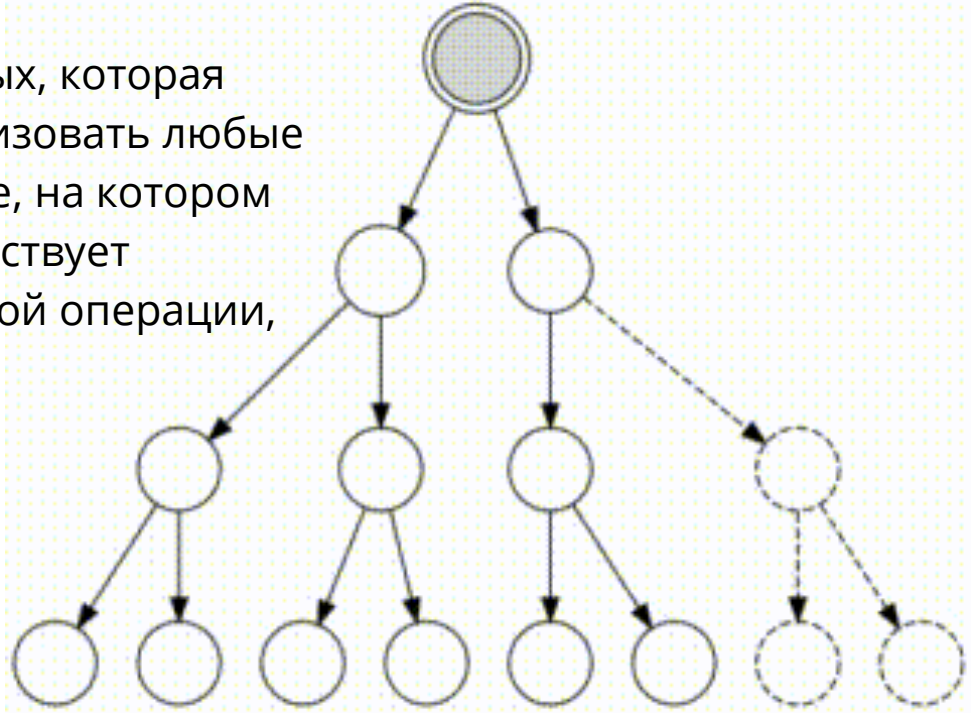
АиСД

Красно-черные деревья
Дерево отрезков
Декартово дерево

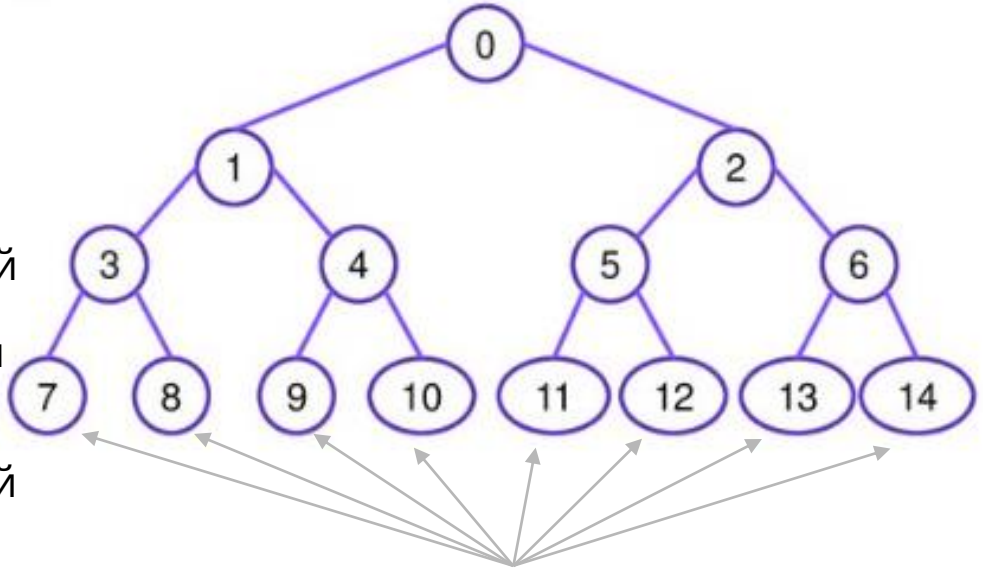
Дерево отрезков

Дерево отрезков - это структура данных, которая позволяет за асимптотику O(logn) реализовать любые операции, определяемые на множестве, на котором данная операция ассоциативна, и существует нейтральный элемент относительно этой операции, то есть на моноиде.



Описание структуры

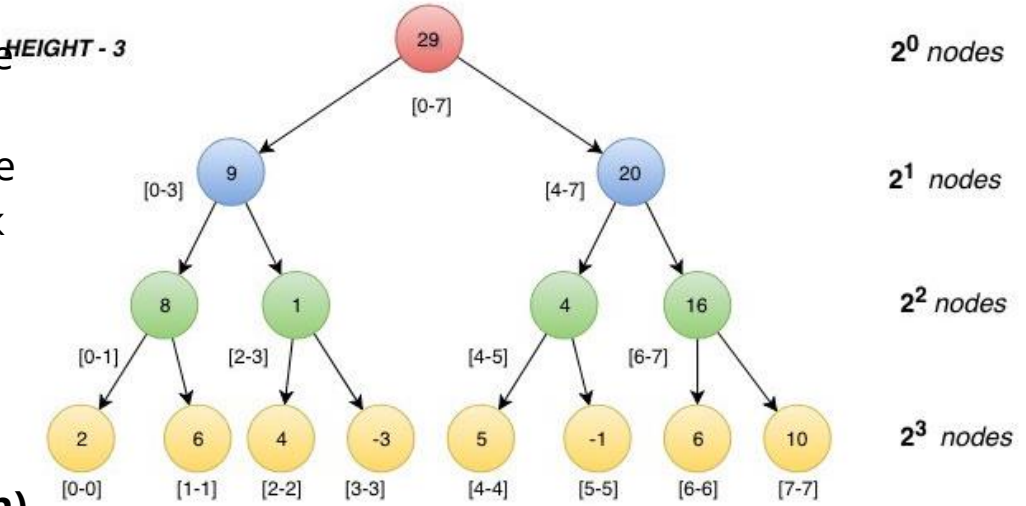
- Дерево отрезков - корневое дерево
- Листья - элементы исходного массива
- Другие вершины - имеют два ребенка
- Каждой вершине в соответствие поставлен интервал, являющийся объединением интервалов ее детей (если у вершины есть дети), либо интервал, содержащий конкретный элемент массива (для листьев).
- В каждой вершины хранится значение некоторой ассоциативной функции **f** на данном интервале



Исходный массив
[7, 8, 9, 10, 11, 12, 13, 14]

Построение дерева отрезков

- Сначала запишем значения элементов a[i] в соответствующие листья дерева.
- Рекурсивно подсчитаем значение вершин предыдущего уровня как сумму значений в двух листьях.



Асимптотика построения дерева отрезков составит, таким образом, **O(n)**.

