Advanced Data Structures ✧ Spring 2025

Homework 0

Due Tuesday, January 28, 2025 at 9pm Central Time

- This homework is primarily meant as a (self-)diagnostic tool to ensure that you have the necessary algorithmic background/experience to succeed in this class. However, it will count toward your final course grade.
- Submissions should be typeset in LaTeX; a solution template will be available on the course web site soon. Submit your solution for each numbered problem on Gradescope as a separate PDF file.
- As a general rule, your solution for each numbered problem should fit in *at most three* typeset pages (not including figures and references); two pages is usually a good target. If your solution requires significantly more than three pages, you are probably including too much detail. Write breadth-first, not depth-first.
- You may use any source at your disposal—paper, electronic, or human—but you must cite every source that you use, and you must write everything yourself in your own words. All submissions must include a complete list of sources and collaborators, **even if that list is empty**.
  - Cite written sources exactly as you would in a research paper; BibTeX is your friend.
  - If you use ChatGPT or another LLM, you must include a *complete* transcript of your prompts and the LLM's responses. (This won't count toward your 3-page budget.)
- Clarity, precision, and style matter more than correctness; correctness matters more than efficiency. Solutions that are clear but incorrect are worth more than solutions that are correct but unreadable. Clear and correct solutions that are suboptimal are better than optimal but opaque solutions, or fast but incorrect solutions. In particular, submissions that mirror ChatGPT's sloppy and repetitive writing style will be heavily penalized.
- *Stars indicate harder problems; these *might* become easier later in the semester. ★Larger stars indicate problems that I don't know how to solve; these may or may not be open.

Rubric: All problems will be graded on the same crude four-point (sic) scale:

0 = Missing, completely unclear, complete misunderstanding, or a picture of a clown (hopefully rare)
1 = A good-faith but incorrect solution that shows understanding of the problem
2 = Significant progress toward a clear, correct, and efficient solution
3 = A mostly clear, mostly correct, and nearly efficient solution
4 = Full credit: A completely clear, correct, and efficient solution
5 = Extra credit: Above and beyond (probably rare)

0. List up to three theory/algorithms classes you have previously taken *where a significant fraction of the audience were graduate students*, such as CS 473, CS 474, CS 475, CS 498 TC, CS 574, or CS 579. For each class, list the course name and number, the term when you took the course, the instructor's name, and the grade you received. In addition, for classes you took somewhere other than Illinois, please include a brief description of the course and the URL of a course web page (or at least a detailed syllabus, not just a catalog description).

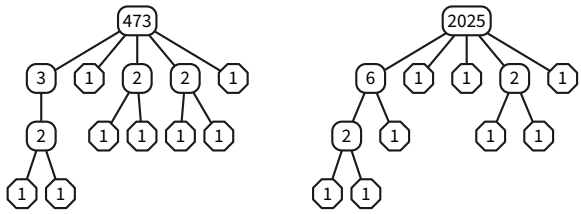
1. *Runnary* (short for ***run**-length-encoded bi**nary***) is a compressed variant of the standard binary representation for positive integers. Instead of listing the bits in the binary representation directly, we record the lengths of runs—maximal substrings of equal bits. Because the first run of a binary number always contains 1s, and later runs alternate between 0s and 1s thereafter, we only need to record the run lengths, not their contents.

   For example, the standard binary representation of 473 is 111011001, so its runnary representation is the sequence $[3, 1, 2, 2, 1]$. Similarly, the runnary representation of $2025 = 11111101001_2$ is the sequence $[6, 1, 1, 2, 1]$.

   (a) Describe and analyze an algorithm RunnaryInc, which changes the runnary representation of some positive integer N into the runnary representation of $N + 1$. In particular, describe the precise data structure your algorithm uses to store the input and output sequences.[1] For example, RunnaryInc$([3, 1, 2, 3]) = [3, 1, 2, 2, 1]$.

   (b) The elements of the runnary representation of an integer are themselves integers, which we can represent *recursively!* The resulting representation of any integer is an ordered rooted tree. The representation for 1 is a single leaf; the representation for any integer $N > 1$ is a node whose subtrees recursively represent the sequence of numbers in the runnary representation of N.

   For example, the following trees are the recursive runnary representations of the integers 473 and 2025. (The node labels are only for illustration; they are not part of the actual representation.)



   Describe and analyze an algorithm RecRunnaryInc, which changes the recursive runnary representation of some integer N into the recursive runnary representation of $N + 1$. As in part (a), describe the precise data structure you use to represent the input and output trees.

   (c) Give a *tight* analysis of the running time of your RecRunnaryInc algorithm *as a function of the number N represented by the input tree*. That is, find a function f such that your algorithm always runs in $O(f(N))$ time, ***and*** show that for infinitely many N, your algorithm requires $\Omega(f(N))$ time.

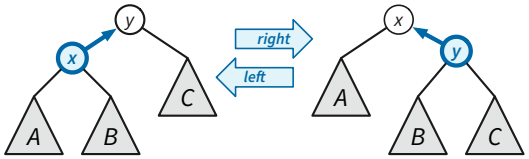
[1] Remember that a single Python instruction does *not* necessarily run in $O(1)$ time!

