COMP 285 (NC A&T, Spr '22) Homework 5

Due. Friday, February 25th, 2022 @ 11:59 PM!

Homework Expectations: Please see Homework.

Exercises The following questions are exercises. We encourage you to work with a group and discuss solutions to make sure you understand the material.
Points This assignment is graded out of 100 points. However, you can get up to 120 points if you complete everything. These are not bonus points, but rather points to help make-up any parts you miss.

Fun with Data Structures and Hashing

Written Problems The following questions are to be submitted in written/typed form to gradescope.

1 Interview Practice: Moose Tree (20 pt.)

A moose comes to you with the following claim. They say that they have come up with a new kind of binary search tree, called `mooseTree`, even better than red-black trees!

More precisely, `mooseTree` is a data structure that stores comparable elements in a binary search tree. It might also store other auxiliary information, but the moose won't tell you how it works. The moose claims that `mooseTree` supports the following operations:

- `mooseInsert(k)` inserts an item with key k into the `mooseTree`, maintaining the BST property. It does not return anything. It runs in time $O(1)$.
- `mooseSearch(k)` finds and returns a pointer to node with key k, if it exists in the tree. It runs in time $O(\log(n))$.
- `mooseDelete(k)` removes and returns a pointer to an item with key k, if it exists in the tree, maintaining the BST property. It runs in time $O(\log(n))$.

Above, n is the number of items stored in the `mooseTree`. The moose says that all these operations are deterministic, and that `mooseTree` can handle arbitrary comparable objects.

1.1 Making Moose Trees (8 pt.)

You want to get a better understanding of the `mooseTree`. To do so, you think it'd be helpful to write an algorithm that takes as input a vector of unsorted elements and creates a mooseTree using the methods above.

We'll get you started.

```
algorithm makeMooseTree(A):
    // Input: A is a vector of n comparable elements.
    T = mooseTree() // creates an empty mooseTree.
    // FILL THIS IN.
    return T
```

[We are expecting: Completed pseudocode for the algorithm, the running time in $O(\cdots)$, and a short explanation justifying the running time.**]**

1.2 From Moose Trees to Sorted Vectors (8 pt.)

Now that you understand how to make `mooseTrees`, you consider the opposite. How can you go from a `mooseTree` to an **sorted** vector?[1]

We'll get you started [2].

```
algorithm getElements(T, A):
    // Input: T is a mooseTree with n elements.
    // Input: A is the vectors where we should add elements into.

    // FILL THIS IN.
```

[We are expecting: Completed pseudocode for the algorithm, the running time in $O(\cdots)$, and a short explanation justifying the running time.**]**

1.3 Wait a second... (4 pt.)

You think the moose's logic is a bit loosey-moosey. How do you know the moose is wrong?

Notes:

- You may use results or algorithms that we have seen in class without further justification.
- Proofs that just say "since it's impossible to do these operations that fast in BST, this data structure can't exist" is not sufficient.

[We are expecting: Use the results from the sections above (as well as results from class) to write a short explanation as to why the Moose is wrong.**]**

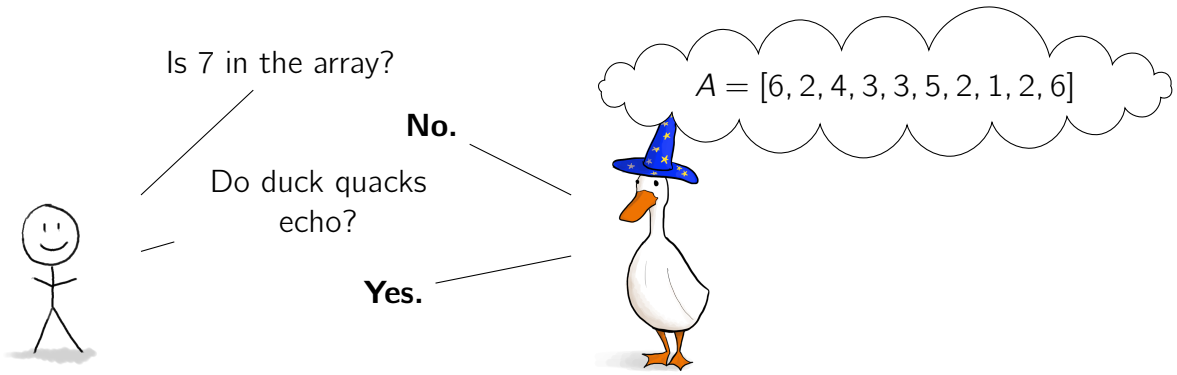
[1] If you're having trouble, take a look at Lecture 12, Slides 15-27

[2] Since the `mooseTree` is still a kind of binary search tree, you can access the root of `mooseTree` by calling `mooseTree.root()`.