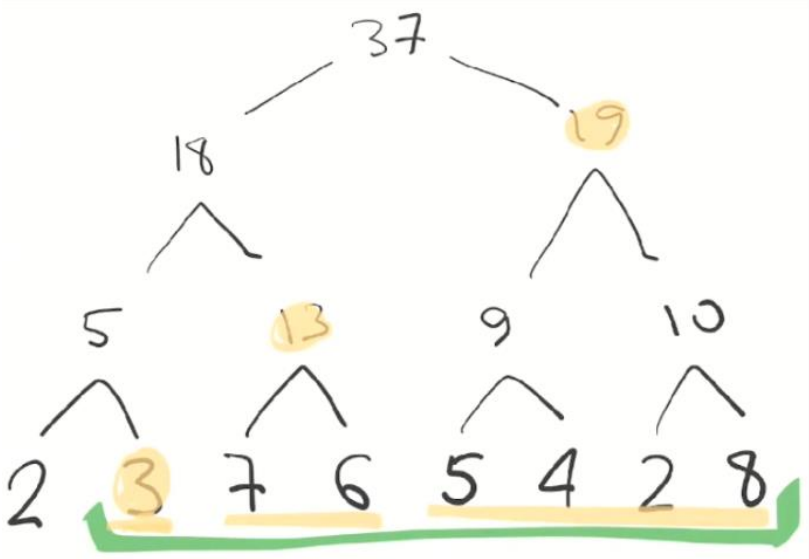
Корень этого дерева соответствует отрезку [0, n)

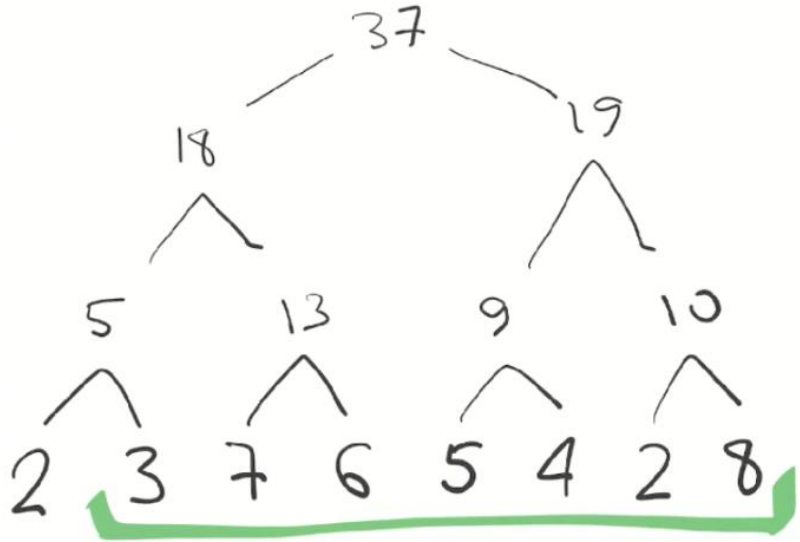
Свойства:

- Высота дерева отрезков равна Θ(logn): на каждом новом уровне длина отрезка уменьшается вдвое.
- Любой полуинтервал разбивается на O(logn) неперекрывающихся полуинтервалов, соответствующих вершинам дерева.
- Дерево содержит менее 2n вершин.
- При n, отличных от степеней двойки, не все уровни дерева отрезков будут полностью заполнены.

Запрос суммы отрезка

- Чтобы найти сумму отрезка, нужно сложить элементы, которые, так или иначе, покрывают наш отрезок:





