CS1200: Intro. to Algorithms and their Limitations Anshu & Vadhan

Problem Set 2

Harvard SEAS - Fall 2024 *Due: Wed 2024-09-25 (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. (designing reductions) The purpose of this exercise is to give you practice designing reductions and proving their correctness and runtime. Consider the following computational problem:

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

Computational Problem AreaOfConvexPolygon

   (a) Show that AreaOfConvexPolygon$\leq_{O(n),n}$ 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, θ) in constant time.

   You may find the following useful (and you may use them without proof):
   - The polar coordinates (r, θ) 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 θ is the angle between the positive x-axis and the ray from the origin to the point.
   - The area of a triangle is $A = \sqrt{s(s-a)(s-b)(s-c)}$ where a, b, c are the side lengths of the triangle and $s = \frac{a+b+c}{2}$ (Heron's Formula).

   (b) Deduce that AreaOfConvexPolygon can be solved in time $O(n \log n)$.

   (c) (*challenge; extra credit; optional[1]) Come up with a way to avoid conversion to polar coordinates and any other trigonometric functions in solving AreaOfConvexPolygon in time $O(n \log n)$. Specifically, design an $O(n)$-time reduction that makes $O(1)$ calls to a Sorting oracle on arrays of length at most n, using only arithmetic operations $+$, $-$, $\times$, $\div$, and $\sqrt{}$, 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.)

[1] 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.

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. (composition of reductions) The purpose of this exercise is to give you practice in working with the abstract definition of reductions.

   Let Π, Γ, Λ be computational problems, and suppose that $\Pi \leq_{T_1(n),q_1(n)\times h_1(n)} \Gamma$ and $\Gamma \leq_{T_2(n),q_2(n)\times h_2(n)} \Lambda$, for non-decreasing functions $T_1, T_2, q_1, q_2, h_1, h_2$. Determine functions T_3, q_3, h_3 in terms of $T_1, T_2, q_1, q_2, h_1, h_2$ such that

   $$\Pi \leq_{O(T_3(n)),q_3(n)\times h_3(n)} \Lambda,$$

   and justify your answers. (Hint: you can take $h_3(n) = h_2(h_1(n))$.)

3. (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 $O(h)$.[2]

   (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 that's (at least) linear in the number of nodes of the tree rather than linear 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

[2] 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).

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 $O(\log n)$.

4. (reflection) We aim for cs1200 to be a collaborative learning community. Describe two concrete ways in which you have supported, or will try to support, your classmates' learning in the course. Be specific, connecting your answer to the structure of cs1200.

   Note: As with the previous psets, you may include your answer in your PDF submission, but the answer should ultimately go into a separate Gradescope submission form.

5. Once you're done with this problem set, please fill out this survey so that we can gather students' thoughts on the problem set, and the class in general. It's not required, but we really appreciate all responses!