Problem Set 2

All parts are due Thursday, October 2 at 11:59PM. Please download the .zip archive for this problem set, and refer to the README.TXT file for instructions on preparing your solutions. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A

Problem 2-1. [15 points] **Building a Search Tree**

Recall that the running time of many operations supported by a binary search tree depends on the height of the tree, which is ideally logarithmic in the number of nodes in the tree. In this exercise, for a constant $c > 0$ we call a tree with n nodes c-*balanced* if its height is at most $c \log_2 n$.

(a) [5 points] Describe an algorithm (clearly in plain English or pseudocode) with asymptotically optimal running time that does the following: Given a sorted array $L[1], \ldots, L[n]$, the algorithm should generate a binary search tree containing the elements of L. Moreover, there must exist a constant c such that the resulting tree is always c-balanced.

(b) [5 points] Prove that your algorithm is correct and achieves the desired height. That is, prove that there is a constant c such that your algorithm always outputs a valid binary search tree out of the given array and that the tree is c-balanced.

(c) [5 points] Argue that your algorithm achieves the best possible asymptotic running time.

Problem 2-2. [25 points] **Search trees**

Consider an abstract data type supporting the following operations:

- INSERT(x) takes an integer x and adds x to the data structure (you may assume that the same x is never inserted twice).
- COUNTINRANGE(a, b) takes integers a and b, where $a \leq b$, and returns the number of items in the data structure that lie in the range $[a, b]$.

As an example, such a data structure can be used to store different arrival times of flights in an airport, so that it is possible to report the number of flights arriving at a given time period.

In all parts of this problem, T is a binary search tree that stores n distinct integers.

(a) [5 points] Suppose we use the binary search tree data structure with no modification to support range queries. Describe an implementation of COUNTINRANGE(a, b). For simplicity, you may assume that a and b are integer values that are guaranteed to be already stored in T. The algorithm should run in $O(k+h)$ time where k is the number of nodes with values in the range $[a, b]$ and h is the height of T.

(b) [15 points] The running time $O(k+h)$ from the previous part can be improved to $O(h)$ by appropriately augmenting the search tree (that is, adding auxiliary information in the nodes that help speed up the queries). Describe an augmentation strategy that enables both INSERT and COUNTINRANGE operations to be implemented in $O(h)$ time. Prove that your strategy works by implementing INSERT and COUNTINRANGE in $O(h)$ time, and proving their correctness. Make sure that the auxiliary information is appropriately updated whenever necessary.

(c) [5 points] Given an unsorted array of n numbers (you may assume that there are no duplicates), and "black box" access to a range search data structure, describe a sorting algorithm that makes use of the INSERT and COUNTINRANGE operations to return a corresponding sorted array of n numbers. However, your algorithm is limited to only use the following resources:

- $O(n)$ calls to the INSERT and COUNTINRANGE operations, and
- $O(n)$ additional time for any arbitrary computation.

Show that your algorithm is correct.

Problem 2-3. [20 points] **Dynamic Medians**

Recall that a median of a set of n distinct elements is an element of the set that is larger than or equal to exactly $\lfloor (n+1)/2 \rfloor$ or $\lceil (n+1)/2 \rceil$ elements (note that if S has even size, there are two medians).

Using only heaps, design a new data structure for maintaining a set S of integers, supporting the following operations with the given running times (n is the number of items currently present in the set):

- CREATE( ): Create an empty set S in constant time.
- INSERT(x): Add a new given number x to S in time $O(\log n)$. (assume no duplicates are added to S).
- MEDIAN( ): Return a median of S in constant time (if there are two, either may be returned).
- EXTRACTMEDIAN( ): Extract and return a median of S in $O(\log n)$ time (if there are two, either may be extracted).

Describe your data structure, implement the above three operations and prove that they are correct and attain the desired running time.

Part B

Problem 2-4. [40 points] **Bridge Traversal**

James Cameron is designing a new world for his next movie, *Avatar 2*. His new world, Hesiod, is close in proximity to Pandora but lacks the lush, tropical rainforests and instead has many more oceans and volcanoes. Similar to Pandora, however, Hesiod's core creates an incredibly strong magnetic field, which causes everything made from superconductive Unobtainium to float. As a first step in creating the new world, Cameron decides to place floating bridges throughout. The idea is that these bridges will allow the indigenous species from Pandora, the Na'vi, to get around, bypassing the oceans and volcanoes. Cameron has a layout of bridges in mind but he is having trouble figuring out exactly how many bridges the Na'vi will be able to traverse and how many will be used for show. He comes to MIT for help with this mind boggling problem.

Using your new knowledge of data structures and algorithms you decide that you can take on the job with relative ease. You tell Cameron that you have found a way to determine the number of bridges (for each layout) that can be used if the Na'vi start traversing bridges from point $(0, 0)$ and go forward until they fall into the watery abyss as shown in **Figure 1**.

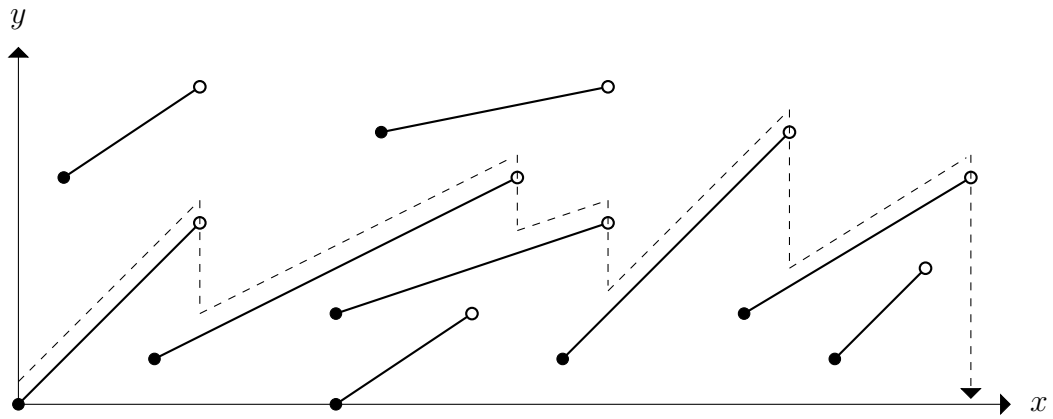

Figure 1: Example Na'vi bridge traversal starting at $(0, 0)$ and crossing 5 bridges.

Implement the `traverse` function that takes in a list of bridges and outputs the total number of bridges that a Na'vi native would be able to traverse before falling into the hot, watery abyss. Traversal will always start at $(0, 0)$ and the Na'vi must fully cross each bridge until the end. At the end of a bridge the Na'vi will fall directly downward (only in the negative y direction). Every bridge can be thought of as a line segment that goes from $(start_x, start_y)$ to (end_x, end_y) and should be considered as containing the start point but not the end point. In other words, Na'vi can fall onto the start points but cannot fall onto the end points. Also note that $start_x$ will always be less than end_x and $start_y$ will always be less than end_y, so that the bridges always go upward. Finally, bridges never touch each other. Your solution should run in $O(n \log(n))$ time.

To get you started, we have provided you with a `solution_template` file. You are free to use any of the Python standard libraries and functions. You should know that Python's implmentation

of sorting (timsort) runs in $O(n \log(n))$ time. Python offers a standard min heap through the `heapq` standard library. Python does not offer a binary search tree as part of the standard library, as such we have provided you with an implementation of an AVL tree. See the `README` for more details.