t(n, input) - время работы алгоритмы при входных данных input размера n. Тогда время работы алгоритма $t(n) = max_{input} t(n, input)$.

$t(n) = O(f(n)) \Leftrightarrow \exists c > 0,\ N > 0 : \forall n > N\ t(n) \leq c \cdot f(n)$.

$t(n) = o(f(n)) \Leftrightarrow \forall c > 0, \exists N : \forall n > N\ t(n) \leq c \cdot f(n)$.

$t(n) = \Omega(f(n)) \Leftrightarrow \exists c > 0,\ \forall N,\ \exists n > N,\ input : t(n, input) \geq c \cdot f(n)$.

$t(n) = \omega(f(n)) \Leftrightarrow \forall c > 0,\ \forall N,\ \exists n > N,\ input : t(n, input) \geq c \cdot f(n)$.

$t(n) = \theta(f(n)) \Leftrightarrow t(n) = \Omega(f(n)),\ t(n) = O(f(n))$.

Алгоритм является полиномиальным, если $t(input) = O(|input|^k)$. $|input|$ - битовая длина.

Сильно полиномиальный алгоритм - $t(n) = O(Poly(n))$ - string-poly

Слабо полиномиальный алгоритм - $O(Poly(n, \log C))$ - weak-poly

Псевдо полиномиальный алгоритм - $O(Poly(n, C))$ - pseudo-poly

- unit test (обычные, запускаем просто тесты)
- integration (разные компоненты программы нормально живут вместе)
- prod (тестирование от самого начала до конца)

Вот нам split и merge ДД рассказали.

Insert Генерируем вершину с нашим ключом и случайным приоритетом. Сплитим дерево по x, который хочется вставить. Мёрджим левое поддерево с вершиной, а потом полученное с правым поддеревом.

Erase Сплитим по x + 1, потом левое поддерево по x. Затем мёрджим два крайних дерева.

Обсудим, как удалить элемент, если нельзя инкрементировать, а можно только сравнивать.

1 способ. Пусть удаляемый ключ - A. Найдём следующий ключ в ДД - B. Сплитим по B, потом левое по A. Мёрджим два крайних дерева.

2 способ. Делаем два сплита (первый отправляет равные ключи влево, второй отправляет равные ключи вправо). Тогда, с помощью таких операций можно выразить удаление.

Ещё один способ вставки (на практике быстрый) Спускаемся по бин дереву особо не задумываясь. Идём вниз по бин дереву, пока не нарушаются условия для приоритетов. Когда условия нарушилось, берём это поддерево, сплитим его и ставим нашу вершину на это место, а левыми и правыми сыновьями делаем расспличенныедеревья.

Ещё один способ удаления (аналогичный) Ищем элемент. Мёрджим два поддерева дерева, а потом просто подвешиваем полученное дерево на место исходной вершины.

Подсчёт k-ой порядковую статистику. Для каждой вершины храним размер поддерева. Идём по дереву и смотрим на размер левого поддерева. Если он больше, чем текущий счётчик, то идём в левое поддерево, если равен, то мы уже там где надо, иначе уменьшаем счётчик на размер левого поддерева + 1 и идём вправо. Изначально счётчик равен k.

По неявному ключу. Хотим структуру: массив, к индексам которого можно обращаться за log и вставлять в середину за log.

Переделаем split: Делаем разрезы не по ключу, а по количеству (совметим идеи split и k-ой порядковой статистики). Теперь мы не пользуемся тем, что ключи отсортированны и мы можем ключи использовать как какой-нибудь мусор.

Итерироваться по такому массиву можно за линию (каждый переход по ребру туда и обратно занимает одну операцию).

Вращение в массиве: Вырезаем splitom первые k элементов, а потом меняем два поддерева местами и мёрджим.

Ещё всякие приколы: Умеем находить максимум на отрезке. В вершине будем хранить значение максимума в поддереве. Выспличиваем наш подотрезок и смотрим на значение в корне.

Умеем зеркально отражать подотрезки. Будем в каждой вершине хранить модификаторы и если надо - будем их проталкивать вниз. Когда разворачиваем поддерево вершины - просто говорим, что его модификатор = 1. Теперь, когда хотим проталкивать делаем вот что: убираем флажок в нашей вершине меняем ссылки на левого и правого сына и ксорим с единицей флажок детей. Теперь, вершина - абсолютно нормальная.

Построение декартача за линию, когда x уже отсорчены.

Пойдём слево направо. Когда вставляем новую вершину идём снизу вверх от самой большой вершины дерева к корню и смотрим, в какой момент мы можем вставить нашу вершину. Мы ее вставляем как корень соответствующего поддерева, а потом переподвешиваем её туда, куда нужно. Аммортизированно - линия. Потенциал - длина правого пути (упражнение))). При этом правый путь надо хранить стэком.

interleave (как merge, только ключи первого дерева не обязательно меньше). Тут трэш начался)) Если коротко, то мы огкусываем от первого дерева все элементы, которые меньше минимума во втором, потом отрезаем от второго дерева, ну и продолжаем так, пока оба дерева не пропадут. А потом просто мёрджишь по порядку.