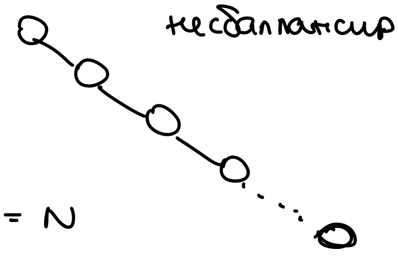
Декартово дерево

1. Улучшение рез-в
2. Рекурсивно → SPLAY-дерево
3. Неаксиоматичное дерево.

Дерево поиска: H – высота

Find	$O(H)$
Insert	$O(H)$
Remove	$O(H)$

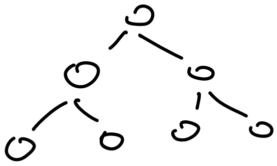
несбалансир.



$H = N$

Реализуем самобалансирующееся дерево поиска.

$\boxed{H \sim \log_2 N}$ ← приблизительное



$\} H = \log_2 N$

Декарт → система координат

```
Node {
     value
}
```

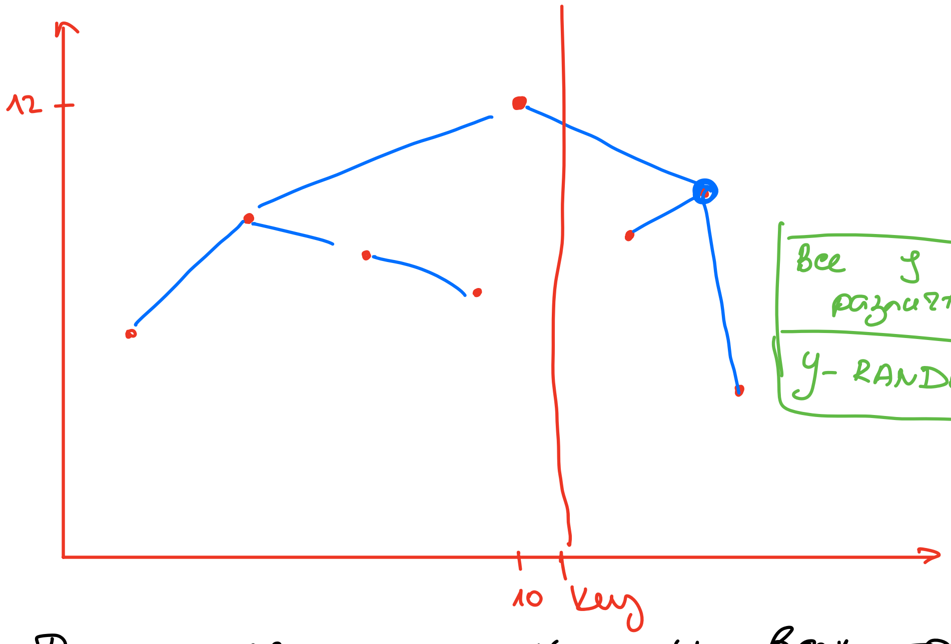
Node {

key → x

priority → y

}

(10, 12)



Все y различны

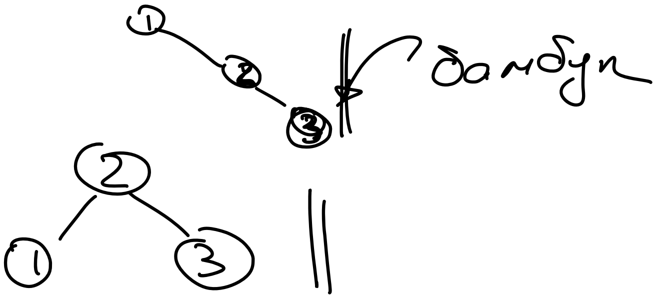
y - RANDOM

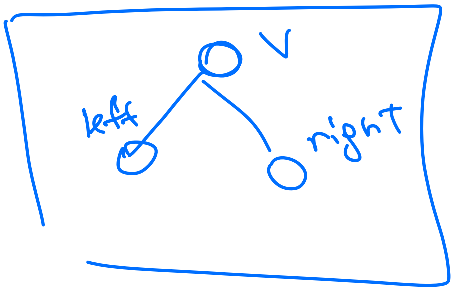
y - куча (пирамида)

x - BST

Для определенных x и y всех точек декартово дерево единственно.

