

1. Построим граф на n вершинах и проведём ориентированное ребро из i в j если $a_j = i$. Заметим что из каждой вершины выходит одно ребро и в каждую вершину входит одно ребро. Значит наш граф разбивается на циклы.

Утверждается что сортировка делает количество обменов равное n минус количество циклов в графе.

Посмотрим почему это правда. Каждую итерацию мы берём из какого-то цикла минимальный элемент и меняем его со следующим по порядку в цикле. Сам элемент из цикла выкидывая. Заметим что размер цикла уменьшился на 1. А так как массив становится отсортированным, тогда и только тогда когда в графе все вершины изолированы, получаем что худший случай сортировки - в графе один большой цикл.

Посчитаем количество таких перестановок, что в графе будет один цикл.

Давайте посмотрим на граф. Заметим что по порядку вершин в цикле мы можем построить единственную перестановку, удовлетворяющую этому циклу. Однако каждой хорошей перестановке (худший вариант для сортировки), удовлетворяет n разных циклов. Где каждый - циклический сдвиг другого.

Всего циклов $n!$ - как количество перестановок. Мы поняли что циклические сдвиги нас не интересуют, значит можно зафиксировать один элемент на месте (например единицу поставить на позицию n), тогда получим что остальные элементы можно перемешать $(n - 1)!$ различными способами.

Значит всего перестановок на которых сортировка работает за максимальное число обменов $(n - 1)!$

2. Если рассматривать алгоритм из условия задачи, то он делает максимальное количество свапов в случае когда на первом месте стоит 1. Действительно, на каждой итерации 1 подвинется к своей позиции в конце массива на единицу. Это значит что каждая итерация сортировки нам важна.

Что стоит на остальных позициях нас не волнует, это просто перестановка чисел от 2 от n . А количество таких $(n - 1)!$, что и является ответом на задачу.

3. (a) Вот нам дали перестановку A на которой пузырьрёк работает максимальное число свапов. Нужно показать по ней единственную перестановку, на которой сортировка вставками будет работать за максимальное число свапов

Сделаем B по следующему алгоритму.

$$i = \{2, 3, \dots, n\}, B_{A_i} = A_{i-1}$$

$$i = 1, B_{A_1} = A_n;$$

Как видим, если построить граф по B , то в нем будет один единственный цикл длины n . Значит мы построили нужную перестановку.

- (b) В одну сторону построили, теперь построим в обратную.

Нам дали перестановку A , на которой сортировка вставками работает за максимальное количество свапов. Нам нужно найти пе-

перестановку B такую, что сортировка пузырьком на B работает за максимальное число свапов.

Возьмём и выпишем в новую перестановку B' вершины в порядке обхода одного единственного цикла в соответствующем B графе (мы помним что он единственный). Теперь возьмем циклический сдвиг этой B' такой, что $B'_n = 1$ (помним что можем циклически сдвигать эту перестановку и ничего не поменяется)

После сдвига видим что в конце этой перестановки стоит единичка, а это значит что пузырьёк на ней будет работать за максимальное число свапов.

Получили биекцию.

4. Разобьем исходный массив a на блоки размера k и отсортируем числа в блоках любой сортировкой за $\mathcal{O}(n \log k)$.

Сольем первых два блока за $2k$ операций. Получим массив b из $2k$ элементов.

Заметим что k минимальных элементов из всего массива будут находиться в первых k элементах b . Почему это так?

Эти элементы не могли стоять на позициях больших $2k$ по условию.

А первые $2k$ элементов отсортированы и лежат в массиве b .

Теперь возьмем и запишем во второй блок вторую часть массива b . Заметим что условие на расстояние от элемента до своей позиции сохранилось.

Тогда если первую часть массива b мы запишем в первый блок, и повторим такой merge $\frac{n}{k}$ раз каждый раз сливая два соседних блока и записывая результат обратно, то получим отсортированный массив a . Суммарно merge работает за $\frac{n}{k}k = n$ операций.

Получается сложность $n \log n + n = \mathcal{O}(n \log n)$.

5. Отсортируем массив p с помощью сортировки подсчетом.

Возьмем $\frac{m}{2}$ элемент массива p и найдем $p_{\frac{m}{2}}$ порядковую статистику в a за $\mathcal{O}(n)$

После того как мы нашли нужный элемент, сделаем partition в a относительно найденного x .

Что мы теперь знаем? А знаем мы то что все порядковые статистики $p_i : i < \frac{m}{2}$ будут находиться левее элемента x в a , а все элементы $p_j : j > \frac{m}{2}$ будут находиться правее x в a . Тогда запустимся рекурсивно от левых частей p, a и правых частей p, a и применим такой же алгоритм.

Однако просто такой алгоритм применить нельзя. Нужно для правой ветви рекурсии все соответствующие p_i уменьшать на количество элементов $x' \in a : x' \leq x$

Что бы не делать лишних действий можно передавать offset, изначально равный нулю, как параметр рекурсии и рассматривать все p_i как $p_i - \text{offset}$.

За сколько это работает?

Всего у нас $\mathcal{O}(\log m)$ уровней. На каждом уровне суммарно $\mathcal{O}(n)$ операций, так как отрезки, которым соответствуют вызовы рекурсии не пересекаются.

Тогда алгоритм работает за $\mathcal{O}(n \log m)$

6. Нам дали массив a .
 Построим массив префиксных сумм p :
 $p_0 = a_0$
 $\forall i > 0, p_i = p_{i-1} + a_i$
 Так же пусть $p_{-1} = 0$ для простоты.
 Запустим на нем подобие `merge_sort`
 Пусть функция `solve(p, l, r)` считает ответ для массива p на отрезке от l до r и сортирует этот отрезок.
 Как посчитать ответ для произвольного отрезка?
 Возьмем $m = \lfloor \frac{l+r}{2} \rfloor$
 Вызовем `solve(p, l, m)`, `solve(p, m + 1, r)`
 Добавим в ответ то что вернули эти вызовы. Мы тем самым посчитали все нужные отрезки, границы которых лежат по одну сторону от $m + \frac{1}{2}$
 Теперь найдем отрезки, концы которых по разные стороны от границы.
 Пусть сумма элементов на левой половине равна s Тогда если в левой половине мы взяли границу i , а в правой половине — j , то сумма на таком отрезке равна $(s - p_{i-1}) + (p_j - s) = p_j - p_{i-1}$.
 Нас интересует количество отрезков таких что $p_j - p_{i-1} \geq k$.
 Тогда если мы будем идти по левой половине так что p_i будут убывать, то сумма отрезка при фиксированном j будет возрастать.
 Аналогично если зафиксируем i и будем уменьшать p_j , то сумма будет убывать.
 Отлично! Применим два указателя.
 Поставим j в правой части на минимальный элемент и будем перебирать i по убыванию p_i . Будем искать максимальный j такой что $p_j - p_i < k$. Заметим что j всегда будет двигаться в одну сторону (по возрастанию p_j). Значит делаем так:
 Двигаем на единицу i , двигаем j пока $p_j - p_i \geq k$. Прибавим к ответу j , так как ровно j отрезков подходят для этого i .
 За сколько это работает?
 На каждом слое рекурсии суммарно $\mathcal{O}(n)$ операций. Всего слоёв $\mathcal{O}(\log n)$, следовательно время работы $\mathcal{O}(n \log n)$.