Problem 1: Pairs

Solution 1: You can define a hash table and hash each a_i $(1 \leq i \leq n)$ into the hash table. Then for each a_j $(1 \leq j \leq n)$, you can find $\frac{k+3a_j}{2}$ or $\frac{-k+3a_j}{2}$ in the hash table in $O(1)$.

Solution 2: Sort array $A = (a_1, a_2, \dots, a_n)$ in $O(n\log n)$. Then for each a_j $(1 \leq j \leq n)$, you can find $\frac{k+3a_j}{2}$ or $\frac{-k+3a_j}{2}$ in the sorted array in $O(\log n)$.

Problem 2: Binary Tree

You can recursively construct a **binary search tree** based on the given preorder traversal. In the next step, you can traverse the binary search tree and compute the level of each node during the traversal and update the sum.

Problem 3: Quartile

Note that in this problem, we are asked to find the first quartile multiple times (probably $\Omega(n)$ times), and thus in each time we have to compute the first quartile in a sublinear time. It means the algorithm which find the $\lceil\frac{n}{4}\rceil$-th smallest element in time $O(n)$ doesn't work.

The main idea is to define two heaps and update them in each step. The first one is a max-heap which stores the $\lceil\frac{n}{4}\rceil$ smallest elements and the second one is a min-heap which stores the $n - \lceil\frac{n}{4}\rceil$ largest elements where n is the number of elements seen so far. As an example assume the following elements seen so far:

$$(23, 17, 6, 33, 59, 98, 12, 76)$$

In this example 6 and 12 would be in the first heap (max-heap), and the remaining elements would be in the second heap (min-heap).

Note that at each step the first quartile is the maximum element of the first heap (max-heap). Now you have to design an algorithm to update both heaps after each insertion in $O(\log n)$.