Эффективные сортировка

Сортировка слиянием

- "Разделяй и властвуй"
- Делим массив пополам на две части: левую и правую
- Допустим, мы их как-то отсортировали
- Можем объединить за $O(n)$: идём и выбираем минимальный элемент из двух массивов
- Применяем эту идею рекурсивно, пока размер массива больше 1
- На каждом "уровне" делаем $O(n)$ операций
- Уровней рекурсии $\lceil \log_2 n \rceil$, потому что каждый раз делим пополам
- Итого $O(n \log n)$

Почему не пишем основание логарифма в асимптотике:

$\log_a n = \log_a b \cdot \log_b n$, при этом $\log_a b$ это какая-то константа

Сравнениями лучше нельзя

- Пусть, $f(n)$ - сколько сравнений надо сделать, чтобы отсортировать массив длины n

$$2^{f(n)} \geq n!$$
$$f(n) \geq \lceil \log_2(n!) \rceil$$
$$f(n) \geq \log_2(n \cdot (n-1) \ldots 2 \cdot 1) \geq \log_2(n \cdot (n-1) \ldots \cdot \frac{n}{2}) \geq \log_2((\frac{n}{2})^{n/2}) \geq \frac{n}{2} \log_2 \frac{n}{2} = \Omega(n \log n)$$

Ω это как O, только ограничивает снизу.

Сортировка подсчетом

- Пусть, все числа в массиве a от 1 до C, где C не очень большое.
- Создадим массив cnt_i размера C: сколько чисел в массиве равны i.
- Пройдем по cnt и перезапишем a

- Асимптотика $O(n + C)$
- Можно использовать, когда $max - min$ мало.

Бинарный поиск

- Есть массив чисел, надо проверить, есть ли там какой-то элемент, равный x
- В общем случае $O(n)$ на один такой запрос
- Если элементы посортированы можем лучше

Бинарный поиск

- Пусть, элементы упорядочены по неубыванию (как возрастание, но бывают одинаковые)
- Идея: посмотрим на центральный элемент, сравним с искомым
- Если он равен, то мы его нашли
- Если он больше искомого, имеет смысл рассматривать только левую половину
- Иначе только правую
- Каждый раз сужаем область поиска в два раза
- $O(\log n)$ на один запрос

Тонкости реализации

- Хотим: нерекурсивно, не путаться с ± 1, будет легко обобщить на следующую тему
- Пусть, $f(i)$ - функция, которая говорит, правда ли что $a_i \geq x$ (возвращает 0/1)
- Тогда f выглядит как 000...000111...111
- Выберем l, r, такие что гарантированно $f(l) = 0, f(r) = 1$. Представим, что слева от массива идут $-\infty$, а справа $+\infty$. Тогда $l = -1, r = n$.
  ```
  while r - l > 1:
    m = (l + r) // 2
    if f(m): r = m
    else: l = m
  ```
- После выполнения r - первый подходящий, l - последний неподходящий
- Заметим, что никогда не вызовемся от некорректных значений

Бинарный поиск по ответу

- Задача: даны n палок длин a_i, мы умеем их ломать
- Хотим получить k палок одинаковой целочисленной длины, требуется максимимзировать эту длину
- Хочется просто перебрать все значения длины и для каждого проверить, является ли оно решением: $O(n \cdot \max a)$
- Идея: если можем получить длину x, то можем и длину $x-1$.
- $f(x)$ - можем ли мы получить k палок длины x, можем проверить за $O(n)$.
- f выглядит как 111...111000...000 $\Rightarrow$ можем сделать бинпоиск по x по аналогии.
- $O(n \cdot \log \max a)$

Вещественный бинпоиск

- Дана возрастающая функция f, хотим найти $x : f(x) = 0$
- Делаем бинпоиск по x
- Хотим знать x с какой-то точностью
- **Нельзя писать `while r - l > eps` : проблемы с погрешностью!**
- Вычисляем количество итераций, когда бинпоиск сойдется до нужной точности.

Тернарный поиск

- Дана выпуклая функция f, хотим найти ее минимум.
- Берем $m_1 = \frac{2l+r}{3}, m_2 = \frac{l+2r}{3}$
- Вычисляем $f(m_1)$ и $f(m_2)$
- Если $f(m_1) > f(m_2)$, то минимум не может быть между l и $m_1 \Rightarrow$ переходим к отрезку $[m_1; r]$
- Иначе минимум не может быть между m_2 и $r \Rightarrow$ переходим к отрезку $[l; m_2]$
- Снова выбираем число операций, чтобы точность была достаточной.