Студент: Маргарита Чудова
Группа: 2
Дата: 27 апреля 2022 г.

---

1. Каждую гайку и болт положим в массив как пару чисел (x_i, z), где x_i — номер пары гайка-болт (различные у всех пар, одинаковые у соответствующих гайки и болта), а z равен 0, если элемент болт, и 1, если элемент является гайкой.
   Сначала отсортируем массив по ключу z — все болты в левую часть, все гайки в правую. Мы научились это делать на семинаре за $\mathcal{O}(2n) = \mathcal{O}(n)$, когда решали задачу 2: заводим указатели l и r, двиггаем их до тех пор, пока не нашли 0 и 1 не на своем месте, меняем их местами. Псевдокод:

   ```
   l, r = 0, n - 1
   while l < r:
      while l < r and a[i] == 0:
          l +=1
      while r > -1 and a[i] == 1:
          r -= 1
      if l < r:
          swap(a[i] = a[r])
   ```

   Получили как бы два массива — масиив гаек и массив болтов — внутри одного. Сделаем некий аналог Inplace Qsort по первому элементу пар: будем брать опорный элемент среди болтов, а сортировать Qsort-ом гайки в правой части массива.
   То есть мы берем некий болт, сортируем гайки так, что все гайки с меньшим номером лежат левее гайки с номером, соответствующим болту, с большим - правее, а опорный болт среди болтов ставим на место, соответсвующее гайке с тем же номером: если у гайки место $n+j$, то у болта будет j. Сложность такой перестановки $\mathcal{O}(n \log n)$ в среднем. После всех преобразований элементы $a[i], a[n+i]$ будут образовывать пары гайка-болт, результирующая сложность $\mathcal{O}(n \log n + n) = \mathcal{O}(n \log n)$.

3. Назовем $a = [a_1, \ldots, a_n]$ массив из n чисел, $p = [p_1, \ldots, p_n]$ массив порядковых характеристик. Отсортировать массив p можно за $\mathcal{O}(n+m)$ с помощью сортировки подсчетом. Возьмем медиану в этом массиве — $p_{m/2}$.
   Мы умеем искать k-ю порядковую характеристику с помощью задачи 4 с семинара за $\mathcal{O}(n)$, при этом массив в конечном итоге будет выглядеть так: $x = [x_{i_1} \ldots x_{i_j} \ldots x_{i_n}]$, где x_{i_j} — $p_{m/2}$-я порядковая статистика, слева элементы меньше нее, справа — больше.
   Повторим процедуру для пар массивов $[x_{i_1} \ldots x_{i_j}]$ и $[p_1 \ldots p_{m/2}]$, $[x_{i_j} \ldots x_{i_n}]$ и $[p_{m/2+1} \ldots p_m]$: ищем медианную порядковую характеристику алгоритмом типа Qsort, разбиваем то, что в итоге получилось, на два массива, и так пока не найдутся все порядковые характеристики. Всего разбиений будет $\mathcal{O}(\log m)$, значит результирующая сложность $\mathcal{O}(n \log m + m)$.

4. (a) По позиции в отсортированном массиве: понятно, что индексы слева от медианы и справа от медианы равноправны при такой метрике (кстати, вопрос: если у нас нечетное k, то важно ли, с какой стороны мы берем сколько элементов? просто по этой метрике элемент $i_m - j$ ничем не отличается от $i_m + j$). Тогда можем взять $\frac{k}{2}$ элементов с одной и с другой стороны от медианы.

   Умеем за $\mathcal{O}(n)$ находить k-ю порядковую характеристику в массиве, так что найдем за это время медиану m. Элементы после всех преобразований разобьются следующим образом:

первая часть до медианы, в ней все элементы меньше m, во второй части элемент равный m, в третьей — большие. Пусть длина первой части равна l, а второй r. В первой части мы найдем $l-\frac{k}{2}$-ю порядковую статистику, а в третьей — $\frac{k}{2}$-ю. В них элементы распределятся аналогично на три части, и, поскольку мы рассматриваем индексы отсортированного массива, в первой части в качестве ответа нужно взять все, что лежит после $l-\frac{k}{2}$-ой порядковой статистики, а в третьей — все, что до $\frac{k}{2}$-ой.

