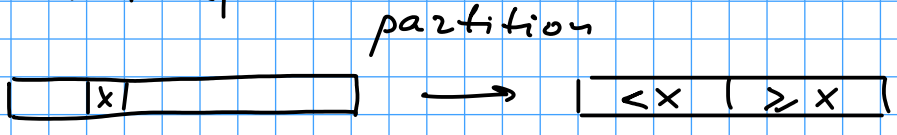
Быстрая сортировка (Quick Sort)

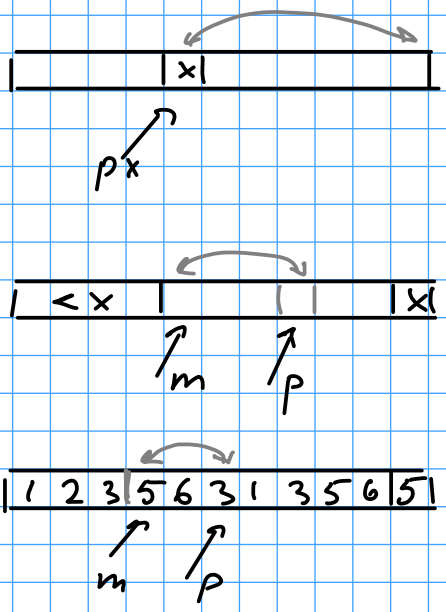
Тони Хоар




```
QuickSort (A, i, j):
    if j - i ≤ 0
        return  // один эл-т в массиве
    px = pivot(A, i, j)  // индекс разделителя
    m = partition (A, i, j, px) // m - новая позиция разделителя
    QuickSort(A, i, m-1)
    QuickSort(A, m+1, j)

partition (A, l, r, px):
    A[px] ⟷ A[r]
    m = l
    for p = l to r-1          pivot
        if A[p] < A[r]:
            A[p] ⟷ A[m]
            m = m+1
    A[r] ⟷ A[m]
    return m

pivot (A, i, j):
    return i
```



] A - sorted

A[1:n]

A[1:1] A[2:n]

A[2:2] A[3:n]

NB: на отсортированном массиве

$$T(n) = T(1) + T(n-1) + O(n)$$

$$T(n) = \sum_{k=1}^{n} \Theta(k) = \Theta(n^2)$$

В худшем случае $\Theta(n^2)$.

Утв: Quick Sort требует $O(n \log n)$ операций „в среднем", т.е. $E[\#\text{операций}] = O(n \log n)$

Какое св-во нужно от разделителя?

1. $T(n) = T(100) + T(n-100) + O(n)$
2. $T(n) = T(\frac{n}{100}) + T(\frac{99n}{100}) + O(n)$

```
pivot(A, i, j):
    return median(A, i, j, (i+j)/2) // 3 эл-та

pivot(A, i, j):
    return median(A, i, j) // для всего массива
pivot(A, i, j):
    return rand(i, j)
```

Вероятностный алгоритм

Детерминированный алгоритм + строка случ. бит.

$A(x, r)$ ← строка случайных битов

↑ вход алгоритма.

1. „Монте Карло"

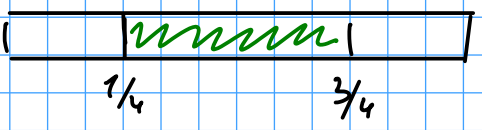
   Быстро выдаёт приближённое решение

2. „Лас Вегас"

   Выдаёт точное решение, но иногда работает долго.

Ожидаемая глубина рекурсии

] хорошее разбиение, если pivot попадает в отрезок $[1/4, 3/4]$



$P[\text{pivot хороший}] = 1/2$

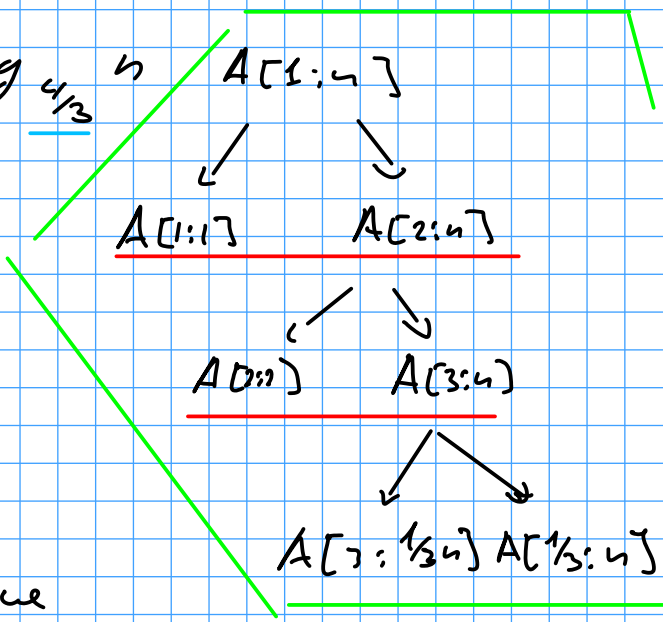
(pivot выбирается случайно)

