# Сортировки с квазилинейной сложностью

## 1 Сложность задачи сортировки

Пусть есть $(a_1, ..., a_n)$ из $n$ элементов линейно упорядоченного множества. Нужно найти перестановку $x$ множества $(1, ..., n)$ такую, чтобы

$$i < j \implies a_{x(i)} < a_{x(j)}, \forall 1 \leq i < n, i < j \leq n.$$

Считаем, что нам ничего не известно о структуре множества и в распоряжении есть только операции сравнения. Оценим минимальное количество операций сравнения, необходимых для решения задачи. Пусть $\Pi_0$ – множество всех перестановок, $|\Pi_0| = n!$. Изначально, все перестановки являются кандидатами на ответ. Сравним два элемента последовательности $a_{i_1}$ и $a_{j_1}$. Множество $\Pi_0$ разобьется на два подмножества $\Pi_1^< = \{\pi \in \Pi_0 : \pi(a_{i_1}) < \pi(a_{j_1})\}$ и $\Pi_1^> = \{\pi \in \Pi_0 : \pi(a_{i_1}) > \pi(a_{j_1})\}$. Выберем подходящее подмножество, назовем его $\Pi_1$. Мощность одного из подмножеств не меньше, чем $n!/2$, в худшем случае $|\Pi_1| \geq n1/2$. После $k$ сравнений мощность множества $\Pi_k$ может достигать $n!/2^k$. Так как искомая перестановка единственна, то получаем, что $k \geq \log_2 n!$. Используем формулу Стирлинга:

$$n! \asymp \sqrt{2\pi n}(\frac{n}{e})^n.$$

$$\log_2 n! = \log_2 \sqrt{2\pi n}(\frac{n}{e})^n = \log_2 \sqrt{2\pi n} + n(\log_2 n - \log_2 e) = n \log_2 n + O(n).$$

Отсюда, сложность задачи сортировки оценивается как $O(n \log n)$. Рассмотрим ряд алгоритмов сортировки с таким временем работы.

## 2 Быстрая сортировка

Сортировка Тони Хоара (быстра сортировка, Quick sort) – один из наиболее популярных алгоритмов. Алгоритм состоит из трех шагов. Сначала выбирается один элемент из массива (опорный элемент). Массив разбивается на две части: в первую часть входят элементы меньше опорного, во вторую часть – больше опорного. Сам опорный элемент может как попасть в одну из частей, так и просто оказаться между ними (зависит от конкретной реализации алгоритма, но при этом не влияет на результат работы алгоритма). Третьим шагом к обоим частям массива рекурсивно применяется данный алгоритм. Массив, который содержит менее двух элементов, является отсортированным, и применять к такому массиву сортировку не надо. Быстрая сортировка является неустойчивой.

## 2.1 Время работы

Докажем, что среднее время работы алгоритма равно $O(n \log n)$.

Пусть $X$ – полное количество сравнений элементов массива с опорным, надо найти это количество. Переименуем элементы массива как $z_1 ... z_n$. Введем множество $Z_{ij} = \{z_i, z_{i+1}, ..., z_j\}$. Не теряя общности, предположим, что все элементы массива различны.

Кажадя пара элементов сравнивается не более одного раза, т.к. сравнения происходят только с опорным элементом, а после разбиения опорный элемент больше не участвует в сравнениях. Вычислим полное количество сравнений:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij},$$

где $X_{ij} = 1$, если произошло сравнение $z_i$ с $z_j$, $X_{ij} = 0$ иначе.

Применим опреацию матожидания и воспользуемся свойством линейности:

$$E[X] = E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} P\{z_i \text{ сравнивается с } z_j\}.$$

Вычислим вероятность того, что $z_i$ сравнивается с $z_j$. Поскольку все элементы различны, то при выборе опорным элементом $x$ впоследствии не будут сравниваться никакие $z_i$ и $z_j$, для которых $z_i < x < z_j$. Если $z_i$ – опорный элемент, то он будет сравниваться с каждым элементом $Z_{ij}$, кроме самого себя. Тогда элементы $z_i$ и $z_j$ сравниваются тогда и только тогда, когда первым в множестве $Z_{ij}$ опорным элементом был выбран один из них.

$$P\{z_i \text{ сравнивается с } z_j\} = P\{\text{первым опорным элементом был } z_i\} +$$

$$P\{\text{первым опорным элементом был } z_j\} = \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k} = \sum_{i=1}^{n-1} O(\log n) = O(n \log n).$$

Матожидание времени работы быстрой сортировки равно $O(n \log n)$.

