5.1 Построить наиболее сложный сложный пример для алгоритма быстрой сортировки, который рекурсивно запускается на двух массивах: меньше среднего и больше среднего (не меняя при этом порядок). Причем сложность есть сумма длин массивов, на которых алгоритм будет запущен рекурсивно.

Очевидно, что суммарная длина массивов на каждой итерации не больше, чем N-1, где N – длина массива, который нужно отсортировать (она на каждой итерации разная). Причем случай N-1 будет самым худшим и его легко построить: достаточно сделать так, чтобы, во-первых, опорный элемент был единственным и, во-вторых, был бы минимальным (или максимальным, это неважно). Для простоты будем считать, что опорный элемент – минимальный для всего массива.

Таким образом нужно построить массив, такой, чтобы его средний элемент был минимальным – таким образом суммарная длина массивов будет равна N-1 (один пустой, другой – весь, за исключением опорного); а после каждой итерации в середине снова бы оказывался минимальный элемент.

Рассмотрим вопрос как построить такой массив за линейное время.

Заметим, что если нам дан массив четной длины (2k), то средним будет элемент с номером k. На это место поставим, например, число один. Вспомним, что алгоритм обладает очень важным свойством – он не меняет порядок элементов в массиве, то есть на следующую итерацию поступит точно такой же массив, что и был, только единичка будет «вырезана». Значит, в этом новом массиве (длиной N-1), найдем середину и поставим туда число «2». Причем новая длина (2k-1) будет уже нечетной, но номер середины останется тем же: k, но уже в новом массиве, длиной на единицу меньше. Именно поэтому «проекция» этого места на изначальный массив будет с номером k+1, то есть справа от только что поставленной единицы. Далее массив опять сократится, элементы пойдут в том же порядке и индекс новой середины будет k-1: а это в точности место слева от единицы в изначальном массиве. Следующие число опять захочет встать на место k-1. Можно чуть сменить стиль рассуждений и смотреть на массив не сокращающийся, а получающийся вычеркиванием тех чисел, которые мы написали: как только число хочет встать на место, которое уже занято, указатель сдвигается на ближайшее справа свободное место.

Таким образом получается, что если нам пришел массив четной длины, то сокращая его до нечетной, мы ставим элемент на правую «границу» уже написанных элементов и наоборот: если массив был нечетной длины, то на левую. Получается, что заполнениие идет с чередованием: сначала в середину, потом направо (если длина была четной, и налево, если бы нечетной), потом налево, потом направо – чередуя направления.

Ставится очевидным линейный алгоритм заполнения массива: заведем два указателя, которые изначально указывают на середину. Заполним средний элемент. Затем в зависимости от того, какая получилась длина массива (точнее длина «свободной» части), двинем соответствующий указатель (правый – направо, левый – налево) и поставим туда число, на единицу бОльшее, чем предыдущее.

5.2(а) Поиск пика в унимодальном массиве.

Заметим, что выбрав любой элемент, мы за O(1) можем определить, возрастает или убывает массив на отрезке с этим элементом: для этого посмотрим на предыдущий (или следующий, если такого нет). Таким образом, можно применить аналог бинарного поиска: разделим массив пополам и выберем нужный. Сложность та же: O(lgN), т.к. $T(N) = T(N/2) + O(1)$

Выберем средний элемент массива. Если он больше, чем предыдущий, то значит пик находится где-то в правом отрезке (так как массив сначала возрастает, потом убывает), иначе – слева. Причем отрезки в обоих случаях всегда включают опорный средний элемент. Запускаем процедуру поиска рекурсивно на интересующем нас массиве.

Алгоритм можно завершить, когда длина массива будет не больше двух: просто сравниваем элементы и возвращаем наибольший.

5.2(б) Найти AABB (axis aligned bounding box) строго выпуклого многоугольника за логарифм.

Достаточно очевидно, что координата левого нижнего угла искомого прямоугольника есть минимальная абсцисса всех точек и минимальная ордината. Аналогично, правый верхний угол прямоугольника есть максимальная абсцисса и ордината.

