertex in T it ends in.

(f) What is the average bit length per character in the optimal encoding for the following set of characters and frequencies: $(C, .7), (T, .2), (G, .05), (A, .05)$?

Solution:

1.4 which is $.7(1) + .2(2) + .1(3)$, since characters G and A are merged first and will be encoded with three characters, and then T will be merged with this subtree and will be encoded with one character, and finally C will be merged and will be encoded with 1 character.

(g) Recall that an increasing subsequence is a sequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ where $i_j < i_{j+1}$ and $a_{i_j} < a_{i_{j+1}}$. What is the recurrence for the Longest Increasing Subsequence problem on input $a_1, \ldots, a_n$? Write your answer in terms of subproblems $L(i)$, where $L(i)$ is the length of the longest increasing subsequence ending at i.

Solution: $L(i) = \max_{j<i, a_j<a_i} L(j) + 1$. The longest increasing subsequence that ends at i, must include a_i and a longest increasing subsequence ending at some a_j for some $j \leq i$ and $a_j < a_i$.

(h) Consider the problem of finding the number of paths (with repeated edges allowed) from s to t with 2^k edges in an (unweighted) graph G. In a dynamic program, we define the subproblems $C(u, v, i)$ to be the number of paths from u to v of length 2^i.

(i) Give a recurrence for $C(u, v, i)$.

Solution: $C(u, v, i) = \sum_{w \in V} C(u, w, i-1)C(w, v, i-1)$. A path of length 2^i is two paths length 2^{i-1}. The recurrence sums over each possible intermediate endpoint and multiplies the number of choices for the first path and the number of choices of the second.

(ii) Using this idea, give a runtime for computing the number of paths from s to t in terms of the number of vertices, n, the number of edges, m, and k (You may assume that arithmetic operations can be done in $O(1)$ time.)

Solution: $O(n^3 k)$. The table has size $O(n^2 k)$ and it takes n time to fill in the entries. Technically, the number of paths could be exponential and then the number of bits is $O(n)$ which makes the arithmetic operations much more costly.

(i) Given the payoff matrix $\begin{bmatrix} 2 & -1 \\ 1 & 3 \end{bmatrix}$ for a two person game, what is the best defense strategy for the row player? (Specify as a vector.)

Solution: $[2/5, 3/5]$. Set both column payoffs to be equal $2x_1 + 1(1 - x_1) = -1x_1 + 3(1 - x_1) \implies x_1 + 1 = 3 - 4x_1$ or $5x_1 = 2$.

(j) You are given a flow network $G = (V, E)$ with edge capacities $c(\cdot)$ and maximum flow value of F, and a valid flow $f(\cdot)$. What is the value of the minimum s-t cut in the **residual network** in terms of F and the amount of flow, $|f|$, routed by $f(\cdot)$ from s to t?

Solution: $F - |f|$. The sum of $f(\cdot)$ and the maximum flow in the residual network yields a maximum flow for the original flow problem, and thus the maximum flow in the residual network has value $F - |f|$ which is then the value of the min-cut in this network.

(k) Consider a linear program $\max c^T x$, s.t. $Ax \leq b, x \geq 0$. For x and y being feasible primal and dual solutions the value of the vector $y^T b - c^T x$ is:

$\circ \geq 0$	$\circ \leq 0$	$\circ = 0$	$\circ$ any real number

Solution: ≥ 0. Since the primal is a maximization problem, any primal solution value is upper bounded by any dual solution value.

4 Short Answer - Post-MT2

(4 points each)

(a) A travelling salesman tour can be viewed as an ordering of the n cities. In a local search approach, what is the number of possible moves from a fixed solution where a move swaps two vertices in the ordering?

Solution: $\binom{n}{2}$. This is just the number of pairs.

(b) Let $|\psi\rangle := \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$, the equal superposition of all n-bit strings. After measuring the first k qubits of $|\psi\rangle$, how many possible outcomes are there if we then measure the whole of $|\psi\rangle$?

Solution: 2^{n-k}. Once we measure the first k bits of $|\psi\rangle$ and get outcome y, $|\psi\rangle$ collapses to the state $\frac{1}{\sqrt{2^{n-k}}} \sum_{x \in \{0,1\}^{n-k}} |y\rangle\,|x\rangle$.

