CS 120: Intro to Algorithms and their Limitations

Problem Set 2

Due: September 27, 2023 11:59pm **Denny Cao**

Collaborators:
No. of late days used on previous psets: 0
No. of late days used after including this pset: 1

§1 Reductions

(a) *Proof.*

We first describe the reduction algorithm.

- Preprocess an array by converting the Cartesian coordinates to polar coordinates, and store θ as the key and $(r, (x, y))$ as the value. Our array will contain $(\texttt{key}, \texttt{value})$ pairs in the form $(\theta_i, (r_i, (x_i, y_i)))$, for all $i \in \{0, 1, \dots, n-1\}$, where (x_i, y_i) represents the ith coordinate point in the input and (θ_i, r_i) represents the ith coordinate point in the input in polar coordinates.
- Sort this array by K_i, the angle of the coordinate points in polar form to create a sorted array S by an oracle call to `Sorting`.
- Given S, for each pair of adjacent elements $(K'_i, V'_i) = (\theta'_i, (r'_i, (x'_i, y'_i)))$ and $(K'_{i+1}, V'_{i+1}) = (\theta'_{i+1}, (r'_{i+1}, (x'_{i+1}, y'_{i+1})))$, determine the area of the triangle formed by the two points and the origin at $(0, 0)$ by using Heron's Formula and then add it to a variable `area`:
  - Let $a = r'_i, b = r'_{i+1}, c = \sqrt{(x'_{i+1} - x'_i)^2 + (y'_{i+1} - y'_i)^2}$.
  - Let $s = \dfrac{a+b+c}{2}$.
  - The area of the triangle is then $\sqrt{s(s-a)(s-b)(s-c)}$.
  - For (K'_{n-1}, V'_{n-1}), the adjacent element will be the first element, (K'_0, V'_0).
- Return `area`.

We now want to prove our reduction algorithm:

1. has the desired runtime and call size to the `Sorting` oracle, and
2. is correct.

- Forming an array in the first step takes time $O(n)$, as by our assumption, converting into polar coordinates is constant time, and thus there are $O(1)$ operations for each point in the input of size n. Computing basic operations on adjacent elements in the array will take $O(n)$, as there are $O(1)$ operations performed for each of the n elements in the array. Adding the area of the triangle to `area` will also be a basic operation, taking time $O(1)$, and thus with n areas, will take $O(n)$. Thus, the total time taken by the reduction is $O(n)$. Also, the array passed to `Sorting` has size n, as claimed and the oracle is only called once.

- All convex polygons with n sides can be constructed by n triangles using adjacent pairs of vertices and a point in the interior of the polygon as vertices of each triangle. The sum of the area of the triangles will thus be the area of the convex polygon. Our reduction algorithm sorts by θ in order to obtain adjacent vertices and then obtains the area of the triangles using the origin, $(0,0)$ as the third point which the computational problem states is in the interior of the convex polygon. It then sums all triangle areas together and returns the sum, which is the area of the convex polygon. Thus, our algorithm is correct.

We have shown that `AreaOfConvexPolygon` $\leq_{O(n),n}$ `Sorting` and that the reduction is correct. □

(b) *Proof.* We have shown an $O(n)$-time algorithm for `AreaOfConvexPolygon` that makes one call to the `Sorting` oracle on an array of size n. Thus by Lemma 3 (Lecture 3: Reductions),

$$\text{Time}_{\texttt{AreaOfConvexPolygon}}(n) \leq O(n) + \text{Time}_{\texttt{Sorting}}(n)$$

Let the `Sorting` oracle be a sorting function that runs in $O(n \log n)$ time. Thus:

$$\text{Time}_{\texttt{AreaOfConvexPolygon}}(n) \leq O(n) + O(n \log n) = O(n \log n)$$

Thus, `AreaOfConvexPolygon` can be solved in time $O(n \log n)$. □

(c) *Proof.* Let γ be an algorithm that solves Γ and let π be an algorithm that solves Π. As Π can be reduced to Γ, we can view γ as the oracle that solves Γ. Let π call γ.

Then, the runtime of π can be bounded by the sum of the maximum time for preprocessing and postprocessing for γ and the maximum time it takes for all calls of γ. As each call of γ takes time $T(n)$ for an input size n and the greatest input size possible is $h(n)$, then each call of γ takes time at most $O(T(h(n)))$.

As γ is called at most $k(n)$ times, the maximum time for all calls of γ is $O(k(n)T(h(n)))$. As the reduction runs in time at most $g(n)$, then the preprocessing and postprocessing for γ takes at most $O(g(n))$ time. Thus, the maximum time it takes for an algorithm π to solve Π is $O(g(n) + k(n) \cdot T(h(n)))$. It follows that the maximum time it takes for Π to be solved if Γ can be solved in at most $T(n)$ is $O(g(n) + k(n) \cdot T(h(n)))$. □

(d)

Proof. We first describe the reduction algorithm.

- Preprocess 4 arrays $Q_j, j \in \{1,2,3,4\}$, where Q_j denotes the array of coordinates in the jth quadrant. For each Cartesian coordinate pair, (x_i, y_i), $i \in \{0,1,\ldots,n-1\}$, compute the slope m_i between the point and the origin $(0,0)$ by setting $m_i = \dfrac{y_i}{x_i}$ (if $x = 0$, then the slope will be `MAX` if $y > 0$ and `MIN` if $y < 0$). We then insert each coordinate pair into its corresponding array in $(\texttt{key}, \texttt{value})$ pairs in the form $(m_i, (x_i, y_i))$, for all $i \in \{0,1,\ldots,n-1\}$. If $x_i = 0 \wedge y > 0$, then insert into Q_1. If $x_i < 0 \wedge y = 0$, then insert into Q_2. If $x_i = 0 \wedge y < 0$, then insert Q_3, $x > 0 \wedge y = 0$, then insert Q_4.