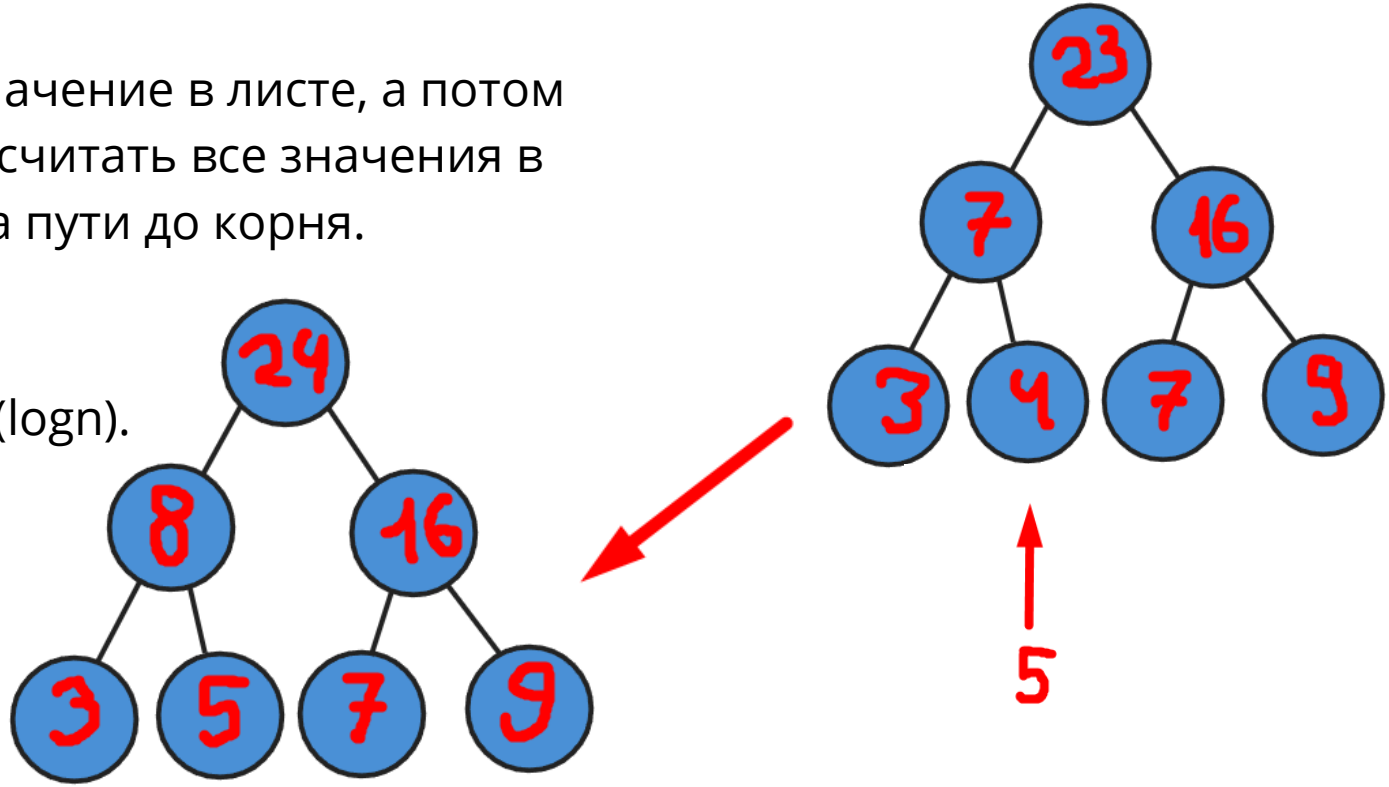
Результат: 3 + 13 + 19 = 35.

Если бы отрезок состоял из тысячи элементов, достаточно было бы сложить в среднем 10 элементов!

Запрос обновления элемента

- Изменить значение в листе, а потом заново пересчитать все значения в вершинах на пути до корня.

Время работы O(logn).



Применение

- С помощью дерева отрезков можно искать сумму чисел, максимум или минимум, но и любую функцию, удовлетворяющую свойству ассоциативности.
- Выполнения массовых операций на многомерных структурах данных.
- Поиск НОД / НОК.
- Поиск префикса массива с заданной суммой.
- Персистентные структуры данных.

Декартово дерево

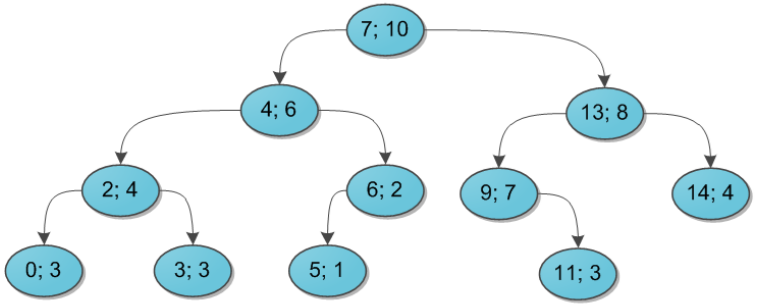
Рене Декарт не является создателем декартова дерева, но он является создателем декартовой системы координат, которую мы все знаем и любим.