(b) По значению: мы умеем за $\mathcal{O}(n)$ находить k-ю порядковую характеристику в массиве. Найдем медиану за это время, пусть ее значение равно m. Преобразуем элементы массива x в элементы массива y следующим образом:

$$x[i] \to y[i] = (|x[i] - m|, x[i])$$

Поскольку модуль функция необратимая, будем хранить в y исходные значения для восстановления. Теперь за $\mathcal{O}(n)$ найдем k-ю порядковую статистику (пусть равна j) среди первых значений элементов y. Тогда получим, что массив y разобъется на 3 части: первая часть до k-ой порядковой статистики, в ней все элементы меньше j, во второй части все элементы равные j (то есть один элемент, по условию все они различны), в третьей — большие. Ответом будут все вторые значения пар $y[i]$, попавшие в первую и вторую части.

6. (a) На $n+1$ дополнительной памяти заведем макс-кучу H, и положим в нее первые $n+1$ элементов массива a. Пройдемся по оставшимся $n-2$ элементам a: берем элемент $a[i], i = n+2 \dots 2n-1$, сравниваем его с $H[0]$ (то есть с максимумом в куче), если $a[i] < H[0]$, то извлекаем максимум из кучи и добаваляем в нее $a[i]$ (для всех элементов это займет у нас $\mathcal{O}(n \log n)$ времени).
Таким образом, мы получили макс-кучу с $n+1$ минимальными элементами массива. Понятно, что медиана это либо $H[1]$, либо $H[2]$, сравним эти элементы, ответом будет больший из них.

P.S: я не очень поняла, зачем были нужны прибавленные/вычтенные единички в условии. Стоило ли рассмотреть задачу аккуратнее?

(b) Пусть в качестве дополнительной памяти у нас есть массив на $2k$ элементов. Сначала положим в него $2k$ элементов массива, найдем медиану за $\mathcal{O}(2k)$, и удалим из массива все, что больше нее, осталось k элементов. Положим новые k элементов из исходного массива на место удаленных, повторим процедуру, и так до тех пор, пока элементы в массиве не закончатся. В итоге останется только k элементов, меньших или равных k-ой порядковой статистике, а значит k минимумов. Всего итераций будет $\frac{n}{k}$, каждая итерация занимает не более $\mathcal{O}(k)$ времени.

7. (a) Каждое число должно участвовать в сравнении хотя бы один раз, иначе мы не будем иметь о нем совсем никакой информации и не сможем понять, является ли оно минимальным. Допустим, что каждое число участвует только в одном сравнении (такого не будет, но для доказательства этого пункта этого предположения достаточно). В каждом сравнении участвуют два числа, то есть общее количество сравнений в таком случае равно $n/2$, меньше него быть не может.

(b) Зайду немного с другой стороны: понятно, что для того, чтобы найти максимальное значение, каждое число в массиве придется сравнить хоть с чем-то, иначе мы не будем иметь вообще никакой информации о нем.
Логично, что наименьшее количество сравнений получится, если сравнивать каждое следующее число с кандидатом на максимум, полученным в предыдущих сравнениях: это автоматически отсечет все элементы, которые заведомо меньше, при этом остальные мы еще не успели потрогать, значит о них нет информации, и каким бы то ни было образом отсечь эти элементы не получится. Даже если мы начнем сравнивать элементы группами, внутри

отдельной группы такая стратегия будет наилучшей, а в конечном итоге максимумы, полученные внитри отдельных групп, придется сравнить, что по сути будет той же процедурой с немного другим порядком.
В первом сравнении мы обработаем 2 элемента, а каждое следующее будет дополнительно задействовать еще один элемент. Пройтись таким образом необходимо по всем элементам массива, поэтому всего сравнений будет $n-1$.

А вообще сравнения можно представить в виде графа — каждая вершина это число в массиве, а ребро — сравнение. Граф точно должен быть связным, а то мы ничего не будем знать про изолированные вершины или их системы. Такой минимальный граф — дерево, у него $n-1$ ребро.

8. (a) Будем сравнивать элементы в парах — на это уйдет n сравнений. В каждой паре будет больший и меньший элемент (числа различны), все большие отправим в один массив, а все меньшие — в другой. Размеры массивов будут n, в массиве меньших элементов будем искать минимум, а в массиве больших — максимум. И то, и другое по предыдущей задаче можно сделать за $n-1$ сравнение, то есть всего $3n-2$ сравнений.