Подробно опишем алгоритм нахождения максимального икс, для остальных случаев все аналогично.

Так как по условию нам дан строго выпуклый многоугольник, то, значит, в нем нет внутренних углов равных 180 градусов. Поэтому, в нем не может быть более двух точек с одинаковыми абсциссами (пар может быть несколько, но вот больше двух точек с абсциссой, например, x, быть не может). Иначе, либо в нем будет внутренний угол в 180°, либо это невыпуклый многоугольник.

Запишем все иксы всех точек в порядке обхода. Причем после последней вершины запишем еще раз первую.

Понятно, что у нас может и не получиться унимодальный массив: почти всегда это будет «бимодальный»: сначала он либо возрастает, либо убывает; затем меняет характер; затем снова меняет характер.

Иногда может быть и унимодальный: когда первой выбрана самая левая, например, точка: тогда он вначале возрастает (до пика – до самой правой точки), затем убывает.

По предыдущему утверждению, в этом массиве не может быть больше двух подряд идущих одинаковых элементов, поэтому можно считать его «почти различным»: сравнивать элемент будем не только с предыдущим, но и с пред-предыдущим в том случае, если этот элемент оказался равен предыдущему.

Именно поэтому можно применить тот же метод, что и в пункте (a): нужно научиться за O(1) определять, где находится пик для любого заранее выбранного элемента.

Для начала заметим, что у нас вообще говоря может быть 4 разных поведений массива: два унимодальных (возрастание-убывание) и два бимодальных (возрастание-убывание-возрастание и убывание-возрастание-убывание). За O(1) мы можем отсеять два из них: просто посмотрим на вторую вершину в порядке обхода. Пусть так оказалось, что у нас массив первого типа: то есть сначала возрастание, потом убывание (потом, может быть, снова возрастание в случае не унимодального массива).

Далее все аналогично пункту (a): выберем средний элемент (i) и посмотрим на поведение отрезка. Если там убывание – все хорошо, пик уже был и у нас на отрезке [1..i] точно унимодальный массив и свели задачу к пункту a.

Если же у нас там возрастание, то посмотрим на значение элемента i: если он больше начала (исходного массива), то мы в «первом» возрастании и пик только будет и смотрим на правый отрезок, если меньше – то мы во «втором возрастании» и пик уже был, смотрим на левый отрезок (заметим, что в случае унимодального массива всегда будет первый случай). Далее запускаем процедуру рекурсивно.

Сложность одного поиска, очевидно, как и в пункте (a) – O(lgN). Всего можно сделать 4 прохода (по два на каждую координату), поэтому общая сложность тоже O(lgN).

5.3 На прямой расположено n точек. У каждой точки есть вес. Найти такую точку, чтобы чтобы сумма произведений каждого веса точки на расстояние от точки до искомой была минимальна.

Будем предполагать, что наши точки отсортированы по координате x, если это не так – отсортируем их.

Предположим, что мы нашли такую точку q, которая минимизирует функцию. И пусть такая точка лежит в интервале между x[i] и x[i+1]. Посчитаем сумму весов слева от этой точки и справа. Пусть слева эта сумма равна L, а справа – R. Тогда искомая функция равна:

$$S_{min} = (q - x_1) * w_1 + \ \ldots + (q - x_i) * w_i + (x_{i+1} - q) * w_{i+1} + \ldots + (x_n - q) * w_n$$

Предположим, мы сдвигаем нашу точку влево на небольшое расстояние d, при этом точка не выходит за пределы интервала, в котором она расположена. Тогда новая функция равна

$$S = (q - d - x_1) * w_1 + \ \ldots + (q - d - x_i) * w_i + (x_{i+1} - q + d) * w_{i+1} + \ldots + (x_n - q + d) * w_n$$

Или:

$$S = S_{min} - d * (L - R)$$

Таким образом, если сумма чисел слева от точки будет больше, чем сумма справа, то получим противоречие.

