CS 4820, Spring 2019 **Homework 5, Problem 2**
Name: Kevin Klaben NetID: kek228 Collaborators: Quinn Lui

(2) *(15 points)* Given as input a list of n points $L = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ on the real plane, your task is to compute the largest rectangle (in terms of area) that can be formed by selecting two points from L, one representing the bottom-left vertex of the rectangle and the other representing the top-right vertex. For simplicity, assume that all the a_i's and b_i's are distinct real numbers.

(2a) *(3 points)* Define two lists of points BL and TR in the following way:

$$BL = \{(a_i, b_i) \in L : \text{for any } j \neq i, \text{ either } a_i < a_j \text{ or } b_i < b_j\}$$

and

$$TR = \{(a_i, b_i) \in L : \text{for any } j \neq i, \text{ either } a_i > a_j \text{ or } b_i > b_j\}.$$

Prove that there exists a rectangle with largest area (using points from L) that has its bottom-left vertex in BL and top-right vertex in TR. Provide an $O(n \log n)$ time algorithm to compute BL and TR. You must output each of the two lists BL and TR by sorting them according to the x-coordinates of the points (in increasing order). **You don't have to provide proof of correctness of your algorithms. You do have to analyze run-time of the algorithms you provide.**

(2b) *(2 points)* Let (a_i, b_i) and (a_j, b_j) be points in BL such that $a_i < a_j$. Further, let (a_k, b_k) and (a_ℓ, b_ℓ) be points in TR such that $a_k < a_\ell$. Define $\Delta_{e,f}$ to be the area of the rectangle using (a_e, b_e) as the bottom-left vertex and (a_f, b_f) as the top-right vertex, where $e \in \{i, j\}$ and $f \in \{k, l\}$. Prove that $\Delta_{i,k} + \Delta_{j,\ell} > \Delta_{i,\ell} + \Delta_{j,k}$.

(2c) *(10 points)* Design an algorithm that runs in time $O(n \log n)$ to compute the largest rectangle (in terms of area) that can be formed by selecting the bottom-left vertex from BL and the top-right vertex from TR. The output of the algorithm should be the area of the largest rectangle.

(a)

```
BL(L)
sort[]=mergesort(L by x-value lowest to highest)
output[]=[]
miny=the y value of sort[0]
for each (x,y) in sort do
  if y<=miny then
    add (x,y) to the end of output
    miny=y
  end if
end for
return output
```

Runtime Analysis BL: Initially the mergesort operation will take O(nlogn) time. Next the foreach loop will take an addition O(n) time as it iterates over each element doing O(1) operations each time. Thus the overall runtime is O(nlogn)+O(n)=O(nlogn).

The algorithm for TR

```
TR(L)
sort[]=mergesort(L by x-value highest to lowest)
output[]=[]
maxy=the y value of sort[0]
for each (x,y) in sort do
  if y>=maxy then
    add (x,y) to the end of output
    maxy=y
  end if
end forreturn output after having its order reversed
```

Runtime Analysis RT: Initially the mergesort operation will take O(nlogn) time. Next the foreach loop will take an addition O(n) time as it iterates over each element doing O(1) operations each time. Finally, the output is reversed whcih could take a maximum of O(n) time. Thus the overall runtime is O(nlogn)+O(n)+O(n)=O(nlogn).