(If asked, clarify to student that we measure in the computational basis $|0\rangle, |1\rangle$.

(c) How many elements of $\{1, \ldots, 48\}$ have a multiplicative inverse modulo 49?

Solution: 42. Everything that is relatively prime to 7 (or 49.)

(d) We are using the weighted majority algorithm for the experts problem. Suppose there are n experts, we multiply an expert's weight by $(1-\epsilon)$ anytime they are wrong, and we make the wrong decision for the first t days. Give an upper bound on the total weight of all experts at the end of day t.

Solution: $n(1-(\epsilon/2))^k$. Anytime we make a mistake, we know that experts consisting of at least half the total weight have their weights multiplied by $(1-\epsilon)$, i.e. the total weight of all experts is multiplied by at most $(1-(\epsilon/2))$. So after k days, the total weight (initially n) is at most $n(1-(\epsilon/2))^k$.

(e) Recall that with n experts and 1 perfect expert that the algorithm of following the advice of the majority of experts who have not made a mistake ensures that we don't make more than $\log n$ mistakes. Give an improved bound on the number of mistakes if k of the experts are perfect.

Solution: $\log \frac{n}{2k-1}$. After this many mistakes, there are $2k-1$ experts left who have not made a mistake, and the k experts who never make mistakes are the majority.

5 Reductions

(a) (**8 points**) Consider a weighted directed graph $G = (V, E)$ with integer edge weights. In the minimum cycle cover problem (call this problem **MCC**) we want to find the minimum weight set of simple cycles where every vertex participates in exactly one cycle. Give a reduction from this problem to the problem of finding a perfect matching of minimum weight in a bipartite graph (call this problem **MWPBM**).

(i) Fill in the blank: We take an instance of _____
and create an instance of _____

(ii) Describe the graph created by the reduction:
Vertices:
Edges:

(iii) Given a solution to the reduced instance, describe how to retrieve a solution to the original instance:

Solution: Create a bipartite graph with vertices l_v, r_v for each $v \in V$. Add the edge (l_u, r_v) with weight $w_{u,v}$ for all edges $(u, v) \in E$ to this graph. Given a solution to MWPBM, we get a solution to MCC consisting of the edge (u, v) for each (l_u, r_v) in the MWPBM solution. Clearly the weight of the solutions are the same, and since the MWPBM solution is a perfect bipartite matching, the in-degree and out-degree of every vertex in the MCC solution is 1, i.e. the solution consists of cycles and includes all the vertices.

(b) (**4 points**) Notice that a travelling salesman tour is a cycle cover in the weighted complete graph on the cities. Does this imply that minimum cycle cover is NP-hard? Briefly justify your answer.

Solution: No. The minimum weight cycle cover may not be connected (there might be multiple disjoint cycles in the minimum weight cycle cover).

(c) (**8 points**) Given a directed graph G with weighted edges, the Shortest Simple Path problem is to find the shortest simple path between vertices s and t. The graph may have negative edges and/or negative cycles. **Recall that simple paths do not repeat vertices.** Show that Simple Shortest Path is NP-Hard by reducing from Rudrata Path.

(Recall that a Rudrata path is a simple path with arbitrary endpoints that includes all the vertices.)

(i) Fill in the blank: We take an instance of _____
and create an instance of _____

(ii) Describe the graph created by the reduction:
Vertices:
Edges:

(iii) Given a solution to the reduced instance, describe how to retrieve a solution to the original instance:

Solution: Given an undirected graph G', construct graph G where each edge from G' produces two directed edges in G with weight -1, and two additional vertices s and t where s has outgoing arcs of weight 0 to every vertex and t has incoming edges of weight 0 from every vertex.

If the shortest path from s to t has weight $1 - |V|$ then there is a path in G' that visits every vertex corresponding to a vertex in the original graph exactly once and thus corresponds to a Rudrata Path in G. Similarly, a Rudrata path in G corresponds to a path of cost $1 - |V|$ in G' between two vertices which can then be extended to a path between s and t.