Аналогично рассуждая для случая, если L<R (при этом мы двигаем нашу точку чуть правее, снова не выходя за границы интервала) приходим к выводу, что для того чтобы наша точка q была искомой на интервале, необходимо и достаточно, чтобы сумма весов слева была равна сумме весов справа.

Если мы попробуем двинуть точку на границу интервала, мы получим (рассуждения абсолютно аналогичные – чисто алгебраические действия, за небольшим исключением: при движении точки на границу интервала одно слагаемое зануляется), что она является решением и на границах отрезка: значение функции там равно значению внутри интервала.

Отсюда вывод: если нам не нужно искать **все** те точки, где достигается минимум, то достаточно лишь просмотреть те варианты, где искомая точка **совпадает** с какой-либо из данных точек (то есть фактически просмотреть лишь границы отрезков) и выбрать наименьший.

Это можно сделать за линию: для этого надо научиться за O(1) пересчитывать суммы слева от указателя и справа от него.

Пусть у нас уже была посчитана сумма для l первых точек (то есть мы стоим на l+1-ой). Она равна

$$S_l = (q - x_1) * w_1 + (q - x_2) * w_2 + \ldots + (q - x_l) * w_l$$

Подвинем указатель вправо (пусть расстояние до следующей точки равно d), посмотрим как поменяется эта сумма:

$$S_{l+1} = (q + d - x_1) * w_1 + (q + d - x_2) * w_2 + \ldots + (q + d - x_l) * w_l + d * w_{l+1}$$

$$S_{l+1} = S_l + d * \sum_{j=1}^{l+1} w_j$$

Выходит, новая левая сумма равна старой, плюс расстояние умножить частичную сумму весов длины l+1. Сделав за линию предподсчет частичных сумм весов, сможем за O(1) поддерживать левую сумму в актуальном состоянии.

Для правой суммы аналогично, за исключением того, что из нее придется вычитать и, соответственно, придется предпосчитать частичные суммы «справа». Но все равно за линию.

Пробежимся по массиву – найдем минимальную сумму.

Сложность: O(NlgN) на сортировку, O(N) на поиск точки минимума.

5.4 Про почти отсортированный массив.

1. На полностью отсортированном массиве алгоритм выдаст True всегда: это очевидно следует из корректности алгоритма бинарного поиска.

2. Пусть нам дан не почти-отсортированный массив.

Назовем элементы в массиве «плохими», если они не находятся через наш бинарный поиск; и соответственно «хорошими» - если находятся. (Таким образом полностью отсортированный массив состоит из «хороших» элементов).

Утв1. Рассмотрим бинарный поиск в пусть даже несортированном массиве (он есть в задании). Пусть мы ищем элемент x и нам бинпоиск дал индекс i, а если ищем элемент y, то бинпоиск даст индекс j. Тогда, если i<j, то x<=y.

Пусть для ключа x все те элементы a[mid] с которыми он сравнивается есть x1, x2,..., xK (их O(lgN)), а для ключа y – они y1, y2,..., yT. Причем достаточно очевидно, что x1=y1. И пусть номер l - тот наименьший номер, где эти последовательности различаются: xl != yl.

Тогда можно заключить, что (вспоминая, что есть предположение о том, что их позиции различны и i<j):

$$x < x_{l-1} = y_{l-1} \leq y$$

Если нарисовать дерево сравнений, то ставится чуть более понятно: элемент y не может пойти по правой ветке, когда элемент x пошел по левой, иначе не получится, что i<j.

Утв2. Если массив не почти-отсортированный, то в нем как минимум 10% элементов – плохие.

Предположим обратное: меньше 10% плохих элементов, значит как минимум 90% - хорошие. Удалим из массива все плохие элементы. Рассмотрим любые два хороших элемента: x и y. Пусть x находится на месте i, а y находится на месте j. Если i<j, то x<=y, а если j<i, то y<=x (это утв.1). Получается, что любая пара находится в отсоритрованном порядке. Противоречение с «не почти-отсортированным массивом».

Утв3. Если массив не почти-отсортированный, то алгоритм выдаст False с вероятностью не меньше 2/3 при выборе соответствующего k.

