Homework 1

Elliott Pryor

24 Jan 2021

Problem 1

Some sort of binary search.

Jointly search P and P' Pick $a \in P, \;\; b \in P'$ Want line with highest z intercept

Idea: O(n) run Grahm's Scan on it. Already sorted so takes O(n) time.

Idea: Start at a compute point $b \in P'$ tangent to a (O (log n)). Then reverse, find a tangent to b. Then forward, find b tangent to a. Then done????

Algorithm 1 Tangent Function

function $\textsc{Tangent}(a, P)$
    Binary search to find point of tangency
    $low \leftarrow 0$
    $high \leftarrow |P|$
    while !found **do**
        $m \leftarrow \lceil (low + high)/2 \rceil$
        $line \leftarrow \overrightarrow{a, m}$
        if $line([m+1]_x) > [m+1]_y$ and $line([m-1]_x) > [m-1]_y$ **then**
            found it (is supporting line)
            return m
        else if $line([m+1]_x) > [m+1]_y$ **then**
            line intersects some point before m (tangent point to left)
            $high \leftarrow m - 1$
        else if $line([m-1]_x) > [m-1]_y$ **then**
            line intersects some point after m (tangent point to right)
            $low \leftarrow m + 1$
        end if
    end while
end function

function $\textsc{UpperTangent}(P, P')$
    Run Tangent 3 times to find upper tangent.
    $a \leftarrow |P|/2$ // Random point in P
    $b \leftarrow Tangent(a, P')$ // So we can 'see' p_i from b
    $a \leftarrow Tangent(b, P)$ // Finds p_i
    $b \leftarrow Tangent(a, P')$ // Finds p_j
    return $\overrightarrow{a, b}$
end function

Problem 2

1. The points on the $Pareto(P)$ fall on a series of horizontal and vertical line segments. It follows a line in 'Manhattan' distance. In order to make the analogous assertion, we define corner $C(p)$ $p \in P$ as the region $(x, y) \in \mathbb{R}^2$ $x \leq p_x, \ \ y \leq p_y$.

   Then a point p is on the Pareto of a set P if and only if there is no other corner $C(p'), \ \ p \neq p'$, containing p

2. This is almost identical to Grahm's Scan. We replace the Orient() function with a different comparison. We simply compare the y values of adjacent points. Since the points are in sorted, x, order if a a point p_i has a larger y-coordinate than p_{i-1} the p_{i-1} would be in $C(p_i)$ so is not on the Pareto. Since this comparison only needs the first point in the stack, we also adjust initialization of S to only push p_1 onto the stack, and to not terminate the while loop unless there are 0 points in the stack.

   This has the same running time as Grahm's Scam. Since after sorting, the algorithm takes a linear pass through all of the points. It also only pops a point at most once from the stack. The only difference with this algorithm from Grahm's Scan is the Orient. Since comparing y-coordinates is also a constant time lookup, our running time is $O(nlog(n))$ due to the sorting in line 2.

Algorithm 2 Pareto Scan

1: **function** ParetoScan(P)
2:     sort P by increasing x value
3:     push p_1 onto stack S
4:     **for** $i \leftarrow 2, ..., n$ **do**
5:         **while** $|S| \geq 1$ and $p_{i,y} \geq S[top]_y$ **do**
6:             pop S
7:         **end while**
8:         push p_i onto S
9:     **end for**
10: **end function**

3. Our modified Jarvis march algorithm would search for the point with the greatest y value. Then it would loop and search for the point with the next greatest y-coordinate to the right of the previous point ($p_{i,x} > p_{i-1,x}$). This loop is repeated $h - 1$ times (the first point was found in initialization step of finding point with largest y-coordinate).

4. We start by dividing P into n/h sets of size h. We run Pareto Scan (Algorithm 2) on each of the n/h sets. The runtime of this step is $O(h \log(h))$ repeated n/h times: $O(n \log(h))$.

   We can then merge these in $O(n)$ time. We know that each pareto front found is at most h long. We merge these fronts in sorted order by x. This takes $O(n)$ this is the same as merge operation in MergeSort. We then iterate through this list and build a pareto from this. We know this takes $O(n)$ since they are sorted, so it is the same operation as in Algorithm 2

Algorithm 3 Jarvis Stairs

```
function JarvisStairs(P)
    find point p_1 with largest y-coordinate
    S ← p_1
    for i ← 2, ..., h do
        find point p_i with largest y-coordinate such that p_{i,x} > p_{i-1,x}
    end for
    return S
end function
```

Algorithm 4 Chan Pareto

```
function ChanPareto(P)
    divide P into P_1, P_2, ...P_{n/h} where |P_i| = h
    Solve each P_i using Algorithm 2 and store paretos in S_i
    merge paretos S_i in sorted x order into P'
    push p'_1 onto stack S
    for i ← 2, ..., n do
        while |S| ≥ 1 and p'_{i,y} ≥ S[top]_y do
            pop S
        end while
        push p'_i onto S
    end for
    return S
end function
```

Problem 3

We say $Orient(a, b, c) > 0$ if it is oriented counter clockwise (makes left turn), and $Orient(a, b, c) < 0$ if it is oriented clockwise (right turn)

Algorithm 5 Tangent Function

function TANGENT(a, P)
    Binary search to find point of tangency
    $low \leftarrow 0$
    $high \leftarrow |P|$
    p_1, p_2
    while !found **do**
        $c \leftarrow \lceil (low + high)/2 \rceil$
        $b \leftarrow c - 1, \;\; d \leftarrow c + 1$ b is point counter-clockwise from a, d is point clockwise from a
        if $Orient(a, c, b) = Orient(a, c, d)$ **then**
            found tangent
            $p_1 \leftarrow m$
            break while
        else if $Orient(a, c, b) > 0$ and $Orient(a, c, d) < 0$ **then**
            Need to move search point left (counter-clockwise)
            $high \leftarrow b$
        else if $Orient(a, c, b) < 0$ and $Orient(a, c, d > 0)$ **then**
            need to move search point right (clockwise)
            $low \leftarrow d$
        end if
    end while
    Repeat same while loop but flipping inequality signs to find other tangency point.
    return p_1, p_2
end function