

## Problem Set 2

Harvard SEAS - Fall 2023

Due: Tue Sep. 27, 2023 (11:59pm)

**Your name:****Collaborators:****No. of late days used on previous psets:****No. of late days used after including this pset:**

Please review the Syllabus for information on the collaboration policy, grading scale, revisions, and late days.

1. (reductions) The purpose of this exercise is to give you practice formulating reductions and proving their correctness and runtime. Consider the following computational problem:

<b>Input</b>	: Points $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$ in the $\mathbb{R}^2$ plane that are the vertices of a convex polygon (in an arbitrary order) whose interior contains the origin
<b>Output</b>	: The area of the polygon formed by the points

**Computational Problem** AreaOfConvexPolygon

- (a) Show that  $\text{AreaOfConvexPolygon} \leq_{O(n), n} \text{Sorting}$ . Be sure to analyze both the correctness and runtime of your reduction. In this part and the next one, you may assume that a point  $(x, y) \in \mathbb{R}^2$  can be converted into polar coordinates  $(r, \theta)$  in constant time.

You may find the following useful:

- The polar coordinates  $(r, \theta)$  of a point  $(x, y)$  are the unique real numbers  $r \geq 0$  and  $\theta \in [0, 2\pi)$  such that  $x = r \cos \theta$  and  $y = r \sin \theta$ . Or, more geometrically,  $r = \sqrt{x^2 + y^2}$  is the distance of the point from the origin, and  $\theta$  is the angle between the positive  $x$ -axis and the ray from the origin to the point.
- The area of a triangle is  $A = \frac{\sqrt{s(s-a)(s-b)(s-c)}}{2}$  where  $a, b, c$  are the side lengths of the triangle and  $s = \frac{a+b+c}{2}$  ([Heron's Formula](#)).

- (b) Deduce that  $\text{AreaOfConvexPolygon}$  can be solved in time  $O(n \log n)$ .
- (c) Let  $\Pi$  and  $\Gamma$  be arbitrary computational problems, and suppose that there is a reduction from  $\Pi$  to  $\Gamma$  that runs in time at most  $g(n)$  and makes at most  $k(n)$  oracle calls, all on instances of size at most  $h(n)$ . Show that if  $\Gamma$  can be solved in time at most  $T(n)$ , then  $\Pi$  can be solved in time at most  $O(g(n) + k(n) \cdot T(h(n)))$ . Note that the case  $k(n) = 1$  was stated in class; the case  $k(n) > 1$  is useful as well, such as in Part ?? below.
- (d) (\*challenge; extra credit; optional<sup>1</sup>) Come up with a way to avoid conversion to polar coordinates and any other trigonometric functions in solving  $\text{AreaOfConvexPolygon}$  in time  $O(n \log n)$ . Specifically, design an  $O(n)$ -time reduction that makes  $O(1)$  calls to a

<sup>1</sup>This problem is meant to be done based on your enjoyment/interest and only if you have time. It won't make a difference between N, L, R-, and R grades (meaning it will only impact whether an R gets increased to an R+), and course staff will deprioritize questions about this problem at office hours and on Ed.

Sorting oracle on arrays of length at most  $n$ , using only arithmetic operations  $+$ ,  $-$ ,  $\times$ ,  $\div$ , and  $\sqrt{\phantom{x}}$ , along with comparators like  $<$  and  $==$ . (Hint: first partition the input points according to which quadrant they belong in, and consider the slope of the line from a vertex  $(x,y)$  to the origin.)

Similar techniques to what you are using in this problem are used in algorithms for other important geometric problems, like finding the Convex Hull of a set of points, which has applications in graphics and machine learning.

2. (augmented binary search trees) The purpose of this problem is to give you experience reasoning about correctness and efficiency of dynamic data-structure operations, on variants of binary-search trees.

Specifically, we will work with *selection data structures*. We have seen how binary search trees can support min queries in time  $O(h)$ , where  $h$  is the height of the tree. A generalization is *selection* queries, where given a natural number  $q$ , we want to return the  $q$ 'th smallest element of the set. So `DS.select(0)` should return the key-value pair with the minimum key among those stored by the data structure `DS`, `DS.select(1)` should return the one with the second-smallest key, `DS.select(n-1)` should return the one with the maximum key if the set is of size  $n$ , and `DS.select((n-1)/2)` should return the median element if  $n$  is odd.

In the Roughgarden text (§11.3.9), it is shown that if we *augment* binary search trees by adding to each node  $v$  the size of the subtree rooted at  $v$ , then Selection queries can be answered in time  $O(h)$ .<sup>2</sup>

- (a) In the Github repository, we have given you a Python implementation of size-augmented BSTs supporting search, insertion, and selection, and with a stub for `rotate`. One of the implemented functions (`search`, `insert`, or `select`) has a correctness error, another one is too slow (running in time linear in the number of nodes of the tree rather than in the height of tree), and the third is correct. Identify and correct these errors. You should provide a text explanation of the errors and your corrections, as well as implement the corrections in Python.
- (b) Describe (in pseudocode or pictures) how to extend `rotate` to size-augmented BSTs, and argue that your extension maintains the runtime  $O(1)$ . Prove that your new rotation operation preserves the invariant of correct size-augmentations. (That is, if every node's size attribute had the correct subtree size before the operation, then the same is true after the operation.)
- (c) Implement `rotate` in size-augmented BSTs in Python in the stub we have given you.

*Food for thought (do read - it's an important take-away from this problem):* This problem concerns size-augmented binary search trees. In lecture, we discussed AVL trees, which are balanced binary search trees where every vertex contains an additional *height* attribute containing the length of the longest path from the vertex to a leaf (height-augmented). Additionally, every pair of siblings in the tree have heights differing by at most 1, so the tree is height-balanced. Note that if we augment a binary search tree both by size (as in the above problem) and by height (and use it to maintain the AVL property), then we create a dynamic data structure able to perform `search`, `insert`, and `select` all in time  $O(\log n)$ .

---

<sup>2</sup>Note that the Roughgarden text uses a different indexing than us for the inputs to `Select`. For Roughgarden, the minimum key is selected by `Select(1)`, whereas for us it is selected by `Select(0)`.