PSO 2 — Week 7: Solutions

CS381, Fall 2026

Problem 1

Throughout, for two lines s and t with $a_s < a_t$, write $x(s,t)$ for the x-coordinate where they cross. Since t is the steeper of the two, it is above s exactly to the right of $x(s,t)$.

(a) Stack scan

```
sort the lines so that a_1 < a_2 < ... < a_n
S := empty stack
for l = 1 to n:
    while S has at least two lines:
        t := top(S);  s := the line just below t
        if x(t, l) <= x(s, t): pop(S)       // t never reaches the top again
        else: break
    push l onto S
return S, read from bottom to top
```

Invariant. After the first k lines are processed, the stack holds the lines of the envelope of those k lines, in left-to-right order from bottom to top. A line t with s just below it has its piece beginning at $x(s,t)$, where the envelope switches from s to t; the bottom line's piece begins at $-\infty$.

Correctness. The picture is this: the new line is the steepest so far, so from some point on it is on top, and it buries every line whose piece lies entirely beyond that point. Those lines are the ones at the top of the stack.

In detail, let E be the envelope of the lines processed so far, and ℓ the new line. Then ℓ is above an earlier line j exactly to the right of $x(j,\ell)$, so it is above all of them, and hence above E, exactly to the right of the largest of those crossing points; call it x^*. So the new envelope is E to the left of x^* and ℓ to the right. A line of E disappears exactly when its whole piece lies to the right of x^*, that is, when its piece begins at or after x^*. Since the stack lists the pieces from left to right, the lines that disappear form a block at the top.

It remains to check that the test pops a line exactly when it disappears. Take the top line t and the line s below it, so that t's piece begins at $x(s,t)$. Then t disappears exactly when $x^* \le x(s,t)$, that is, exactly when ℓ is at least as high as E at $x(s,t)$. At that point E is the line t, so this happens exactly when ℓ is at least as high as t at $x(s,t)$, which is exactly when $x(t,\ell) \le x(s,t)$ — the test. The bottom line never disappears, since its piece begins at $-\infty$ and ℓ is below it far enough to the left. Pushing ℓ then restores the invariant. □

Since the stack holds the lines of the envelope and each line is pushed only once, each line appears on the envelope at most once. So the envelope of m lines has at most m pieces, even though the lines can cross in $m(m-1)/2$ points. Part (b) relies on this.

Time. Each line is pushed once, so it is popped at most once, and the loop does $O(n)$ work in total after the $O(n \log n)$ sort; this is the same counting argument as for Graham's scan. The test needs no division: the crossing points are $x(t,\ell) = (b_t - b_\ell)/(a_\ell - a_t)$ and $x(s,t) = (b_s - b_t)/(a_t - a_s)$, and both denominators are positive, so we can cross-multiply.

(b) Divide and conquer

```
Envelope(lines):
    if there is only one line: return it (it is on top everywhere)
    split the lines into two halves of nearly equal size, in any way
    E1 := Envelope(first half);  E2 := Envelope(second half)
    return Merge(E1, E2)
```

Merging. The envelope of all the lines is the larger of $E_1(x)$ and $E_2(x)$ at every x. Merge the breakpoints of E_1 and E_2 into one sorted list, as in merge sort. Between two consecutive points of that list, E_1 is a single line f and E_2 a single line g. Two lines cross only once, so on that interval the higher of the two is f on one side of their crossing and g on the other. Check in $O(1)$ time whether the crossing falls inside the interval, and output the winning line on each part. Finally, join neighbouring pieces that have the same line.

Correctness. By induction on the number of lines: the recursive calls return the envelopes of the two halves, and the merge takes the larger of the two at every x, which is the envelope of all the lines. □

Time. By (a), the envelope of m lines has at most m pieces, so merging the envelopes of the two halves of n lines handles $O(n)$ pieces and takes $O(n)$ time. The recurrence is $T(n) = 2T(n/2)+O(n)$, as for merge sort, so $T(n) = O(n \log n)$. That is the same bound as (a), but reached without sorting anything: the split was arbitrary.