Из второго утверждения следует, что в таком массиве как минимум 10% элементов – плохие (те, что выдадут False). Найдем вероятность того, что, сделав k выборок, мы хоть раз наткнемся на «плохой» элемент. Это просто: это есть единица минус вероятность того, что ни разу за k выборок не накнемся на «плохой» элемент. То есть искомая вероятность есть:

$$P = 1 - \left(\frac{9}{10}\right)^k \geq \frac{2}{3}$$

Логарифмируя, получаем, что если $k > \frac{lg\left(\frac{1}{3}\right)}{\lg\left(\frac{9}{10}\right)} \approx 10.427$, то алгоритм выдаст False с вероятностью не меньше 2/3.

5.5 Доказать, что матожидание времени работы вероятностного алгоритма поиска k-ой порядковой статистики в массиве A[1..n] есть O(n).

Алгоритм работает так:

- Выберем случайный элемент из массива – он будет опорным в следующем шаге
- Сделаем разбиение аналогичное тому, что делаем в стандартном Qsort'e: слева от опорного элементы не превышающие его, справа – большие его. *Это делается за линию.*
- В зависимости от количества элементов в каждом массиве запускаем рекурсивно на нужном подмассиве.

Так как каждый опорный элемент выбирается независимо и равномерно, то для любого k с вероятностью 1/n в массиве элементов не превышающих опорного ровно k чисел.

Так как мы не знаем, на каком массиве будет вызван рекурсивно наш алгоритм, можно ограничить время T(N) работы алгоритма сверху временем на рекурсивный вызов на бОльшем подмассиве: его длина k-1 (A[1..k-1]), а длина меньшего: n-k (A[q+1..n]). Это моделирует ситауцию, когда искомый элемент находится постоянно в бОльшем подмассиве.

Таким образом можно записать, что

$$T(n) \leq \sum_{k=1}^{n} X_k \cdot T(\max(k-1, n-k)) + O(n))$$

Где X_k есть индикаторная случайная величина, которая равна 1 только в одном случае: когда в левом подмассиве ровно k элементов и равна 0 когда это не так. Причем матожидание этой величины равно 1/n. (по достаточно простому свойству матожидания индикаторной величины – оно равно вероятности того события, при котором случайная величина становится равной единице).

Пользуясь линейностью матожидания и независимостью между количеством элементов в подмассиве и временем работы алгоритма (т.к. неизвестно, в какой подмассив мы в итоге попадем) получаем:

$$E(T(n)) \leq \sum_{k=1}^{n} \frac{1}{n} \cdot E(T(\max(k-1, n-k))) + O(n))$$

Заметим, что мы считаем сумму по всем k и смотрим время работы от максимума между

же нечетное, то тоже каждое, за исключением среднего: T(floor(n/2)), оно добавляется один раз. Поэтому маожидание можно оценить так:

$$E(T(n)) \leq \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} \frac{2}{n} \cdot E(T(k)) + O(n))$$

Дальше действуем аналогично тому, когда решали рекурретные соотношения: убедимся, что E(T(n)) = O(n): предположим что E(T(n)) <= cn для некоторой константы c.

$$E\big(T(n)\big) \leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} ck + an = \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor - 1} k\right) + an \leq \frac{2c}{n}\left(\frac{n^2 - n}{n} - \frac{\frac{n^2}{4} - \frac{3n}{2} + 2}{n}\right) + an$$

$$= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an \leq \frac{3cn}{4} + \frac{c}{2} + an = cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right) \leq cn$$

Нужно показать, что при достаточно большИх с скобка будет положительной. Или:

$$n\left(\frac{c}{4} - a\right) \geq \frac{c}{2}$$

Если выберем с такое, что скобка будет положительной, то можем на нее поделить и получим:

$$n \geq \frac{2c}{\mathrm{c} - 4a}$$

Если предположить, что для всех n, меньших $2c/(\mathrm{c} - 4a)$ время работы алгоритма O(1), то получим, что оценка на матожидание есть O(n).