CSCI 102, Spring 2024, Final Exam

Write your name and netID at the top of the page. This exam is graded out of 19 points. You may use the backs of pages as scratch but please try to keep your answers on the front.

1 Short answer (10 points)

No need to justify your answers unless asked.

1. (1 point) Say I start with a sorted array with n elements and performed the following step T times: pick a random element and switch it with one of its neighbours. Assume that **no element was moved more than once**. What is the worst-case asymptotic computational complexity of sorting such an array with bubble sort in terms of both T and n?

2. (1 point) When we implemented hash maps with separate chaining, we stored an unsorted map at each position of the array. We could have used a AVL tree instead. What's the worst case asymptotic computational complexity of `put` and `get` with this implementation?

3. (2 points) Add the following numbers to a heap then remove them one by one: 1, 3, 5, 7, 9, 10, 8, 6, 4, 2. Draw the heap after each addition and removal.

4. (1 point) What is the asymptotic worst-case asymptotic computational complexity of an efficient algorithm that checks whether a node in an `EdgeListGraph` is in a directed cycle?

5. (1 point) Is it possible to implement a priority queue such that both addition and removeMin have worst-case asymptotic computational complexity better than $O(\log n)$? **Briefly explain your answer.**

6. (2 points) Add the following numbers an AVL tree and remove them in the order the were added: 1, 3, 5, 7, 9, 10, 8, 6, 4, 2. Draw the tree after each removal or addition.

7. (1 points) Redraw the tree from the previous question at the point when all 10 numbers have been added. Perform a series of rotations to bring 1 to the root. Draw the tree after each rotation.

8. (1 points) Say I implement `QuickSort(int[] arr)` and use the middle index as my choice for a pivot `int pivot = arr[arr.length / 2]`. What is the worst case asymptotic computational complexity if my array has size n? Draw an array with the elements $1, 2, \ldots, 10$ that achieves the worst case complexity for my algorithm.

2 Coding (20 points)

1. A. (1 point) Say you have a class `Class` with a method `int hashCode()` that has a small chance of giving you a wrong, random hash. Implement a method `int safeHashCode()` that has a substantially smaller chance of giving you an error and works by calling `int hashCode()` at least 3 times.

(1 point) Describe why you would want to perform your strategy for getting a safe hash on the hash code rather than the hash function.

B. Implement suitable hash codes for the following objects. These do not have to be efficient.

i) (1 point) `String` that is the same whether the string is forward or backwards, i.e. `csci102` and `201icsc` should have the same codes.

ii) (2 points) `Graph<Integer, E>` with directed edges, assuming there are no two vertices storing the same element, and the graph is fully connected.

2. A. (0.5 points) Write a method that takes in a binary tree of integer keys and checks if it is balanced `boolean isBalancedBST(BinaryTree<int> tree)`.

(2 points) Make sure your implementation has asymptotic computational complexity $O(n)$. Hint: use a recursive method. You may want to also use auxiliary variables!

B. (0.5 points)Write a method that takes in a binary tree of integer keys and checks if it is a binary search tree `boolean isBalancedBST(Tree<int> tree, int a, int b)`.

(2 points) Make sure your implementation has asymptotic computational complexity $O(n)$.

3. Throughout this question you may assume we have a class `BinarySearchTree<V>` that implements **both** `Map<Integer, V>` **and** `BinaryTree<Entry<Integer, V>>`.

(1 point) Write a method for `BinarySearchTree<V>`, `public int minKey()`, that returns the smallest key in the tree.

(1 point) Write a method `BSTSort(int[] arr)` that sorts `arr` in two stages: 1) add all elements to `BinarySearchTree<V>`, 2) iteratively remove the smallest element (effectively treating the BST as a priority queue).

(1 point) If the array has size n, what are the best and worst case asymptotic computational complexities of this method if we are **using a BST with regular addition and removal rules**? Write down the arrays that achieve these asymptotic computational complexities.

(1 point) Describe an alternative strategy for the second stage of `BSTSort(int[] arr)` that always takes $O(n)$ time. Implement this alternative. Hint: stage 1 gives you a BST. How can you get the keys in sorted order?

(1 point) If the array has size n, what are the best and worst case asymptotic computational complexities of this method if we are **using a BST with AVL addition and removal rules**?

4. (5 points) A **path of alternating direction** in a graph is a path between two nodes that is made up of edges of alternating direction, for example $A \rightarrow B \leftarrow C \rightarrow D \leftarrow E \rightarrow F$. Write a method `boolean hasAlternatingPath(Graph<V, E> graph, Vertex<V> start, Vertex<V> end)` that determines if there is an alternating path between the vertices `start` and `end`. You may assume you have access to `ArrayStack<E>`, `ArrayQueue<E>`, `HashMapSC<E>` which are implementations of `Stack<E>`, `Queue<E>` and `Map<E>`. Hint: You'll want to have two of each structure from the searches we covered in class, one for paths that end in $\rightarrow$ and another for paths that end in $\leftarrow$.