№1

Сначала посчитаем высоты деревьев. Пусть б.о.о. h1 >= h2.

Затем удалим минимальный элемент из T2. Обозначим его min2, измененное дерево – T2', а его высоту – h2'.

В T1 будем идти вправо, пока не придем в узел x, высота поддерева которого равна h2' или h2'+ 1.

Создадим новое дерево корнем которого является min2, а его левым и правым сыном – x и T2'. Это дерево будет бинарным и сбалансированным.

Теперь вставим это дерево на место, где стоял x.

В конце балансируем полученное дерево, начиная с места вставки.

Все операции занимают не более O(log n) времени.

№4

a) Можно применить стандартный алгоритм поиска следующего элемента. Если у данной вершины есть правое поддерево, переходим в него и идем максимально влево. Если же правого поддерева нет, идем вверх, начиная с данной вершины до тех пор, пока не встретим вершину, которая является левым сыном от ее родителя. Этот родитель и будет искомой вершиной.

Т.к. в среднем случае все ключи x и y разбросаны равномерно, и максимально нам надо будет пройти либо вниз, либо вверх по всей высоте дерева, то асимптотика O(h)=O(log2n).

б) В каждом узле будем также хранить указатель на следующую вершину.

При добавлении нового узла будем искать следующий и предыдущий узел с помощью алгоритма из п. а). У предыдущего узла обновим указатель на только что вставленный, а у вставленного элемента указателю присвоим следующий. Асимптотика – O(3 \* log n) = O(log n).

Аналогичные действия производим перед самым удалением.

№5

Используем стандартный обход дерева. Будем считать, что веса ребер хранятся в узлах под ребрами (у root это 0).

sum = 0;

max = 0;

BypassTree(Node\* node) {

sum += node->weight;

if (sum > max)

max = sum;

if (node->left != nullptr)

BypassTree(node->left);

if (node->right != nullptr)

BypassTree(node->right);

sum -= node->weight;

}

BypassTree(root);

№6

a) Предварительно обернем все элементы в бинарное дерево given\_tree. В каждом узле будем также хранить set – указатель на дерево-множество, в котором находится элемент, соответствующий этому узлу.

Next(x): В given\_tree находим элемент x. Переходим в соответствующее множество. В нем ищем элемент x. Затем стандартным алгоритмом ищем в этом множестве (дереве) следующий за x элемент. Асимптотика: O(3\*log n) = O(log n).

Merge(x, y): В given\_tree ищем x и y. Пусть б.о.о. y->set меньшего размера, чем x->set. Поэлементно копируем все элементы y->set в x->set и заменяем y->set на x->set. Каждое копирование работает за O(log n).

Теперь заметим, что при перемещении числа из одного множества в другое, размер полученного множества будет не менее, чем в 2 раза больше, чем размер исходного множества (того, где лежало число) (мы перемещаем числа из меньшего множества). Поэтому каждое число может быть перемещено не более log n раз. Получили O(n\*log2n). Но при q запросах мы можем затронуть максимум 2q чисел. Поэтому асимптотика O(2q\*log2n) = O(q\*log2 n).

Т.о., общая асимптотика – O(q\*log2n).