$E[\#\text{бросков монеты до орла}] = 2$

$E[\dots] = \frac{1}{2} \cdot 1 + \frac{1}{2}(1 + E)$

$E = \frac{1}{2} + \frac{1}{2} + \frac{1}{2}E \Rightarrow E = 2$

выборов pivot $= 2 \log_{4/3} n$

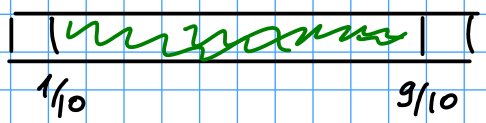


$T(n) = T(\frac{n}{4}) + T(\frac{3n}{4}) + O(n)$

$\geq 2 \cdot T(\frac{3n}{4}) + O(n)$

$b = \frac{4}{3}$

NB: ничего не изменится



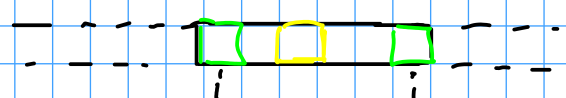
Анализ # сравнений

$E[\#\text{сравнений}] = \sum_{1 \leq i < j \leq n} E[\#\ i \text{ сравнят с } j]$ (в отсорт. массиве)

Утв. 1: ∀ пара сравнивается ≤ 1 раза

Утв. 2: в # сравнениях участвует pivot

$P[i \text{ сравнят с } j] = \frac{2}{j - i + 1}$



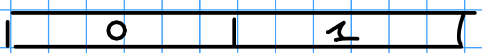
//Предполагаем отсутствие одинаковых элементов//

$$E[\#\ i \text{ сравнится с } j] = P[i \text{ сравнится с } j] = \frac{2}{j-i+1}$$

$$E[\#\text{ сравнений}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = [k = j-i]$$

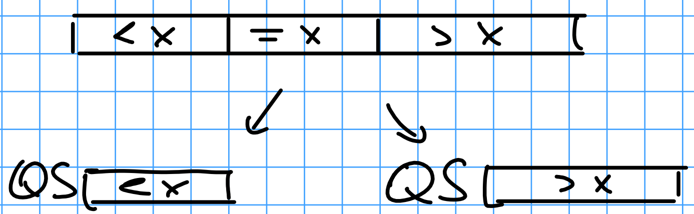
$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \underbrace{\sum_{k=1}^{n} \frac{2}{k+1}}_{= O(\log n)} = \sum_{i=1}^{n-1} O(\log n) <$$

$$< n \cdot O(\log n) = O(n \log n)$$



$\Rightarrow O(n^2)$

1. QuickSort 3: // „работает" и для массивов с одинаковыми эл-тами



2. Intro Sort

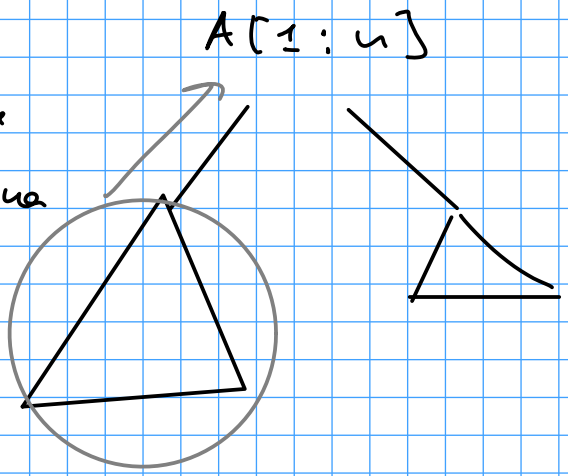
QuickSort и если глубина рекурсии $> c \log n \Rightarrow$ STOP и запускаем HeapSort.

3. QuickSortRec (A, i, j)

```
while j - i > 0 :
    px = pivot (A, i, j)
    m = partition (A, i, j, px)
    if m - i > j - m :
        QuickSort (A, m+1, j)
        j = m - 1
```

else
QuickSort(A, i, m-1)
i = m+1

На # рекурсивном вызова длина массива уменьшается ≥ 2 раза. $\Rightarrow O(\log n)$ - глубина в худшем случае.



4. QuickSort Stable

Переписать partition так, что бы эл-ты переставлялись только целыми числами сдвигами.

partition потребует $O(n \log n)$ в среднем

QuickSort Stable $O(n \log^2 n)$ в среднем