1 2 3





$left.x < V.x \leq right.x$

$v.y > left.y$

$v.y > right.y$

?

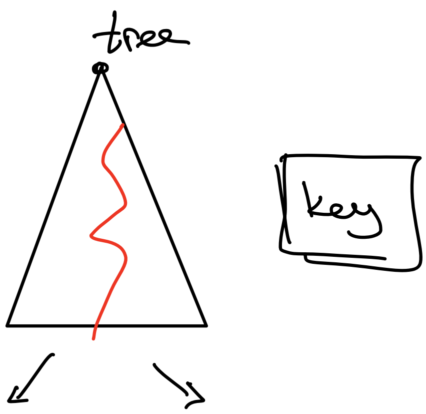
$left.y \vee right.y$

max()
min()
search()
} BST → $\underline{O}(H)$ no x =>

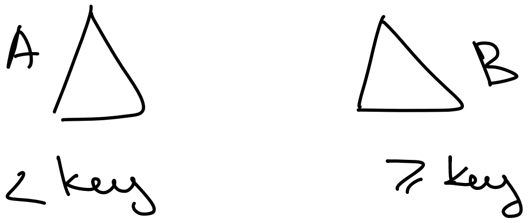
Heap (Max)		BST	
max	1	search	
popmax	logN	push	$\underline{O}(H)$
push		pop	
		min	
		max	

Декартово дерево → $H \sim \log_2 N$

Insert
Remove
→ split
merge



Treap = Tree + Heap

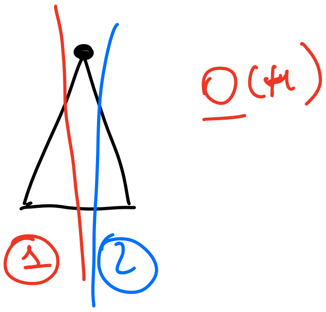


так, чтобы

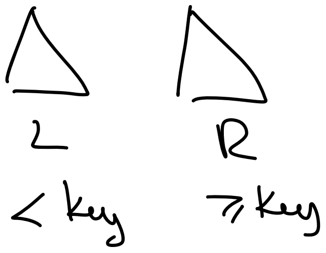
∀a∈A a.x < key

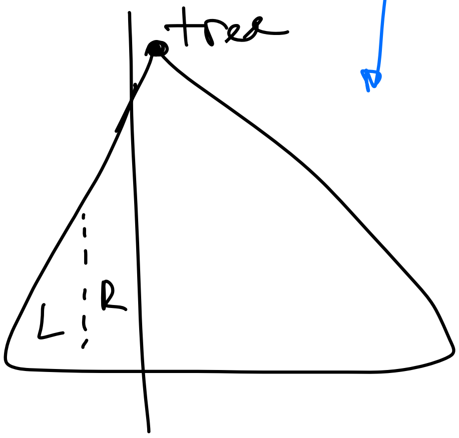
∀b∈B b.x ≥ key.