(d) (**8 points**) A k-bounded spanning tree for an undirected graph G is a spanning tree T such that each vertex has at most k neighbors in T. For integer $k \geq 2$ the **k-Bounded Spanning Tree Problem** is: Given a graph G, does a k-bounded spanning tree exist? Notice that when $k = 2$, this would be the Rudrata Path problem.

Give a reduction from Rudrata Path to 10**-bounded spanning tree**.

(i) Fill in the blank: We take an instance of _____
and create an instance of _____

(ii) Describe the graph created by the reduction:
Vertices:
Edges:

(iii) Given a solution to the reduced instance, describe how to retrieve a solution to the original instance:

Solution: Main Idea: For each vertex in G, add and connect 8 dummy vertices to make a graph H. A Rudrata Path then exists iff H has a 10-bounded spanning tree.

Proof of Correctness: First we will show that if a 10-bounded spanning tree exists, then a Rudrata path exists. Notice that all dummy edges are in T. Since each vertex has 8 dummy vertices, removing these gives a 2-bounded spanning tree. This is a Rudrata path.

We will now show that if a Rudrata path exists, then a 10-bounded spanning tree exists. It is known that each vertex has at most 2 neighbours in a Rudrata path. We add 8 neighbours to every vertex in the path, giving a maximum of 8 neighbours. This is a 8-bounded spanning tree.

6 Universal hash functions.

For this problem, we define a family $\mathcal{H}$ of hash functions from S to T to be *universal* if $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] = \frac{1}{|T|}$.

(a) (**6 points**) Consider a universal hash family, $\mathcal{H}$, of hash functions from $\{0, 1, \ldots, m-1\}$ to $\{0, 1, \ldots, n-1\}$.

(i) Given a subset $S \subset \{0, 1, \ldots m-1\}$, give a reasonable upper bound for $\Pr_{h\in\mathcal{H}}[\exists x, y \in S, h(x) = h(y)]$ in terms of $|S|$ and n.

Solution: $\binom{|S|}{2}\frac{1}{n}$. This is the union bound over the $\binom{|S|}{2}$ choices of x and y.

Also acceptable is $\prod_{i=1}^{|S|}\frac{n-i+1}{n}$, since by a union bound and universality, the ith element in S has probability $\frac{i-1}{n}$ of colliding with any of the previous elements.

(ii) To estimate the size of S, we hash all the values and check how many buckets get items. Let B be the number of buckets which are not empty. If $|S| \leq k$, what should n be so that $\Pr_{h\in\mathcal{H}}[B = |S|] \geq 1/2$? (Don't worry about small additive constants)

Solution: $n \geq k^2$. This makes the above probability at most $1/2$.

(b) (**6 points**) For each of these hash function families, state if the family is universal. If so, explain why. If not, for some value of m give an example of two inputs that collide with probability greater than $1/m$.

(i) The family containing $h_{a_1,a_2,a_3}(x_1, x_2) = a_1x_1 + a_2x_2 + a_3x_1 \mod m$ for each $a_1, a_2, a_3 \in \{0, \ldots, m-1\}$, where m is prime and x_1, x_2 are in $\{0, \ldots, m-1\}$.

Solution: Universal. This can be rewritten as $h_{a_1,a_2,a_3}(x_1, x_2) = (a_1 + a_3)x_1 + a_2x_2 \mod m$. Every value of $a' = a_1 + a_3 \mod m$ occurs with the same frequency in this family, so the family is essentially the same as $h_{a',a_2}(x_1, x_2) = a'x_1 + a_2x_2 \mod m$ which we know is universal.

(ii) The family containing $h_{a_1,a_2}(x_1, x_2, x_3) = a_1x_1 + a_2x_2 + a_1x_3 \mod m$ for each $a_1, a_2 \in \{0, 1, \ldots m-1\}$, where m is prime and x_1, x_2, x_3 are in $\{0, 1, \ldots m-1\}$. (Note that the third term in the definition of h has changed).

Solution: Not universal. $h(1, 0, 0)$ and $h(0, 0, 1)$ both always equal a_1.

