AVL Tree Heist
The problem never asks you to make any AVL Tree rotations, but it does ask you to be able to detect height imbalances. Thus, you should simply maintain the height of a node in each node and update this on every insert operation. If a tree isn't a valid AVL tree at any point in time, then at the end of the insertion, some node on the ancestral path will be unbalanced. Thus, after you recursively insert a node, before you return your answer from the insert, check the heights of the left and right (which should have already been updated). If this difference is more than 1, then the answer for the case is NO. Just mark a global flag to this effect and continue processing all of the insertions. At the end of processing all the insertions, look at the flag to output.

Two common errors on this problem: Trying to answer as soon as an imbalance occurs without reading in all of the data (so that a future case is read in incorrectly), and only checking if the end tree is balanced or not. It's possible to insert nodes in such a way that in the middle the tree isn't balanced, but by the end, it is.

Ordering Paper
Each sheet of 11" x 17" paper you order stores 4 pages of a single exam. If an exam booklet needs n pages, then you must order \( \lceil \frac{n}{4} \rceil \) sheets of paper. (In code this is just (n+3)/4 using integer division.) Then, for each exam, calculate the number of sheets of paper and multiply that by the number of students taking that exam. Finally, sum this product over all exams in a test case to get the result for the test case.

Sorting Exams
This problem is identical to the Week 2 Practice Problem: Add All. The key observation is that our cost function of merging two stacks is the sum of those stacks, which is the same exact cost function in Add All. The observation in that problem was that all things being equal, we'd rather have our current merge operation be of minimum cost, which means merging the two minimum stacks. (The exchange argument formally proves this.) Then, we want to find the two new minimum stacks and repeat until there is only one stack. Thus, store all numbers in a priority queue, delete min twice, add these two numbers together and add that to the total cost, then put this sum back in the priority queue and repeat this n-1 times total, where n was the original number of stacks of exams.
**Tree Sales**

We first note that a single query could be very expensive if we just stored each person's sales in their node in the tree structure. Then, to answer a single query, we would have to visit each node in the subtree of the queried node, which could be up to 100,000 steps and this could happen 100,000 times. (In reality, it wouldn't be this big, but we could still get close to $50000^2$ operations total over 100,000 operations.) Thus, we need an idea to speed up a query. What if we store the sum of ALL nodes in a subtree at the root of that subtree? So, the old storage method might look like what's drawn on the left and we can store tree sums instead as shown on the right, to speed up our ability to answer queries:

Now, we have O(1) answers to queries (as long as we can access a node in O(1) time). But...how long does it take to update all the values for the tree on the right? Basically, a single sale affects each node on the ancestral path of the node making the sale. So say the node that currently has 11 makes a sale of $20, here are the edits to both tree structures:

Now, we have O(1) answers to queries (as long as we can access a node in O(1) time). But...how long does it take to update all the values for the tree on the right? Basically, a single sale affects each node on the ancestral path of the node making the sale. So say the node that currently has 11 makes a sale of $20, here are the edits to both tree structures:

Now, we have O(1) answers to queries (as long as we can access a node in O(1) time). But...how long does it take to update all the values for the tree on the right? Basically, a single sale affects each node on the ancestral path of the node making the sale. So say the node that currently has 11 makes a sale of $20, here are the edits to both tree structures: