Alex Havrilla

Hwk 7

Task 1:

Claim: In any contiguous block trials the number of mistakes made is at most $O(m+log(n))$ where m is the number of mistakes made by the best expert in that block and n is the total number of experts.

Proof. Set the state Φ of a round to be $\sum_{i=1}^{n} w_i$. First we show on any round where we predict incorrectly that

$$\Phi_{new} \leq \frac{7}{8}\Phi_{old}$$

Let Let $P \cup N = [n]$ with $k \in P$ correctly predicting and $j \in N$ incorrectly prediting. Since we overall predicted incorrectly we know $\sum_{j\in N} w_j > \sum_{k\in P} w_k$. So it suffices to show $\sum_{j\in N} w_{j,new} \leq \frac{3}{4}\sum_{j\in N} w_{j,old} = \sum_{j\in N} w_j$ where $w_{j,new}$ corresponds to the weights after penalty.

Let $A \cup B = N$ where $a \in A \implies w_a \geq \frac{1}{4}\frac{1}{n}\Phi = \frac{1}{4}AVG$ and $b \in B \implies w_b < \frac{1}{4}AVG$. Then $\sum_{j\in N} w_j = \sum_{a\in A} w_a + \sum_{b\in B} w_b$.

Since we predicted incorrectly we know

$$\sum_{j\in N} w_j > \frac{1}{2}\sum_{i\in[n]} w_i$$

Further compute for $b \in B$,

$$w_b < \frac{1}{4}\frac{1}{n}\sum_{i\in[n]} w_i = \frac{1}{2}\frac{1}{n}\frac{1}{2}\sum_{i\in[n]} w_i < \frac{1}{2}\frac{1}{n}\sum_{j\in N} w_j$$

so

$$\sum_{b\in B} w_b < |B|\frac{1}{2}\frac{1}{n}\sum_{j\in N} w_j \leq \frac{1}{2}\sum_{j\in w_j} \implies \sum_{b\in B} w_b < \sum_{a\in A} w_a$$

and thus

$$\sum_{j\in N} w_{j,new} = \sum_{a\in A} \frac{1}{2} w_a + \sum_{b\in B} w_b \leq \frac{3}{4}\sum_{a\in A} w_a + \frac{3}{4}\sum_{b\in B} w_b = \frac{3}{4}\sum_{j\in N} w_J$$

as desired. This establishes

$$\sum_{i\in[n]} w_{i,new} = \Phi_{new} \leq \frac{7}{8}\Phi = \frac{7}{8}\sum_{i\in[n]} w_i$$

Let m be the number of mistakes the best expert makes and M be the total number of mistakes we make. Then we can bound over some number of contiguous rounds starting with state Φ_{init} and ending with Φ_{end} and $w_{i^*,s}, w_{i^*,e}$ the weight our best predictor at the start and end

$$\Phi_{end} \leq (\frac{7}{8})^M \Phi_{init}$$

and

$$\Phi_{end} \geq w_{i^*,e} \geq (\frac{1}{2})^m w_{i^*,s} \geq (\frac{1}{2})^m \frac{1}{8}\frac{1}{n}\Phi_{init}$$

where we use the fact that at no point can a weight ever be less than $\frac{1}{8}AVG$ since other wise it would have had to beeen halved when it was less than $\frac{1}{4}AVG$(a contradiction). This yields the inequality

$$(\frac{8}{7})^M \leq 2^{m+3}n \implies M \in O(m + log(n))$$

as desired

□

Task 2:

---

a)

Outline: First use binary search to determine pivot y with max height. Then fan triangulate P from y and determine wich cone/triangle x is in via binary search with line tests.

Let x be a point in the plane. We determine if it is in/on P in $O(log(n))$ time.

Fix $A[n/2] = p$. In $O(log(n))$ time I can find the point y with maximal height in P. This is done via a binary search starting at p. Compare the height of p to its neighbors. Then move in the direction(clockwise is right in the array and left is counterclockwise) with greater height. Note if all three heights are equal we know we are either at a global max or global min due to convexity. So it does not matter what direction we go in(so choose arbitrarily). Since this amounts to a binary search it runs in $O(log(n))$ time.