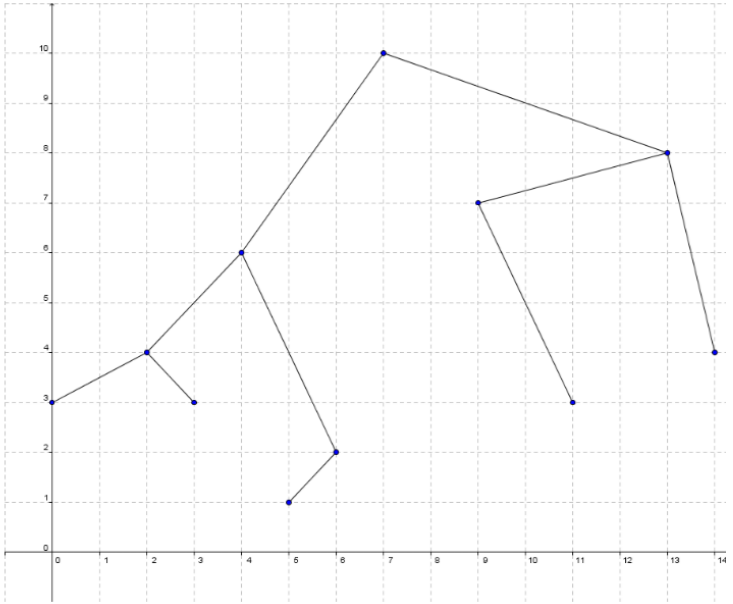
Декартово дерево же определяется и строится так:

- Нанесём на плоскость набор из точек. И **x** зачем-то назовем ***ключом***, а **y** ***приоритетом***.
- Выберем **самую верхнюю точку** *(с наибольшим **y**, а если таких несколько — любую)* и назовём её **корнем**.
- От всех вершин, лежащих слева *(с меньшим **x**)* от корня, рекурсивно запустим этот же процесс. Если слева была хоть одна вершина, то присоединим корень левой части в качестве левого сына текущего корня.
- Аналогично, запустимся от правой части и добавим корню правого сына.

Декартово дерево

Если нарисовать получившуюся структуру на плоскости, то получится действительно дерево — по традиции, корнем вверх:

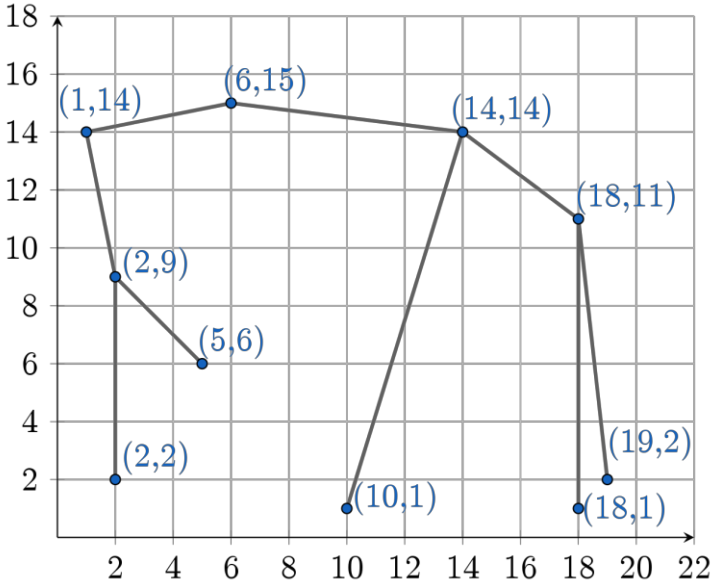




Заметим, что если все y и x различны, то дерево строится однозначно.