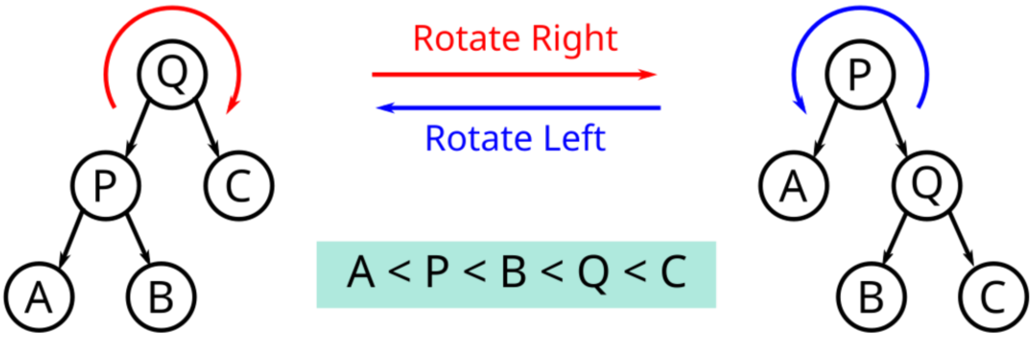
- For each array, sort the array by increasing m_i, the slope of the coordinate points by an oracle call to `Sorting`.
- Join the arrays together into an array S in order, with Q_1 preceding Q_2 which precedes Q_3 which precedes Q_4.
- Given S, for each pair of adjacent elements $(K'_i, V'_i) = (m'_i, (x'_i, y'_i))$ and $(K'_{i+1}, V'_{i+1}) = (m'_{i+1}, (x'_{i+1}, y'_{i+1}))$, determine the area of the triangle formed by the two points and the origin at $(0, 0)$ by using Heron's Formula and then add it to a variable `area`:
  - Let $a = \sqrt{x_i'^2 + y_i'^2}, b = \{x_{i+1}'^2 + y_{i+1}'^2\}, c = \sqrt{(x'_{i+1} - x'_i)^2 + (y'_{i+1} - y_i)^2}$.
  - Let $s = \dfrac{a+b+c}{2}$.
  - The area of the triangle is then $\sqrt{s(s-a)(s-b)(s-c)}$.
  - For (K'_{n-1}, V'_{n-1}), the adjacent element will be the first element, (K'_0, V'_0).
- Return `area`.

Creating the 4 quadrant arrays takes $O(1)$ time. Iterating through the inputs and getting slope and adding to the respective quadrant array for each input takes $O(1)$ time, resulting in $O(n)$ time. We call the oracle $O(1)$ calls, as we always call it for 4 quadrant arrays. Thus, it runs in $O(n)$. If we take sorting into account, $O(n) + O(n \log n) = O(n \log n)$. The correctness proof is the same as (c). $\square$

§2 Augmented Binary Search Trees

(a) The `select` function has a correctness error. When `ind` is greater than `left_size`, the algorithm checks the right subtree. However, it attempts to find the `ind` least key on the right subtree, which would imply that the selected key would be the `ind + left_size` least key, since all elements on the left subtree are less than the right. The fix is to instead find the `ind - left_size` least key on the right, as that would mean the number we selected would be the `ind` least key.

The `insert` function is too slow. The run time is $O(hn)$, as it traverses to only 1 node in every level, but in each level, it calls `calculate_sizes` which has runtime $O(n)$, as it traverses every node in the tree. The fix is to instead of calling `calculate_sizes`, setting the current node k's size to `k.size = 1 + k.left.size + k.right.size`, and removing `k.left.size` if `k.left is None` and removing `k.right.size` if `k.right is None`, allowing the calculation of sizes to be done only on the nodes that are affected by the insertion, and in $O(1)$ time for each node.



(b) We use the rotation algorithm for BSTs depicted above and will extend it to size-augmented BSTs.

We consider the case of a right rotation. In this case, the rotation can remain the same, but we must change the size attributes of the nodes. Let `x.size` denote the size of a node x. Then, after a right rotation, `P.size = A.size + B.size + C.size + 2` and `Q.size = B.size + C.size + 1`. The same argument can be made for the left rotation by interchanging P and Q.

Claim — The extension of `rotation` maintains the runtime $O(1)$.

Proof. The extension maintains the runtime $O(1)$, as the rotation itself takes $O(1)$ time, accessing the size attributes of 3 subtrees A, B, C is a basic operation and takes $O(1)$ time, and setting the size attribute for P and Q is a basic operation and takes $O(1)$. Thus, the runtime remains $O(1)$. □

Claim — The new rotation operation preserves the invariant of correct size-augmentations.

Proof. The new rotation operation preserves the invariant of correct size-augmentations, as the subtrees A, B, C are not affected during the rotation, and the size attributes for A, B, C were correct before the rotation, they remain correct after the rotation. If there exists a parent for the node being rotated (P when rotating left, Q when rotating right), no descendants are removed, and thus the size of the parent will be the same and remain correct. The sizes of P and Q are defined as the sum of the sizes of the children and the node itself, and thus the size of P and Q are also correct. □