AI Masters Algorithms **Name:** *Денис Грачев*

Homework 4

11 октября 2022 г.

Task 1

Очевидно что отсортированный массив будет выглядеть как $0, 0 \dots 0, 1, 1, \dots 1$, причем количество едениц и нулей останется прежним.

1. Посчитать количество 1. Занимаем $\Theta(n)$, обозначим количество едениц k.
2. Вписать в массив $n - k$ нулей затем k едениц. Занимает $\Theta(n)$

Очевидно что полученный массив соответсвует сортировке изначального массивва.

Task 2

Так как отрезки строго вложенные, то порядок верно следующее $l_i < l_j \Leftrightarrow [l_j, r_j] \subset [l_i, r_i]$, так же $\forall i \neq j : l_i \neq l_j$.

Будем двигать точку x с самого левого края к самому правому. Легко видеть что сначала она пересечет n левых краев, затем n правых краев. Каждый раз при пересечении левого края количество покрываемых ею отрезков увеличивается на 1, каждый раз при пересечении правого отрезка количество покрываемых ею отрезков уменьшается на 1. Отсюда легко видеть что точка будет покрыта ровно $\frac{2n}{3}$ отрезками если она находится между $\frac{2n}{3}$ и $\frac{2n}{3} + 1$ левым или правым отрезком (левые надо считать слева, правые справа. Понятно что это будут края одних и тех же отрезков).

Будем сравнивать отрезки по левому краю. Найдем $\frac{2n}{3}$ и $\frac{2n}{3} + 1$ порядковые статистики, обозначим их как k и m. Это занимает сложность $O(n)$. Искомые точки находятся в отрезках $[l_k, l_m]$ и $[r_m, l_k]$.

Task 3

1

На каждом новом запуске $\mathrm{quicksort}$ длина входа уменьшается хотя бы на 1. Таким образом она не может быть больше n. Длина достигает n на массиве $1, 2, 3, \ldots n$, так как каждый раз последний элемент берется как pivot и $\mathrm{partition}$ возвращает индекс последнего элемента.

2

Каждый раз при запуске будем находить медиану текущего массива и менять ее местами с последним элементом.
Тогда вход будет каждый раз делиться на 2 и глубина будет $\Theta(\log(n))$.

Task 4

Пусть $x_1 \leq x_2 \leq \ldots x_n$.
Наиболее наглядный способ - нарисовать график функции $f(s) = \sum_{i=1}^{2n+1} |x_i - s|$. Функция непрерывная, кусочно линейная. Если $s \leq x_{n+1}$ то коэфициент прямой отрицательный, следовательно функция убывает. Если $s \geq x_{n+1}$ то коэфициент прямой положительный, следовательно функция растет. (Утверждение верно, даже если несколько чисел совпадает). Таким образом $s = x_{n+1}$ минимум.
Найти его можно с помощью поиска $n + 1$ статистики за $O(n)$.

Task 5

Отсортируем входной массив, это занимает $O(n \log(n))$.
Рекурсивно найдем решение для левой и правой половины. (Для массива длины 2 проверка очевидна).

Проверим есть ли такое решение, что одно число в левой половине, а второе в правой. Для этого мысленно преобразуем левую часть следующим образом: каждое число заменим на то, которое в сумме с ним дает искомое $a_i \to x - a_i$ и перевернем ее чтобы она осталась отсортированной. Таким образом у нас есть левая *преобразованная* часть которая состоит из отсортированных чисел которые мы хотим найти в правой (тоже отсортированной) части. Для этого сделаем операцию аналогичную merge, заведем два счетчика для каждой части и будем сравнивать значения по ним. У кого значение меньше, тот счетчик и увеличиваем. Таким образом мы найдем момент когда значение в *обработанной* левой и правой части совпадают, следовательно соответсвующие элементы дают в сумме искомое число. Это занимает $O(n)$.

Преобразование левой части можно провести (это занимает $O(n)$), а можно просто идти в левой части справа налево и при сравнении заменять $a_i \to x - a_i$.

Итого алгоритм

1. Отсортировать массив $O(n \log(n))$
2. Рекурсивная часть
   (a) Разделить массив на две части $O(1)$
   (b) Рекурсивно проверить есть ли пара чисел в левой половине $T(\frac{n}{2})$
   (c) Рекурсивно проверить есть ли пара чисел в правой половине $T(\frac{n}{2})$
   (d) Проверить нет ли такого ответа, что одно число в левой половине второе в правой $O(n)$

Корректность

- Если такая сумма есть и находится в левой половине то мы найдем ее рекурсивно.
- Если такая сумма есть и находится в правой половине то мы найдем ее рекурсивно

- Если такая сумма есть и находится в разных половинах то мы ее найдем с помощью предложенной операции.

$T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n\log(n))$.

Сложность $O(n\log(n)) + T(n) = O(n\log(n))$

Task 6

Будем пока считать что все числа различны. Тогда составим массив P, где на i-ом месте стоит j - позиция элемента в отсортированном массиве.

Тогда отрезок $[i, P_i]$ должен находиться внутри одного непрерывного подмассива, иначе i-ый элемент не сможет попасть на j место, таким образом это необходимое условие.

Так же это и достаточное условие, так как $a_i < a_j \Leftrightarrow P_i < P_j$ (можно сравнивать не элементы а позиции куда они попали после сортировки). Соответсвенно если для всех i внутри непрерывного подмассива, P_i так же лежит в подмассиве, то порядок внутри этого подмассива совпадает с порядком всего массива ограниченным на этот подмассив.

Таким образом необходимо объеденить все отрезки $[i, P_i]$ и понять на какие отрезки они разбивают массив.

Для удобства создадим массив пар $\text{segments} = [(\min(i, P_i), \max(i, P_i))]$ и отсортируем его (сначала сравниваем по первому значению затем по второму). Заведем две переменных $l, r = segments[1]$ и $i = 1$.

Будем обновлять их по следующему принципу

- Если $\text{segments}[i+1][1] < r$, то $r = \max(\text{segments}[i+1][2], r)$
- Иначе, сохрани (l, r) и $l, r = \text{segments}[i+1]$
- $i \mathrel{+}= 1$.

т.е. если левый край следующего отрезка попадает на текущий, то продлить правый край текущего до правого края следующего.

Корректность объединения: так как отрезки отсортированы по левому краю, то если есть $\text{segments}[i+1][1] > r$, значит и все последующие отрезки в текущий не пересекают, так как поддерживался максимум правого края, то все текущие отрезки полностью лежат внутри $[l, r]$. Итоговый алгоритм

- Отсортировать массив и построить P. $O(n \log(n))$
- Построить и отсортировать массив segments. $O(n) + O(n \log(n)) = O(n \log(n))$
- Объединить отрезки и сохранить все (l, r). $O(n)$.

Итоговая сложность $O(n \log(n))$.

Чтобы избежать проблем если числа не различны, необходимо использовать стабильную сортировку. Действительно, рассмотрим a_i, a_j, такие что $i < j,\ a_i = a_j$. Обозначим индексы куда они могут попасть как $P_i < P_j$. Тогда вариант со стабильной сортировкой $\cup[a_i, P_i], [a_j, P_j] \subset \cup[a_i, P_j], [a_j, P_i]$ так как если $a_i < P_j$, то $[a_i, P_i], [a_j, P_j] \subset [a_i, P_j]$ иначе $[a_i, P_i], [a_j, P_j] \subset [a_j, P_i]$.