2 Interview Practice: Wise Duck (20 pt.)

A wise duck has knowledge of an array A of length n, so that $A[i] \in \{1, \ldots, k\}$ for all i (note that the elements of A are not necessarily distinct). You don't have direct access to the list, but you can ask the wise duck *any* yes/no questions about it. For example, you could ask "If I remove $A[5]$ and swap $A[7]$ with $A[8]$, would the array be sorted?" or "are ducks related to grebes?"

This time you did bring a paper and pencil, and your job is to write down all of the elements of A in sorted order.[3] You are allowed to take all the time you need to do any computations on paper with the wise duck's answers, but the wise duck charges one bandito burrito per question.



Design an algorithm which outputs a sorted version of A which uses $O(k \log n)$ badito burritos. You may assume that you know n and k, although this is not necessary.

[We are expecting: Pseudocode and a clear English explanation of what it is doing. An explanation of why the algorithm uses $O(k \log n)$ ice cream cones. You do not need to prove that your algorithm is correct.**]**

[**HINT:** *One approach is to think first about what you would do for* $k = 2$*: that is, when* A *contains only the numbers* 1 *and* 2*. What information do you need to write down a sorted version of* A *in this case?*]

[3] Note that you don't have any ability to change the array A itself, you can only ask the wise duck about it.

3 Exercise: Hash Functions (20 pt.)

With 217 students currently enrolled in computer science[4] at North Carolina A&T University, we would like to keep track of student records.

The key for each person will be their 9-digit Banner ID.

3.1 Our own hash functions! (10 pt.)

Consider using a hash table of size 80 and the hash function $h(N) =$ sum of each digits of N. For example, $h(012345678) = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36$.

Is this a good hash function to use? List two reasons supporting your decision.

[We are expecting: The answer to whether this is a good hash function or not, and two reasons explaining why.**]**

3.2 Good hash families (10 pt.)

Consider a hash table of 99 buckets. For $m \in \{1, \cdots, 1000\}$, let $h_m(x) =$ last two digits of mx. For example, $h_4(01234567) =$ last two digits of $4 \times 01234567 = 4938268 = 68$. Define $\mathcal{H} = \{h_i : i \in \{1, \cdots, 1000\}\}$.

Is H a good family of hash functions? Justify your decision.

[We are expecting: The answer to whether $\mathcal{H}$ is a good family of hash functions or not, and a brief justification.**]**

4 Feedback: Homework Thoughts (10 pt.)

In order to improve the homework in future iterations, complete this form.

[We are expecting: You should submit the form with your @aggies.ncat.edu email to track your submission.**]**

[4] See here if you'd like to see other years!

Coding Problems The following questions are to be submitted as a ".zip" file on Gradescope.

5 Coding (50 pt.)

After completing the written portion of the assignment, you should submit it to Gradescope.

For the coding portion, you can get your starter code for C++ and Python. You are welcomed to implement the solution in whichever language you prefer.

Note that the starter code also include a few test cases you can run on repl.it. However, the full test suite is the one run on Gradescope.

Please reference the `README.md` included in your starter code for detailed instructions.

Submitting the Assignment

This assignment is a combination of written and programming questions. Both portions of the assignment should be submitted through Gradescope.

The "Homework 5: Fun with Data Structures and Hashing" assignment is the written portion, for which you should submit a **typed** response to the non-coding questions (questions 1-**??**). Each response should clearly be marked with its corresponding number. You are free to use the provided templates, print the questions and write your answers, or to simply type your responses on a blank document (whatever works for you).

The "Homework 5: Coding" is the programming portion of the assignment. For this portion, download the ".zip" file from replit and upload this ".zip" file as your answer to Gradescope. You can upload the assignment as many times as you want.