Построение сбалансированного дерева

Достаточно часто встречается задача построения бинарного дерева поиска, содержащего все элементы заданного массива. Возможен тривиальный алгоритм:

```
var t:tree, a:array[1..N] of integer;
begin
    t := nil;
    for i := 1 to N do
        add(t, a[i]);
end;
```

Но он имеет квадратичную сложность и может построить несбалансированное дерево.

Давайте попытаемся построить дерево минимальной высоты, содержащее элементы заданного массива.

Заметим, что в случае правильного решения задачи, высота получившегося дерева будет равна **[log2 N]**, то есть округлённому вверх логарифму N по основанию 2.

Как устроено оптимальное построение дерева? Если мы выбрали в качестве корня какую-то вершину, то её правые и левые поддеревья должны иметь высоту не больше **[log2 N] - 1** = **[log2 N - 1]** = **[log2 (N / 2)]**. Таким образом, если мы выберем корень так, чтобы в его правом и левом поддереве оставалось не больше **N/2** вершин, мы решим задачу.

Из свойства бинарного дерева поиска мы знаем, что все вершины в левом поддереве должны быть меньше значения в корне, а все вершины в правом — больше. Таким образом, в качестве корня нам идеально подойдёт элемент, стоящий на **N/2**-ом месте в отсортированном массиве: есть **N/2-1** элементов меньше него, которые пойдут в левое поддерево, и **N/2** элементов больше него, которые пойдут в правое поддерево.

Сначала отсортируем исходный массив **a**, затем с помощью рекурсивной функции **do_gen_tree(i, j)** построим дерево. В качестве параметров функция принимает отрезок массива **a**, для которого нужно построить дерево. Тогда при работе рекурсивной функции возможны два варианта:

- Передан пустой отрезок (**i** > **j**). В этом случае дерево строить не нужно, достаточно вернуть **nil**.
- Передан не пустой отрезок. В этом случае мы берём элемент, стоящий в середине отрезка (то есть на позиции **k**, равной **i + (j - i) div 2**), делаем его корнем нашего нового дерева, а в качестве левого и правого поддерева привязываем рекурсивные вызовы этой же функции от интервалов **[i, k - 1]** и **[k + 1, j]**.

Функция будет работать корректно, так как при вызове от пустого интервала она работает корректно, а при вызове от непустого интервала, она вызывает себя рекурсивно с уменьшенным интервалом.

```
function gen_tree(var a: array[1..N]):tree;
    function do_gen_tree(i, j: integer):tree;
    var t: tree; k: integer;
    begin
        if i > j then
            do_gen_tree := nil;
        else begin
            k := i + (j - i) div 2;
            new(t);
            t^.value := a[k];
            t^.left := do_gen_tree(i, k - 1);
            t^.right := do_gen_tree(k + 1, j);
            do_gen_tree := t;
        end;
    end;

begin
    sort(a);
    gen_tree := do_gen_tree(1, N);
end;
```