2. Let T be a binary tree whose nodes store distinct numerical values. Recall that T is a ***binary search tree*** if and only if either (1) T is empty, or (2) T satisfies the following recursive conditions:

   - The left subtree of T is a binary search tree.
   - All values in the left subtree of T are smaller than the value at the root of T.
   - The right subtree of T is a binary search tree.
   - All values in the right subtree of T are larger than the value at the root of T.

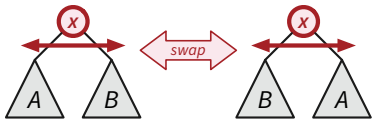
   Describe and analyze an algorithm to transform an *arbitrary* binary tree T with distinct node values into a binary *search* tree, using ***only*** the following operations:

   - Rotate an arbitrary node. Rotation is a local operation that decreases the depth of a node by one and increases the depth of its parent by one.



Left to right: right rotation at x. Right to left: left rotation at y.

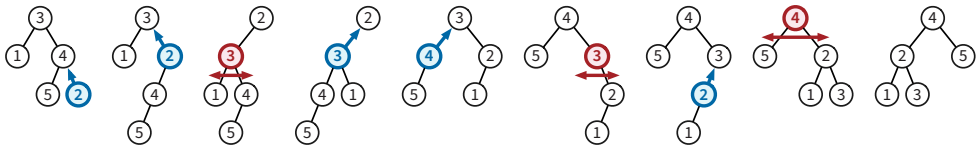
   - Swap the left and right subtrees of an arbitrary node.



Swapping the subtrees of x

For both of these operations, some, all, or none of the subtrees A, B, and C may be empty.

The following example shows a five-node binary tree transforming into a binary search tree in eight operations; see the next page for another example with no rotations.



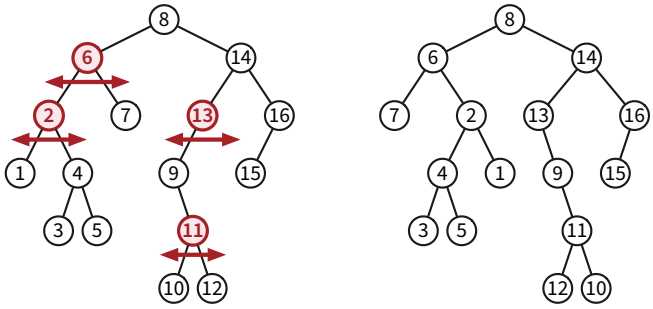
"Sorting" a binary tree in eight steps: rotate 2, rotate 2, swap 3, rotate 3, rotate 4, swap 3, rotate 2, swap 4.

Your algorithm can ***only*** modify T using rotations and swaps. It cannot change the key stored at any node; it cannot directly modify parent or child pointers; it cannot allocate new nodes or delete existing nodes. On the other hand, you may *compute* anything you like for free, as long as that computation does not modify T. In other words, the running time of your algorithm is *defined* to be the number of rotations and swaps that it performs.

For full credit, your algorithm should use as few rotations and swaps as possible in the worst case. *[Hint: $O(n^2)$ operations is not too difficult, but you can do better.]*

3. CS 124 students Chef Gallon and Fade Waygone wrote some inorder traversal code for their MP on binary search trees. To keep things simple, they wisely chose the integers 1 through n as their search keys. Unfortunately, their code contained a subtle bug (which was nearly impossible to track down, thanks to version inconsistencies between Fade's laptop, the submission/grading server, and Oracle's ridiculous licencing terms) that would sporadically swap left and right child pointers in some binary tree nodes (just like in the previous problem). As a result, their traversal code rarely returned the search keys in sorted order.

   For example, given the binary search tree below, if the four marked nodes had their left and right pointers swapped, Chef and Fade's traversal code would return the garbled "inorder" sequence $7, 6, 3, 4, 5, 2, 1, 8, 13, 9, 12, 11, 10, 14, 15, 16$.



   Chef and Fade submitted the output of several garbled traversals, but before they could submit the actual traversal code, Fade's laptop was infested with bees.[2] After receiving a grade of 0 on their MP, Chef and Fade argued with their instructor that they should get *some* partial credit, because the sequences their code produced were at least consistent with correct binary search trees, and anyway the bees weren't their fault.

   Design and analyze an efficient algorithm to verify or refute Chef and Fade's claim (about the binary search trees, not about the bees). The input to your algorithm is an array $A[1..n]$ containing a permutation of the integers 1 through n. Your algorithm should output TRUE if this array is actually the inorder traversal of a binary search tree with keys 1 through n, possibly with some left and right child pointers swapped, and FALSE otherwise. For example, if the input array contains $[5, 2, 3, 4, 1]$, your algorithm should return TRUE, and if the input array contains $[2, 5, 3, 1, 4]$, your algorithm should return FALSE.

   [Hint: If you use a dynamic-programming algorithm, please follow the same outline used in CS 374 and CS 473; in particular, your solution must include an explicit standalone English description of the underlying recursive function.[3]]

[2] Something, something, B-tree, something.

[3] Also, calling your memoization array "dp" makes the baby Jesus cry.