Tree1, Tree2



```
Split(tree, key) {
    если tree = None
        ↳ return (None, None)
    если key > tree.x
        L, R = split(tree.right, key)
        tree.right = L
        return (tree, R)
    иначе
        L, R = split(tree.left, key)
        tree.left = R
        return (L, tree)
```

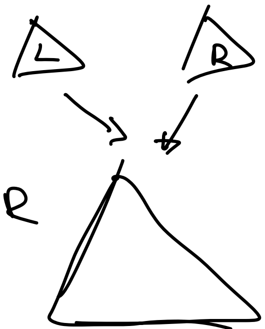


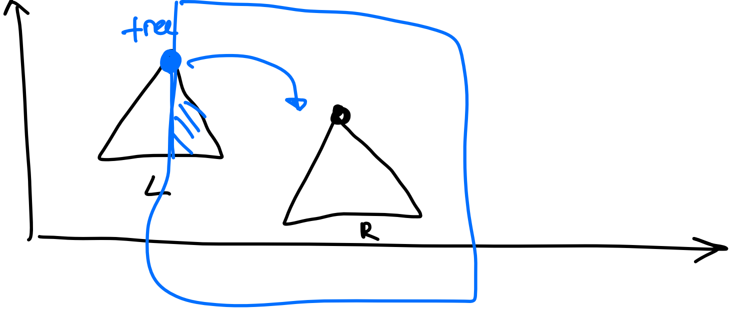


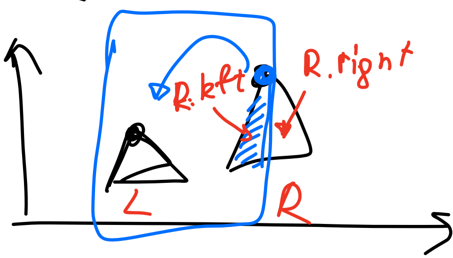
Merge(L, R) => tree

все x из L

меньше x из R







R.left =

Merge (L, R.left)

```
Merge (L, R)
{
    если    L = Null
                ↳ return R
    если    R = Null
                ↳ return L

    если    L.y ≥ R.y:
                L.right = Merge (L.right, R)
                return L
    иначе
                R.left = Merge (L, R.left)
                return R
}
```

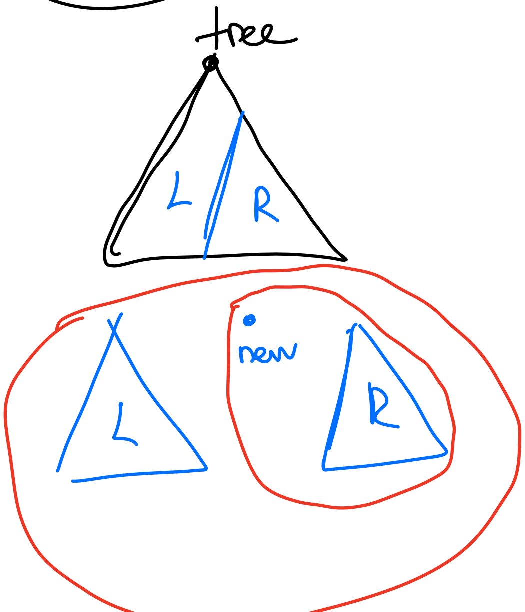
$O(H)$

---

insert

● new

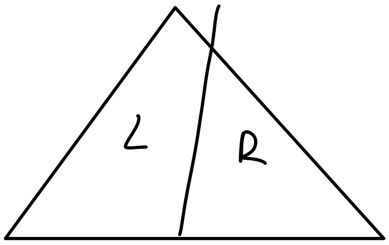
new.x
new.y

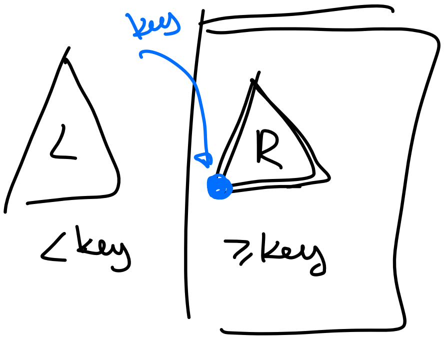


$y = RAN(0, 10^5)$

```
insert (x, y)
    new.x = x
    new.y = y
    L, R = split(x)
    R = merge(new, R)
    root = merge(L, R)
```

remove





```
remove ( key )
    L, R = split ( key )
    R. removeMin ( )
    root = merge ( L, R )
```