7 Approximately Finding the Median of a Stream

We want to design a space-efficient algorithm that scans a stream of m integers between 1 and n and outputs the median of the stream when queried. For simplicity, you may assume that m is odd in all parts of the problem. **No proof of correctness is required for any part.**

(a) (**4 points**) Suppose we have an algorithm A which exactly solves the problem, and the stream S has been scanned by A. Show how to retrieve a sorted copy of S from A by only appending new numbers between 1 and n to the input of A and repeatedly querying A for the median. For simplicity, you may assume you already know the length of S.

Solution: We first add $m-1$ copies of the number 1 to A's input. We then do the following m times: query the median and then add two copies of n. The list of outputs to the queries is the sorted copy of S.

After adding the $m-1$ copies of 1, the smallest element in S is the median of the stream passed to A. After we pass the first 2 copies of the number n to A, the second smallest element in S is the median of the stream passed to A. 2 more copies, and the third smallest element in S becomes the median, etc. So we retrieve a sorted copy of the original stream.

You will now design a *deterministic* algorithm that outputs an *approximate* answer.

(b) (**4 points**) Consider the simpler problem of outputting **Yes** if the median is a value k (fixed ahead of time), and **No** otherwise. Give a streaming algorithm which does this using at most $O(\log m)$ bits.

Solution: Keep track of three counters. The first tracks the total size of the stream, the second tracks the total number of integers less than k, and the third tracks the total number of integers greater than k. Each takes at most $\lceil \log m \rceil$ bits to store, since none of them ever count higher than m. When asked to output, output No if the value in either the second or third counter is at least half the value in the first counter, and Yes otherwise.

(There are multiple answers, but they all use similar ideas. Once can show that any answer needs to use at least $2 \log m - O(1)$ bits, so an answer using a single counter that counts up to m cannot be correct.)

(c) (**4 points**) Give an algorithm which uses $O(\log m \log n)$ bits which when queried outputs a number which is at least the median and at most twice the median.

Solution: One answer is to keep $\log n$ counters which count the number of elements in each of $[1,2), [2,4), \ldots [n/2, n]$. When queried, we just have to find which interval $[i/2, i]$ satisfies that less than half the elements fall in the intervals $[1, i/2]$ and $[i, n]$, which can be done just using these $\log n$ counters, and output i.

Another answer: Part b can easily be modified to answer the question "Is the median in $[k, 2k]$" instead of "Is the median k" by having the third counter be the number of items greater than $2k$. Then, we can use $\log n$ copies of this algorithm to determine if the median is in $[1,2], [2,4], \ldots [n/2, n]$. When queried, we query all the copies of the algorithm from part b until we find an interval the median is in, and output the right endpoint of this interval.

8 Not quite infallible experts.

We are solving the experts problem with n experts, and we know there is a true expert who will make **strictly fewer** than c mistakes for some constant c.

Suppose we run the majority algorithm, but only remove an expert from the set of trusted experts once they have made c mistakes.

(a) (**4 points**) Let ϕ_i be c minus the number of mistakes made by expert i, or 0 if expert i has made c or more mistakes (i.e, $\phi_i \geq 0$ always). Let $\phi = \sum_{i=1}^{n} \phi_i$. Any time this algorithm makes a mistake, by at least what multiplicative factor does ϕ decrease?

(Hint: If the number of trusted experts remaining is t, what's an upper bound on ϕ?)

Solution: $(1 - \frac{1}{2c})$. If there are t trusted experts left, $\phi \leq ct$, and a mistake causes ϕ to decrease by at least $t/2$ since $t/2$ trusted experts must make mistakes for us to make a mistake. In turn, any mistake decreases ϕ multiplicatively by at least $(1 - \frac{1}{2c})$.

(We accepted any reasonable interpretation of "by what factor does ϕ decrease", so $\frac{1}{2c}$ and $\frac{1}{1-\frac{1}{2c}} = \frac{2c}{2c-1}$ are also acceptable)

(b) (**4 points**) Using your answer to the previous part, give an upper bound on the number of mistakes made by this algorithm. (Hint: Use the fact that $(1 - 1/b)^a \leq e^{-a/b}$).

