1 Обязательная часть

1. ***Персистентный стек.***
   Заведем массивчик. У каждого элемента массива два поля: само значение, которое хранится top стека, и "индекс"предыдущей версии стека (что-то вроде ссылки на родителя). В самом начале в массиве храним пустой элемент - $<$null, null$>$.
   Пусть теперь у нас есть массив mass с n элементами (то есть уже n раз мы модифицировали наш стек). Как добавить x к iой версии стека? Добавляем в массив новый элемент: $mass[n+1] =< x, i >$. Как удалить элемент из iой версии стека? Добавить в mass новый элемент - его индекс будет равен ind-полю предка iой версии, а значение val-полю предка: *mass.push(mass[mass[i].ind].val, mass[mass[i].ind].ind)*. Все эти операции, очевидно, делаются за единицу.

2. ***k минимальных на отрезке.***
   Дополнительно в каждой вершине BST храним минимум в поддереве. Теперь, чтобы найти k минимумов, можно k раз найти минимум (звучит не очень). То есть сначала k раз находим минимум и удаляем его, чтобы освободить пространство следующему минимуму. А потом их все возвращаем. Получится как раз $\mathcal{O}(\log n)$.

3. ***Стакан.***
   Храним дерево отрезков по y, а в каждой его вершине - BST по x. Перечисление всех точек сможем сделать за $\mathcal{O}(\log n + k)$.

4. ***AA-insert.***

```
void insert(Node*& v, int x){
    if (v == NULL){
        v = Node(NULL, NULL, x, 1);
    }else if (x > v->val){
        insert(v->r, x);
    }else{
        insert(v->l, x);
    }

    //Вращения

    if (v->l->h == v->h){
        Node* tmp = new Node(v->l);
        v->l = tmp->r;
        tmp->r = v;
        v = tmp;
    }

    if (v->r->r->h == v->h){
        Node* tmp = new Node(v->r);
        v->r = tmp->l;
        tmp->l = v;
        tmp->h++;
        v = tmp;
    }
}
```

2 Дополнительная часть

1. ***k* минимальных нельзя искать быстро.**

   Предположим, можно найти их за $\mathcal{O}(k + \log n)$. Но в таком случае мы сможем и отсортировать массив быстрее, чем за $\mathcal{O}(n \log n)$. Как? Давайте достанем из BST $\log n$ минимальных. А затем отсортируем их за $\log n \cdot \log\log n$. Неплохо. Повторим эту операцию $\frac{n}{\log n}$ раз. В итоге все элементы мы сможем отсортить за $\frac{n}{\log n} \cdot \log n \cdot \log\log n = n \log\log n$. Но ведь такое импоссибл.