6.1. Given an array of pairs $(x, y)$, sorted by $x$. Construct a Treap from it in $O(n)$ time.

6.2. Given an array of pairs $(x, y)$. All $x$ are different, and $y$ can be the same. Check that a Treap can be constructed in a unique way.

6.3. Show that if you take a binary search tree without any balancing algorithm and add elements to it in descending order of $y$, you get a Treap.

6.4. Given an array of numbers from 1 to $n$. Learn how to process requests in $O(\log n)$ using a Treap with implicit keys: 1) reverse the array segment from $l$ to $r$, 2) find the $i$-th element.

6.5. Let's try to build a Treap without storing the $y$ keys. In those places of the code where the comparison of $y$ is performed, we will choose a random variant with a probability of 50%. Show that some sequence of operations can lead to the tree with expected height $\Omega(n)$.

6.6. Let the binary search tree have no nodes with one child (that is, each inner vertex has exactly two children). Is it enough to guarantee that the height of such a tree is $O(\log n)$?

6.7. Show how, on the basis of a binary search tree with implicit keys, to make a Union-Find data structure with time complexity $O(\log n)$.

6.8. In Treap with implicit keys, implement an operation $splitAfter(x)$, which splits the tree containing the node $x$ into two trees, one tree containing elements before $x$, another tree containing all other elements. Consider that the sizes of the subtrees **are not calculated**, but all elements have a pointer to their parent in the tree.