Solution: After making m mistakes ϕ has decreased multiplicatively by at most $(1 - \frac{1}{2c})^m$. Initially $\phi = cn$, and ϕ will always be at least 1 since there is one expert who never makes more than c mistakes, so we get $cn(1 - \frac{1}{2c})^m \geq \phi \geq 1$, where m is the number of mistakes made by the algorithm. Using that $(1 - \frac{1}{2c})^m \leq e^{-\frac{m}{2c}}$ and then solving for m gives $m \leq 2c \ln(cn)$.

(If one doesn't use the hint, solving for m gives $m \leq \frac{\log cn}{\log \frac{1}{1-1/2c}} = \log_{\frac{1}{1-1/2c}}(cn)$ which is also acceptable).

9 Faster Longest Increasing Subsequence.

For input $a_1, \ldots, a_n$ an increasing subsequence is a sequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ where $i_j < i_{j+1}$ and $a_{i_j} < a_{i_{j+1}}$. The Longest Increasing Subsequence (LIS) problem is to find an increasing subsequence of maximum length. For the sake of convenience, we assume that all the a_i's are distinct.

A solution to the LIS problem can be found by playing the patience game on a sequence $a_1, \ldots, a_n$ as follows.

1. Place the first element of the sequence in a pile.
2. For each subsequent element a_j, place it on the leftmost pile for which a_j is less than the top card on the pile. If no such pile exists, a_j goes in a new pile to the right of all other piles.
3. Report the number of piles as the length of the longest increasing subsequence.

For example, if we run the algorithm on the sequence $S = 7, 4, 6, 8, 1$, we obtain piles $(7, 4, 1), (6), (8)$.

(a) (**3 points**) **Briefly** describe an **efficient** implementation of this algorithm and state its runtime.

Solution: $O(n \log n)$. Each addition of the element can be done using binary search as the top cards are always in increasing order from left to right as when something is added it is larger than the item to the left by the algorithm and it is smaller than the element on right which was larger than the previous element on the pile.

For the subsequent parts, we consider the following augmentation to the algorithm. We add a backpointer from a_i to the top element of the pile to the left *at the time that a_i is placed.* No backpointers are recorded for elements in the first pile.

For example, if we run the algorithm on the sequence $S = 7, 4, 6, 8, 1$ with $(7, 4, 1), (6), (8)$, the backpointer from 8 goes to 6, and the backpointer from 6 goes to 4. Notice that following the backpointers from 8, gives the sequence 8,6,4 whose reverse corresponds to an increasing subsequence of S.

(b) (**4 points**) Prove that the sequence of backpointers form a decreasing sequence and that its reverse forms an increasing sequence in $a_1, \ldots, a_n$.

Solution: Each pointer from an element a_j points to a larger a_k since a_j was not placed on a_k's pile. Furthermore $k < j$ since it was processed previously by the algorithm which processes the sequence $a_1, \ldots, a_n$ in order.

(c) (**5 points**) Prove that for any element a_j in the ℓ-th pile from the left, the length of the longest increasing subsequence ending at a_j is ℓ.

Solution: We do this by induction on index of an element. This is certainly true for the first element which in in pile 1.

When we place an element a_i in a pile L it points to the top element of the previous pile. The back-pointers give an increasing sequence of length L as argued in the previous part.

Note that any element of a previous pile has a longest subsequence ending at that element of length at most $L-1$ by the induction hypothesis. Thus there is no better predecessor element in the previous piles.

Any element in the pile L or a larger pile has value less than a_i so cannot be the predecessor for a_i in an increasing subsequence. Moroever, any element added to piles at later times cannot be a predecessor in the increasing subsequence.

Thus, L is the maximum length of any increasing subsequence ending at a_i is of length ℓ. Moreover, the backpointers provide an increasing subsequence of length L, which completes the proof.

Alternatively. One can observe that any pile is in decreasing order and a subsequence of the sequence. Thus, any two elements of a single pile cannot be in an increasing subsequence, and thus the longest increasing sequence is at most the number of piles. This is basically fine, but technically we ask to prove a (slightly) stronger statement which requires arguing that later piles can't be in an increasing sequence ending at earlier piles as argued above.