Wihtout loss of generality and for ease of writing suppose y is $A[0]$. Once at y we first check if x is "above" P by performing a line slide test with y and its clockwise neighbor a and counterclockwise neighbor b. If x is above or the the right of ya or above or to the left of by we know it is not in P. This can be done in constant time.

Otherwise we perform another binary search within the polygon by running the line test on x with y and $A[n/2]$. If x is the left we then check $yA[3n/4]$ and if the right $A[n/4]$. Thus in log time we determine which "cone" x resides in, with respect to the pivot y. Note the part of this cone intersecting P determines a triangle $yA[i]A[i+1]$. To determine if $x \in P$ we perform a final line test on $A[i]A[i+1]$. We know $x \in P$ it is on the same side as y. Otherwise it is outside. This completes in $O(log(n))$ time.

Note that if at any point we determine x is on a line L, it suffices to determine if x is inside P by choosing another line intersecting L and checking if x is on the same side as y. Note that if x is on the line $A[i]A[i+1]$ it must be inside(since we've already determined it's in the cone).

b)

We compute the number of triangles which do not contain Q in $O(n)$ time.

First in $O(n)$ preprocessing time we compute for each vertex $A[i]$ the index j s.t. $A[i]A[j]$ is to the left of Q but $A[i]A[j+1]$ is to the right(or

vice verse). Ie. we determine the cone Q resides in w.r.t each pivot $A[i]$ and identify it with the counterclockwise most line. This is done by locating the maximal vertex y as above and determining the cone for $yA[i]A[i+1]$ containing Q(note that we know this containment is strict since Q does not reside on any lines). Then to determine the cone for $A[1]$ we check first $A[1]A[i]A[i+1]$(can be done with two line tests). If not this then $A[1]A[i+1]A[i+2]$ until we find the contanining cone. Then continue for $A[2]$ until we reach $A[n-1]$ in this manner. Note that as go around the convex polygon the cone of containment also completes one full rotation ie. visits at most n different vertices. Thus although we may do more than a constant amount of work at some $A[i]$, in total we do $O(n)$ work.

Once we have this we count how many triangles do not include Q. At a vertex v we determine the cone/triangle containing Q in $O(1)$ time via preprocessing. Call this $v(i)(i+1)$. Then we the count the triangles with v as a vertex not intersecting this cone. This is done by counting the number of vertices clockwise of $i+1$(until we reach v) M and the number of vertices counterclockwise to i(until we reach v) N. Can be done in $O(1)$ time via looking at indices in the array since it is ordered clockwise. The total number of triangles as described is then given by $N(N-1)/2+M(M-1)/2$. Compute this for every point. Note that every triangle not containing Q gets counted twice this way, so divide the total count by 2. Further we do not undercount as every triangle not containing Q will be counted twice(via a convexity argument).

Task 3:

First note that two arcs $(a,b),(c,d)$ intersect $\iff$ $a<c<b<d$ or $c<a<d<b$(note this wouldn't be true if two endpoints coincided). This gives rise to a sweepline algorithm. Each arc $A_1,...,A_n$ generates a start and endpoint, giving a total of $2n$ x coordinates. First we sort these coordinates and then insert one at a time into a augmented tree(could be implemented

with splay trees) which keep track of the number of nodes in a subtree. If we insert a start point S do nothing. If we insert an end point E, we remove the corresponding start point S and check to see how many points are between this start point and end point by splitting the tree at the start point and end point(this can be done in $O(log(n))$ time). Then we can count the number of nodes in $O(1)$ time via augmentation. Note that this only counts start nodes whose end nodes are after the current end. We are are therefore counting preciesly the intersections involving the arc SE. Since we encounter $O(n)$ points and do at most $O(log(n))$ per encounter, this runs in $O(log(n))$ time. Further we know we do not overcount the number of interesections since each intersection can be identified with the "leftmost" arc. And for each arc we are counting the intersections in which it is the "leftmost" arc. Hence we do not overcount. We also do not undercount since we process every arc.