Описание алгоритма CINV

Тривиальная реализация данного алгоритма имела бы сложность $O(n^2)$, а нам надо $O(n \log n)$. Зная алгоритмы сортировки, такие как *Merge Sort*, то логично поступить с этой задачей так же: мы будем делить наш входной массив на две части каждый раз, пока не дойдём до размера 2. В нём подсчитаем инверсии и будем складывать результаты, но также нам необходимо посчитать инверсии между массивами.

Однако тут есть хитрое решение: мы будем одновременно ещё и сортировать данные маленькие подмассивы. Соответственно, если элемент из правой части будет меньше, чем текущий элемент из левой половины, то все оставшиеся элементы из левой части будут давать инверсии, а это легко посчитать формульно, то есть за $O(1)$.

Описание шагов алгоритма

DIVIDE: Мы «делим» массив пополам, что позволяет уменьшать фронт работы в два раза.

CONQUER: Мы работаем с очень маленькими массивами, что позволяет считать инверсии легко в обеих частях, а потом их складывать.

COMBINE: Когда объединяем, мы считаем количество инверсий *между половинками*, так как внутри мы уже считали, а благодаря отсортированности, мы можем это делать формульно.

Рекуррентное соотношение и анализ сложности

Рекуррентное соотношение имеет вид:

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n), \quad \text{так как } O(n) = n \cdot O(1).$$

Посчитаем асимптотическую сложность по мастер-теореме:

$$a = 2 \geq 1, \quad b = 2 > 1, \quad k = 1 > 0.$$

$$\log_b a = \log_2 2 = 1 = k \;\Rightarrow\; T(n) = O(n \log n), \quad (n^k = n^1,\ f(n) = O(n)).$$

Мы добавляем цикл, который будет проходить уже по ОТСОРТИРОВАННЫМ частям, используя два указателя, на левый и на правый, если выполняется условие мы увеличивам счетчик правой части на 1, потом когда условие перестанет выполнятся мы добавим в общую сумму, но при этом мы не будем сбрасывать счетчик для каждого к, а оставим его таким же, потому что в левой часте уже отсортировано все и в правой, а значит если выполнялось для меньших чисел, то и для больших будет выполнятся.

Почему сложность не возрастает при добавлении циклов

Почему у нас при добавлении нового `for` и вложенного в него `while` мы не получаем $O(n^2)$ или что-то большее, чем $O(n)$.

Пусть размер левой половины — $L = \text{mid} - \text{left} + 1$, а правой — $R = \text{right} - \text{mid}$.

Внешний цикл (`for`) делает L итераций.

Внутренний цикл `while` не сбрасывает переменную k назад. Значит, каждый элемент правой половины просматривается максимум один раз.

Следовательно, общее число шагов $k++$ за весь проход левой половины равно R. Таким образом, получаем:

$$O(L + R) \leq O(n).$$