Q1

For the first problem, the most violent and direct way is to go through all the other nodes with one black node fixed, and find the sum of the corresponding edges. The time complexity of this method is too high. This is because our object of study should not be focused on nodes, but should be translated into edges themselves. We observe that for any edge, its contribution to the final output depends on how many pairs of black nodes are connected through it. So let's consider that we can divide all the black nodes into two parts, one is below this edge, or in other words, is in the subtree with the node pointed to by this edge as root; And the other part is what's left. The two black nodes should be paired one by one, so the number of times the edge plays a role should be the product of the number of the two parts, and the actual role it plays needs to take its own weight into account, which comes to the core formula of the entire program. Then, around this core formula, we use the method of traversing the tree to study every edge, in other words, to find the number of black nodes that are below this edge. So that's the general idea of the program. In contrast, this program is significantly less complex than the others, and the idea is very smooth, the only difficulty is to observe the quantitative relationship behind this formula.

Q2

On the second question, I chose to maintain this list of prices using a list and a BST. When I first find the CLOSEST_ADJ_PRICE, I'll traverse the list, find the difference between the two adjacent prices, and find the smallest one. Finally, record the index of the last price to facilitate subsequent operations. In the subsequent use of this function, instead of iterating through the difference from scratch, I start with the last recorded index and update the minimum by comparing backward. For the solution for CLOSEST_PRICE, I will first construct the price as a BST. For the first time, I will traverse the BST in inorder, to find the difference between the current node and the next node, and determine the minimum difference. In the subsequent solution, instead of iterating over BST from the beginning, I chose to find the successor and predecessor of this added node every time the BUY function added a new node to determine whether the difference caused by the newly added node was smaller than the original minimum value. These specific comparisons are done in the BUY function, and the task after CLOSE_PRICE is to place the minimum value in the final list to be printed. The original job of the BUY function is to insert a new element in the list and BST. So that's the general idea, in the process of solving the problem, the general idea, that is, the use of list and BST is not difficult to think of. But subsequent optimizations, such as the use of successor and predecessor, do require a bit of inspiration. This approach is much less complex than mindless traversal.