(b) Claim: $\Delta_{i,k} + \Delta_{j,\ell} > \Delta_{i,\ell} + \Delta_{j,k}$.
Proof: (a_i, b_i) and (a_j, b_j) (a_k, b_k) and (a_l, b_l) We know $a_i < a_j < a_k < a_l$ and in turn $b_j < b_i < b_l < b_k$ as if $a_k or a_l were < a_i or a_j$ then the definition of Delta would fail, similarly the same holds for the y-values. We arrive at the conclusions for the y-values that $b_j < b_i and b_l < b_k$ as if this were not true then both points (a_i, b_i) and (a_j, b_j) could not be a part of BL and then (a_k, b_k) and (a_l, b_l) could not both be a part of TR.
$\Delta_{i,k}$= $(a_k - a_j)$*$(b_l - b_i)$+ $(a_j - a_i)$*$(b_l - b_i)$+$(a_j - a_i)$*$(b_k - b_l)$+$(a_k - a_j)$*$(b_k - b_l)$
$\Delta_{j,l}$= $(a_k - a_j)$*$(b_l - b_i)$+ $(a_l - a_k)$*$(b_l - b_i)$+$(a_l - a_k)$*$(b_i - b_j)$+$(a_k - a_j)$*$(b_i - b_j)$
$\Delta_{i,l}$= $(a_k - a_j)$*$(b_l - b_i)$+ $(a_j - a_i)$*$(b_l - b_i)$+ $(a_l - a_k)$*$(b_l - b_i)$
$\Delta_{j,k}$= $(a_k - a_j)$*$(b_l - b_i)$+ $(a_k - a_j)$*$(b_k - b_l)$+ $(a_k - a_j)$*$(b_i - b_j)$
The claim can be manipulated to say $0 < \Delta_{i,k} + \Delta_{j,\ell} - \Delta_{i,\ell} + \Delta_{j,k}$.
Substituting the calculations above we get 0 < ((a_k-a_j)*(b_l-b_i)+ (a_j-a_i)*(b_l-b_i)+(a_j-a_i)*(b_k-b_l)+(a_k-a_j)*(b_k-b_l)+(a_k-a_j)*(b_l-b_i)+ (a_l-a_k)*(b_l-b_i)+(a_l-a_k)*(b_i-b_j)+(a_k-a_j)*(b_i-b_j))-((a_k-a_j)*(b_l-b_i)+ (a_j-a_i)*(b_l-b_i)+ (a_l-a_k)*(b_l-b_i)+(a_k-a_j)*(b_l-b_i)+ (a_k-a_j)*(b_k-b_l)+ (a_k-a_j)*(b_i-b_j))
which simplifies to:
0 < $(a_j - a_i)$*$(b_k - b_l)$+ $(a_k - a_j)$*$(b_i - b_j)$.
We know this will always hold as each value being multiplied will always be positive by our claim made earlier that $a_i < a_j < a_k < a_l$ and $b_j < b_i < b_l < b_k$. Thus as the right hand side is always positive and thus greater than zero, the claim holds.
(c)

```
RECRECT(BL,TR)
if ($|BL| == |TR| == 1$) then
  Calculate the size of rectangle from the only bottom left point and the only topright point
  return (BL pt), (TR pt), Area
```

```
else
    Split BL into two sets maintaining order BL1 and BL2, if BL only has one element simply
    make BL1=BL2=BL. Do the same from TR -¿ TR1 and TR2.
    (($blx_1$,$bly_1$),($trx_1$,$try_1$), $a_1$))=RECRECT(BL1,TR1)
    (($blx_2$,$bly_2$),($trx_2$,$try_2$), $a_2$))=RECRECT(BL2,TR2)
    (a,b,area3)=RECRECT(($blx_1$,$bly_1$), ($trx_2$,$try_2$))
    (c,d,area4)=RECRECT(($blx_2$,$bly_2$), ($trx_1$,$try_1$))
    if $a_1 > a_2$ then  return (($blx_1$,$bly_1$),($trx_1$,$try_1$), max($a_1, a_2, area3, area4$))
  else
    return (($blx_2$,$bly_2$),($trx_2$,$try_2$), max($a_1, a_2, area3, area4$))
  end if
end if
```

Calling RECRECT(BL,TR) will result in (blpt,trpt,area) where area is the max area.

Runtime Analysis:
The depth of recursion will be O(logn) and each operation will take O(n) time. Together, the overall runtime will be O(nlogn).
Proof of Correctness:
Claim: the algorithm outputs the maximum area rectangle
Proof:
by induction on the number of points in BL and TR.
Case: —BL—=1 and —TR—=1
Proof: In this case there is only one point in BL and one point in TR and thus, as BL and TR are the absolute top right and bottom left. Thus, the output rectangle is the largest among possible rectangle among these points.
Case: given we have the largest rectangle among half the points in BL with half the points in TR, we will output the largest area among all of BL and all of TR.
Proof: We have the largest rectangle in BL1 and TR1 and the largest rectangle in BL2 and TR2. As seen in part 2 these areas combined will be the larger than the areas of these points swapped. Thus, the only possible area that could be larger would be if one of these swapped was larger than both individual areas. Thus this comparison is done. The max of these areas is returned and used as the future area. This will be the max area and thus the claim holds.