Written Assignment #3

Content Covered: PA3 Reflection, Balanced Binary Search Trees

---

Submission Process, Late Policy and Grading

Due Date: 4/23/23 @ 11:59PM
Total points: 30

Your written solution may be either handwritten and scanned, or typeset. Either way, you must produce a PDF that is legible and displays reasonably on a typical PDF reader. This PDF should be submitted via autolab as WA3. You should view your submission after you upload it to make sure that it is not corrupted or malformed. Submissions that are rotated, upside down, or that do not load will not receive credit. Illegible submissions may also lose credit depending on what can be read. Ensure that your final submission contains all pages.

You are responsible for making sure your submission went through successfully.

Written submissions may be turned in up to one day late for a 50% penalty.
No grace day usage is allowed.

Problem 1 - Pre-Computation

[16/30 points]

For PA3, the StreetGraph class used to store the maps we were searching was implemented using an EdgeList data structure. The first function you had to implement created an external AdjacencyList that you could use for searching your graph.

a. **[4 points]** Derive the unqualified worst-case runtime (in terms of |V| and |E|) to perform a BFS search on a Graph that is implemented using an EdgeList. Show your work. **Note:** this is not what you implemented in PA3.

b. **[2 points]** For PA3, to perform a BFS search of the graph, we first computed the AdjacencyList and then searched the graph. What is the total unqualified worst-case runtime (in terms of |V| and |E|) to compute the AdjacencyList *and then* perform the BFS search. Justify your answer.

c. **[2 points]** Given your answer to part (b) is it worth it to take the time to compute the AdjacencyList before performing BFS if your input Graph is implemented using an EdgeList? Why or why not?

d. **[3 points]** Now imagine instead of searching a Graph, we are searching an Array of data. If the Array contains *n* elements, what is the unqualified worst-case runtime to:

   i. Find a specific value in the array if it is unsorted?
   ii. Find a specific value in the array if it is sorted?
   iii. Sort the array?

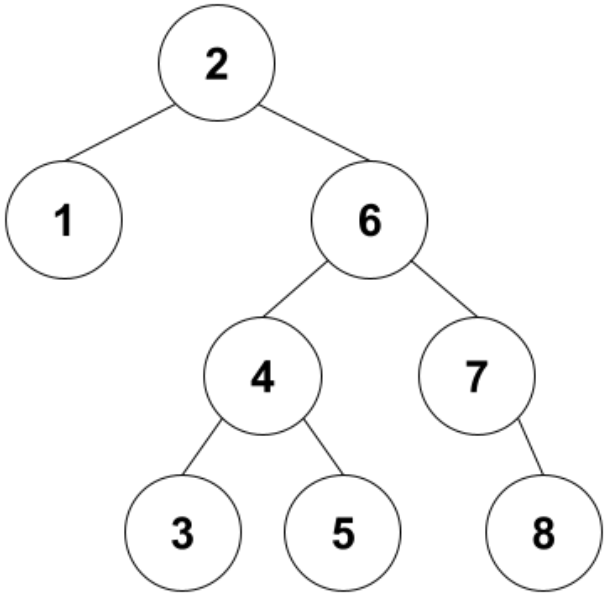
e. **[3 points]** Based on your answers for (d):

   i. What is the runtime to perform *n* searches in row on the unsorted array?
   ii. What is the runtime to sort the array first *and then* perform the *n* searches?
   iii. Is it worth it to sort the Array first and then perform the searches?

f. **[2 points]** Now imagine that the Array contains times for competitors at an event to complete a race, and you want to find the top 3 times. Is it worth it to sort the Array first or not? Explain.

Problem 2 - Balanced Binary Trees

[10/30 points]



a. **[4 points]** The above tree meets the structural requirements for a Red-Black tree. Determine the balance factor for each node, and a valid coloring for each node. When writing out your answer, write out 1-8 each on its own line, followed by the balance factor and color for that node.

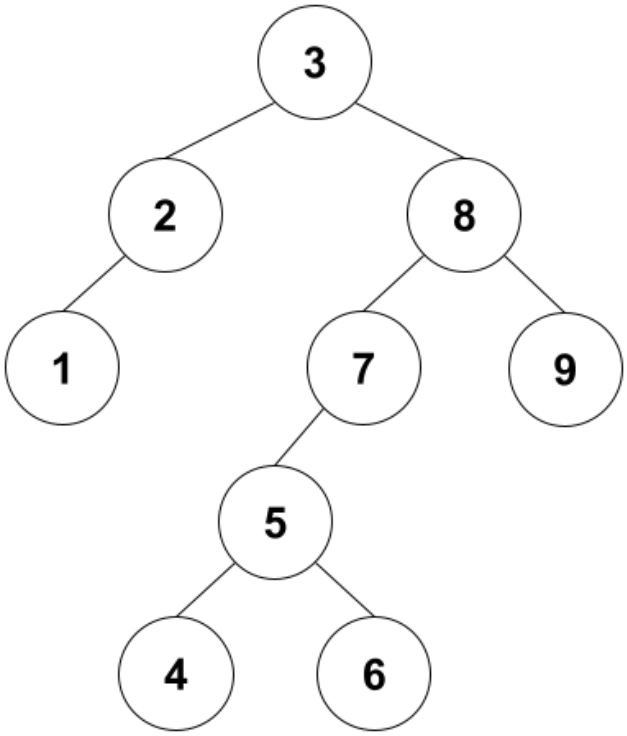
b. **[2 points]** Does the above tree meet the structural requirements to be an AVL tree? Explain why or why not.

c. **[2 points]** Give a value that could be inserted into the above tree and not break the Red-Black tree constraints. Give a value that could be inserted into the above tree that would break the Red-Black tree constraints and therefore require additional operations to fix. (Assume the tree does not allow duplicate values)

d. **[2 points]** Recall that a Tree is also a Graph, therefore we can perform BFS and DFS on trees as well. Does performing a Depth-First Search starting at the root of an arbitrary BST to an arbitrary node of the tree find the shortest path to that node? Explain why or why not.

Problem 3 - Fixing Balanced Binary Trees

[4/30 points]



a. **[2 points]** The above tree does not meet the structural contraints to be an AVL tree OR a Red-Black tree. What is the **shortest series** of rotations you could perform that would cause the tree to meet the structural constraints of an AVL tree. Write your answer as a series of rotations, where each rotation is either: `rotateLeft(N)` or `rotateRight(N)` where `N` is the value of the node that you are rotating around.

b. **[2 points]** After performing the rotations from part (a) to make the tree meet AVL tree constraints, will it also meet Red-Black tree constraints? Why or why not?