CSCI 102 assignment 6 – AVL trees and PriorityQueues

Nov 13

1 AVL trees

Reimplement your code from assignment 5 using AVL rules for `addCount` and `remove`. You may use code from the class website.

2 Priority Queues for sorting

In this section you will show that any implementation of `PriorityQueue` is also an implementation of a sorting algorithm.

- Use your implementation of a `PriorityQueue` from assignment 3 to sort an array of `Integer`s by adding all the integers to a `PriorityQueue<Integer, Integer>` and then removing them one at a time. Call your method `Integer[] sort(Integer[] arr)`.
- Add a `main` method that sorts then prints the array `[6, 7, 2, 4, 3, 8, 1, 5]`.
- In assignment 3 I left you implement `PriorityQueue` however you'd like. In this assignment create an implementation called `UnsortedPriorityQueue` that adds entries by adding them to the end of a list and removes them by searching the list for the lowest priority element.
- Say your `UnsortedPriorityQueue` has n elements; what are the asymptotic computational complexities of adding and removing? What's the asymptotic computational complexity of `sort` using `UnsortedPriorityQueue`? What sorting algorithm that we covered in class does `sort` with `UnsortedPriorityQueue` implement?
- Create an implementation called `SortedPriorityQueue` that adds entries by adding them to a list in the position **that keeps the list sorted (priorities are increasing)** and removes them by removing the first element of the list.
- Say your `SortedPriorityQueue` has n elements; what are the asymptotic computational complexities of adding and removing? What's the asymptotic computational complexity of `sort` using `SortedPriorityQueue`? What sorting algorithm that we covered in class does `sort` with `SortedPriorityQueue` implement?
- Say you've implemented a `VeryCoolPriorityQueue` so that when it has n elements, adding to it is $O(\log n)$ and removing is $O(\log n)$. What's the asymptotic computational complexity of `sort`? Is this more or less efficient than the sorting algorithm you derived above; how does it compare to the best possible complexity of a sorting algorithm?

Please submit your code and answers to the questions in a zipped folder on Brightspace by Nov 20.