Декартово дерево

Таким образом, декартово дерево — это одновременно бинарное дерево по x и куча по y.



В декартовом дереве логарифмическая высота дерева гарантируется не инвариантами и эвристиками, а законами теории вероятностей:

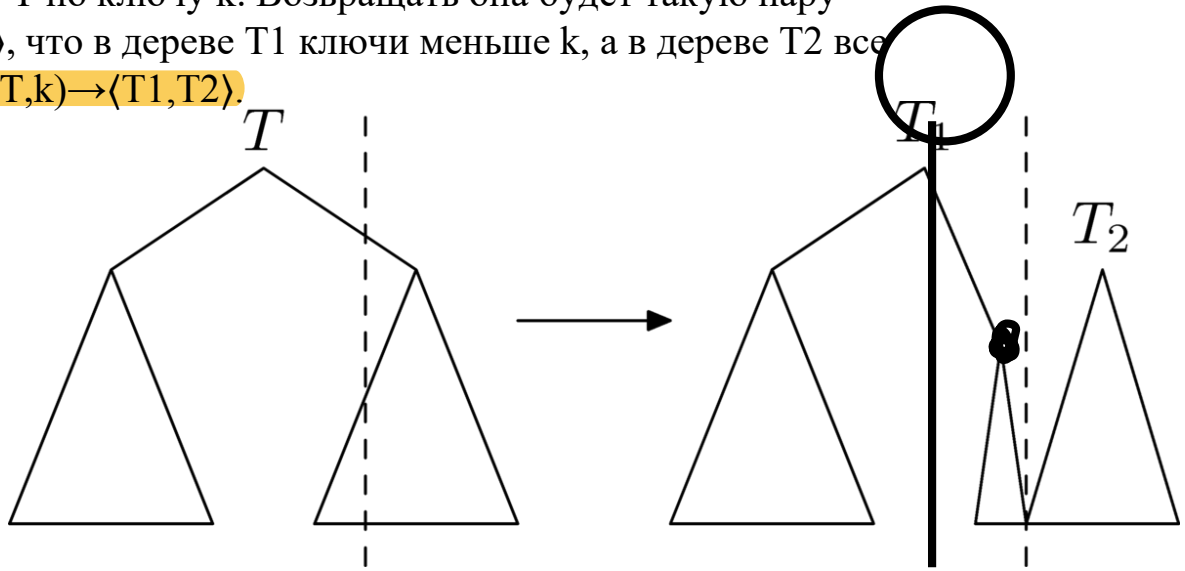
- оказывается, что если все приоритеты (y) выбирать случайно, то средняя глубина вершины будет логарифмической.

Поэтому **ДД** ещё называют **рандомизированным деревом поиска**.

Теорема. Ожидание глубины вершины в декартовом дереве равно $O(\log n)$.

Операции в декартовом дереве

Операция **split** (разрезать) позволяет сделать следующее: разрезать исходное дерево T по ключу k. Возвращать она будет такую пару деревьев ⟨T1,T2⟩, что в дереве T1 ключи меньше k, а в дереве T2 все остальные: split(T,k)→⟨T1,T2⟩.