Однако, в худшем случае время работы сортировки – $O(n^2)$. Такое время достигается, если на каждом шаге рекурсии массив разбивается на части так, что одна часть содержит один элемент, другая – остальные. Например, если опорным элементом является первый элемент, то изначально отсортированный массив будет обрабатыватся $O(n^2)$ времени. Для любого фиксированного положения опорного элемента можно построить контрпример. Для борьбы с этой проблемой есть несколько вариантов. Во первых, на каждом шаге рекурсии можно выбирать случайное положение опорного элемента. Во вторых, можно взять несколько

равномерно расположенных элементов в массиве и посчитать их медиану. Так или иначе, эти способы позволяют избежать подбора контрпримера.

## 2.2 Реализация

В приведенной ниже реализации используется выбор случаного опорного элемента. Изначально функция вызывается: qsort(a, 0, len(a) - 1).

```
from random import randint


def qsort(a, left, right):
    if left >= right:
        return
    i, j = left, right
    x = a[randint(left, right)]
    while i <= j:
        while a[i] < x:
            i += 1
        while a[j] > x:
            j -= 1
        if i <= j:
            a[i], a[j] = a[j], a[i]
            i += 1
            j -= 1
    qsort(a, left, j)
    qsort(a, i, right)
```

Пусть из постановки задачи известно, что в массиве много повторяющихся элементов. Тогда можно изменить приведенную выше реализацию. Теперь будет происходить разбиение массива на три части: элементы меньше опороного, элементы равные опорному и элементы больше опорного. Асимптотика алгоритма остается прежней, но имеет меньшую константу во времени работы.

```
from random import randint


def qsort(a):
    if len(a) < 2:
        return a
    i, j = left, right
    m = a[randint(left, right)]
    x, y, z = [], [], []
    for e in a:
        if e < m:
```

```
            x.append(m)
        elif e == m:
            y.append(m)
        else:
            z.append(m)
    return qsort(x) + y + qsort(z)
```

В такой реализации необходимо $O(n)$ дополнительной памяти в то время, как оригинальынй вариант не использует дополнительную память.

# 3 Сортировка слиянием

Сортировка слиянием представлена двумя шагами: разбиение массива пополам и независимая сортировка обоих частей, слияние двух отсортированных частей массива в один отсортированный массив. Разбиение делается до тех пор, пока части массива не станут содержать ровно один элемент. Слияние двух отсортированных массивов в один отсортированный массив происходит следующим образом:

1. Из обоих массивов берутся первые элементы.
2. Если элемент из первого массива меньше, чем из второго, то он удаляется из первого массива и добавляется в итоговый массив.
3. Иначе в итоговый массив добавляется элемент из второго массива, при этом удаляясь из второго массива.
4. Шаги $1 - 3$ повторяются до тех пор, пока оба массива не пустые.
5. Оставшиеся элементы в непустом массиве добавляются в итоговый массив.

На этапе слияния сортировка использует $O(n)$ дополнительной памяти. Сортировка слиянием является устойчивой.

## 3.1 Время работы

Пусть $T(n)$ – время работы сортировки для $n$ элементов. Оценим $T(n)$. Время работы операции слияния двух массивов длины $n$ оценивается, как $O(n)$. Тогда справедливо соотношение $T(n) = 2T(n/2)+O(n) = 4T(n/4)+2O(n) = ... = nT(1)+\log nO(n)$. Отсюда время работы алгоритма $O(n \log n)$.

## 3.2 Реализация

Функция merge() представляет собой алгоритм слияния двух отсортированных массивов, merge_sort() – функция самого алгоритма сортировки.

```python
def merge(a, b):
    i = j = 0
    n, m = len(a), len(b)
    c = []
    while i < n and j < m:
        if a[i] < b[j]:
            c.append(a[i])
            i += 1
        else:
            c.append(b[j])
            j += 1
    c += a[i:] + b[j:]
    return c


def merge_sort(a):
    if len(a) == 1:
        return a
    m = len(a) // 2
    x = merge_sort(a[:m])
    y = merge_sort(a[m:])
    return merge(x, y)
```

## Список литературы

Кормен, Томас и др. (2013). *Алгоритмы. Построение и анализ. Третье издание*. Издательский дом «Вильямс».