Solutions for 611 Homework 1

Mahim Agarwal, Hanwen Xiong, and Jackson Warley

September 22, 2017

1 Solution to Question 5

1.1 Algorithm

Useful definition:

- l_{max}: the line with maximum slope
- l_{min}: the line with minimum slope
- $l_{nextmax}$: the line with maximum slope in the rest of two subarrays
- $l_{nextmin}$: the line with minimum slope in the rest of two subarrays
- "visible area": $\{y >= l_{max}, y >= l_{min}\}$
- P_{ri}: x coordinate of intersection of l_i and l_{max}
- P_{li}: x coordinate of intersection of l_i and l_{min}

Algorithm:

1. Divide the array of lines into 2 halves evenly
2. Sort the two subarrays by slope in non-increasing order
3. Find the intersection of l_{max} and l_{min}, say(m, n)
4. Compare $l_{nextmax}(m)$ with n, if $l_{nextmax}(m)<n$, throw it; if $l_{nextmax}(m)>n$, put l_{max} into "left_visible" array and update $l_{max} := l_{nextmax}$. Go to step 3 if there's $l_{nextmax}$, put the rest 2 lines into "left_visible" array and go to next step when there's no lines left
5. Find the intersection of l_{min} and l_{max}, say(m, n)
6. Compare $l_{nextmin}(m)$ with n, if $l_{nextmin}(m)<n$, throw it; if $l_{nextmin}(m)>n$, put l_{min} into "right_visible" array and update $l_{min} := l_{nextmin}$. Go to step 5 if there's $l_{nextmin}$, put the rest 2 lines into "right_visible" array and go to next step when there's no lines left
7. "visible" array="left_visible" array $\cap$ "right_visible" array. Basically, "left_visible" array is in non-increasing order slope-wise and "right_visible" array is in non-decreasing order slope-wise. Thus, it's easy to do the intersection by traversing the two arrays reversely

1.2 Time Complexity

Divide: Since we divide the array into 2 subarrays and solve them recursively, we have 2 subproblems of size $n/2$

Conquer: We have to find at most $2(n-1)$ intersections and always $2(n-1)$ comparisons, which takes $\mathcal{O}(n)$. Besides, we have to intersect two arrays, which includes at most n comparisons, also $\mathcal{O}(n)$

So overall we have $T(n) = 2T(n/2)+\mathcal{O}(n)$, according to Master's Theorem we get $T(n) = \mathcal{O}(nlogn)$

1.3 Proof

Lemma 5.3.1: l_{max} and l_{min} are always "visible" within its correspoding scope(i.e. array)

Lemma 5.3.2: l is "visible" if and only if it dominates at the intersection of l_{max} and l_{min}

Lemma 5.3.3: A list of non-increasing(slope-wise) lines $\{l_1, l_2, ..., l_n\}$ are visible if and only if the every single line is "visible" in "visible area" and the list of $\{P_{r1}, P_{r2}, ..., P_{rn}\}$ is also in non-increasing order and the list of $\{P_{l1}, P_{l2}, ..., P_{ln}\}$ is in non-decreasing order

Proof of 5.3.1: Let's first consider l_{max}: $l_{max} = a_{max}x + b$, here b is the biggest intercept of a series of parallel lines. Pick any line of a set of non-vertical lines $l = a'x + b'$. There's always an intersection $(\frac{b'-b}{a_{max}-a'}, \frac{a_{max}b'-a'b}{a_{max}-a'})$. Let $x' = max\{\frac{b'-b}{a_{max}-a'}\}$, then for any $x>x'$, l_{max} always dominates. Similarly, there's always a x'' on which l_{min} always dominates when $x<x''$.

Proof of 5.3.2: Assume there's a "visible" line l that doesn't dominate at the intersection of l_{max} and l_{min}, say(x', y'). Therefore, $l(x')<y'$. Since it's "visible", we know there's some $x = t$ on which l dominates. Then we can infer $a_l = \frac{l(t)-l(x')}{t-x'}$. However, when $t>x'$, $a_l>a_{max}$; Similarly, when $t<x'$, $a_l<a_{min}$. That's contradictory to our assumption, which means l must dominate at $x = x'$.

Proof of 5.3.3: It's obvious that every single line should be "visible" in the "visible area" otherwise it'll be dominated by l_{max} or l_{min} forever. Now assume the orderings of the two x coordinate lists are not as suggested, which means there's some $i>j$, $P_{ri}>P_{rj}$ and $P_{li}>P_{lj}$. That way, l_i will dominate l_j in the "visible area" and thus makes it "invisible"

Proof of correctness to our algorithm: According to *Lemma 5.3.1* we know it's always safe to keep the two special lines(overall l_{min}, l_{max}). According to *Lemma 5.3.2*, every time we throw a line, that line is forever dominated by the current l_{max} and l_{min}. By doing the throwing step we're getting as many "invisible" lines as possible. Finally, our algorithm will generate a set of lines conforming *Lemma 5.3.3*. So we know the array we get has all the "visible" lines.