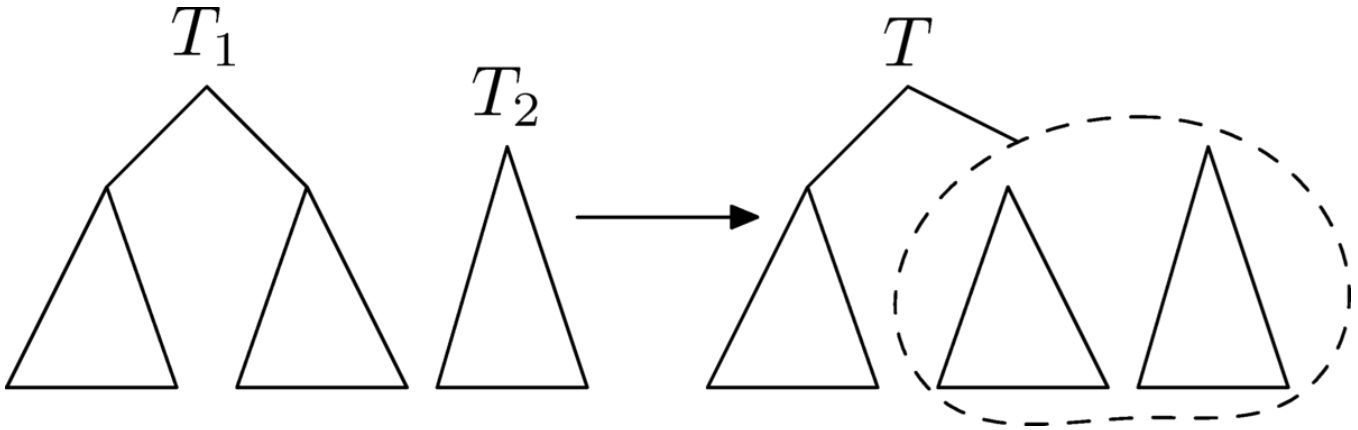


Оценим время работы операции **split**. Во время выполнения вызывается одна операция split для дерева хотя бы на один меньшей высоты и делается еще **O(1)** операций. Тогда итоговая трудоемкость этой операции равна **O(h)**, где **h** — высота дерева.

Операции в декартовом дереве

merge

С помощью этой операции можно слить два декартовых дерева в одно. Все ключи в первом(левом) дереве должны быть меньше, чем ключи во втором(правом). В результате получается дерево, в котором есть все ключи из первого и второго деревьев: merge(T1,T2)→{T}



Операции в декартовом дереве

merge

Пусть нужно слить деревья **T1** и **T2**:
Тогда, очевидно, у результирующего дерева **T** есть корень.

1. Корнем станет вершина из T1 или T2 с наибольшим приоритетом **y**.
    a. Но вершина с самым большим **y** из всех вершин деревьев T1 и T2 может быть только либо корнем T1, либо корнем T2.
    b. Если **y** корня T1 больше **y** корня T2, то он и будет являться корнем.
        i. Тогда левое поддерево T совпадет с левым поддеревом T1.
            1. Справа же нужно подвесить объединение правого поддерева T1 и дерева T2.

Случай, в котором корень T2 имеет больший y, чем корень T1, симметричен этому.

Рассуждая аналогично операции split, приходим к выводу, что трудоёмкость операции **merge** равна **O(h)**, где **h** — высота дерева.

Операции в декартовом дереве

merge и **split** сами по себе не очень полезные, но помогут написать все остальное.

Вот так, например, будет выглядеть код, добавляющий в дерево.

```
Node *root = 0;

void insert (int x) {
    Pair q = split(root, x);
    Node *t = new Node(x);
    root = merge(q.first, merge(t,
q.second));
}
```

Область применимости

Декартово дерево не является самобалансирующимся в обычном смысле, и применяют его по следующим причинам:

- Проще реализуется по сравнению, например, с настоящими самобалансирующимися деревьями вроде красно-чёрного.
- Хорошо ведёт себя «в среднем», если ключи y раздать случайно.
- Типичная для сортирующего дерева операция «разделить по ключу x на „меньше x_0“ и „не меньше x_0“» работает за $O(h)$, где h — высота дерева. На красно-чёрных деревьях придётся восстанавливать балансировку и окраску узлов.

Недостатки декартового дерева:

- Большие накладные расходы на хранение: вместе с каждым элементом хранятся два-три указателя и случайный ключ y.
- Скорость доступа $O(n)$ в худшем, хотя и маловероятном, случае. Поэтому декартово дерево недопустимо, например, в ядрах ОС.