CSC263 – Problem Set 2

Remember to write your **full name** and **student number** prominently on your submission. To avoid suspicions of plagiarism: at the beginning of your submission, **clearly state any resources (people, print, electronic) outside of your group, the course notes, and the course staff, that you consulted**.

Remember that you are required to submit your problem sets as both LaTeX `.tex` source files and `.pdf` files. There is a 10% penalty on the assignment for failing to submit both the `.tex` and `.pdf`.

Due Thursday, February 27, 2020, 22:00; required files: ps2sol.pdf, ps2sol.tex and triangle.py

Answer each question completely, always justifying your claims and reasoning. Your solution will be graded not only on correctness, but also on clarity. Answers that are technically correct that are hard to understand will not receive full marks. Mark values for each question are contained in the [square brackets].

You may work in groups of up to THREE to complete these questions.

1. **[total: 6]** You are given an array `A` that contains n **distinct** numbers. The task is to compute another array `B` such that for all `j`, we want `B[j]` to be the number of elements in `A` that appear after `A[j]` and are strictly smaller than `A[j]`.

   As an example, if you are given the input array `A = [8, 10, 4, 7, 3, 5, 9, 1]`, then you want to output the array `B = [5, 6, 2, 3, 1, 1, 1, 0]`. This is because, in `A`, there are 5 elements that appear after `A[0] = 8` and are smaller than `8` (i.e. `4, 7, 3, 5, 1`), there are 6 elements that appear after `A[1] = 10` and are smaller than `10` (i.e. `4, 7, 3, 5, 9, 1`), etc.

   Design an algorithm to compute `B` given `A` as input. The **worst case** complexity of your algorithm **must be** $\mathcal{O}(n \log n)$.

   This algorithm can be described in brief given what we have learned in class.

   Write a clear description of your algorithm (what data structure is used and how the algorithm utilizes the data structure) and justify its correctness and runtime. As usual, please use learned algorithms and analysis results from lectures and tutorials without repeating them.

2. **[total: 6]** Suppose that we want to generate a BST by inserting the keys 1 through n into an initially empty BST (assume $n > 1$). Assume that the insert sequence is a random permutation of $[1, 2, 3, \ldots, n]$ and each permutation is equally likely to occur. Answer the following questions.

   (a) What is the maximum possible height of the generated BST? Describe **four** different insert sequences that would result in a BST with the maximum height.

   (b) By picking a random permutation of $[1, 2, 3, \ldots, n]$ as the insert sequence, what is the probability that the resulting BST has the maximum height? Show detailed steps of your calculation with clear justification.

3. **[total: 12]** In this problem, you will design the data structure for implementing an ADT called `RESTAURANT-SET`. Below is the description of the ADT.

Objects: A collection of restaurants in the same city. Each `restaurant` has the following attributes:

- `restaurant.name`: a string which is the name of the restaurant. Each restaurant has a unique name, i.e., no two restaurants have the same name.
- `restaurant.cost`: a decimal value such as 22.99 that denotes the cost of a 1 person meal at the restaurant. For simplicity, assume that each restaurant's cost is unique, i.e., no two restaurants have the same cost.
- `restaurant.rating`: a decimal value between 0.0 and 5.0, e.g., 0.012, 3.1415926. For simplicity, assume that each restaurant's rating value is unique, i.e., no two restaurants have the same rating value.

Operations:

- `GET-RESTAURANT(R, name)`: returns the restaurant with name `name` if it exists in the `RESTAURANT-SET R`; returns `NIL` if the restaurant does not exist in `R`.
- `ADD-RESTAURANT(R, restaurant)`: Add a restaurant `restaurant` to the `RESTAURANT-SET R`. You may assume that `restaurant` has a unique name, a unique cost and a unique rating.
- `AVG-COST(R, r1, r2)`, `r1` $\leq$ `r2`: considers all restaurants that have a rating `r` such that `r1` $\leq$ `r` $<$ `r2`, and returns the average of the meal costs over all these restaurants. e.g. If `R` contains the following restaurants: `(A, 22, 3)`, `(B, 35, 4.5)`, `(C, 24, 3.9)`, `(D, 30, 4)` where each tuple represents `(name, cost, rating)`, then the query `AVG-COST(R, 3, 4)` must return the average of the costs of restaurants that have ratings ≥ 3 and < 4, which are `A` and `C`. Thus it must return $\frac{22+24}{2} = 23$.

Requirements: Let n be the size of R,

- `GET-RESTAURANT(R, name)` must have average-case runtime $\mathcal{O}(1)$.
- `ADD-RESTAURANT(R, restaurant)` must have **average-case** runtime $\mathcal{O}(\log n)$.
- `AVG-COST(R, r1, r2)` must have worst-case runtime $\mathcal{O}(\log n)$.

Give a *detailed* description of the design of your data structure by answering the following questions.

(a) **[2]** Which data structure(s) do you use in your design? What is the key of each data structure? What attributes does each node in your data structure(s) keep?

(b) **[2]** Explain concisely in English how your `GET-RESTAURANT` operation works, and justify its runtime.

(c) **[2]** Explain concisely in English how your `ADD-RESTAURANT` operation works, and justify its runtime. In particular, explain why all the attributes kept in each node can be maintained efficiently upon the addition of the new node.

(d) **[6]** Explain how your `AVG-COST` operation works and **write the pseudocode** of this operation, then justify its runtime.

Note: As usual, please do **not** repeat algorithm details or runtime analyses from class or the textbook — just directly refer to known results as needed.

Programming Question

The best way to learn a data structure or an algorithm is to code it up. In each problem set, we will have a programming exercise for which you will be asked to write some code and

submit it. You may also be asked to include a write-up about your code in the PDF/TEXfile that you submit. Make sure to **maintain your academic integrity** carefully, and protect your own work. The code you submit will be checked for plagiarism. It is much better to take the hit on a lower mark than risking much worse consequences by committing an academic offence.

4. [**total: 12**] In this problem, we will deal with the notion of **pseudo-similar** triangles.

   Each triangle is represented by a 3-tuple of positive numbers, specifying the sides of the triangle.

   We say that two triangles `t1` and `t2` are **pseudo-similar** if triangle `t2` can be obtained by rotating and reflecting triangle `t1`.

   For example, `t1 = (6, 9, 12)` is **pseudo-similar** to `t2 = (6, 12, 9)` since `t2` can be obtained by rotating and reflecting `t1`.

   $$\texttt{(6, 9, 12)} \xrightarrow{\text{rotate}} \texttt{(9, 12, 6)} \xrightarrow{\text{reflect}} \texttt{(6, 12, 9)}$$

   However, `t1 = (6, 9, 6)` is not **pseudo-similar** to `t2 = (6, 9, 9)`.

   We say that two triangles are the same **kind** if they are **pseudo-similar**.

   In Python, a triangle will be represented as a 3-tuple of positive integers.

   Your task is to write the function `num_triangle_kinds`, which determines the **number** of different kinds of triangles in the list. **(8 points)**

   Requirements:

   - Your code must be written in Python 3, and the filename must be `triangle.py`.
   - We will grade only the `num_triangle_kinds` function; please do not change its signature in the starter code. include as many helper functions as you wish.
   - You are **not** allowed to use the built-in Python dictionary or set.
   - To get full marks, your algorithm must have average-case runtime $\mathcal{O}(n)$. You can assume Simple Uniform Random Hashing.

   Write-up (4 points): in your `ps3.pdf/ps3.tex` files, include the following: an explanation of how your code works, justification of correctness, and justification of desired $\mathcal{O}(n)$ average-case runtime.