

	Merge Sort	Heap Sort
Время	$O(n \log n)$	$O(n \log n)$
Память	$O(n)$	$O(1)$
Стабильность	+	-
Списки	+	-

### АТД Очередь с приоритетом

- insert (pri, key)
- extract\_max ()
- modify-priority ()

Реализация:	Куча	Массив
insert	$O(\log n)$	$O(1)$
extract_max	$O(\log n)$	$O(n)$
modify-priority	$O(\log n)$	$O(1)$

### Быстрое сортирование (Quick Sort)

Сэр Томас Хоар



Далее рекурсивно отсортируем обе части массива

```

def quick_sort(array, left, right):
    if right <= left + 1:
        return

    pivot = random.randint(left, right - 1)

    mid = partition(array, pivot, left, right)

    quick_sort(array, left, mid)
    quick_sort(array, mid + 1, right)

    return array

```

```

def partition(array, pivot, left, right):
    x = array[pivot]

    swap(array, pivot, right - 1)

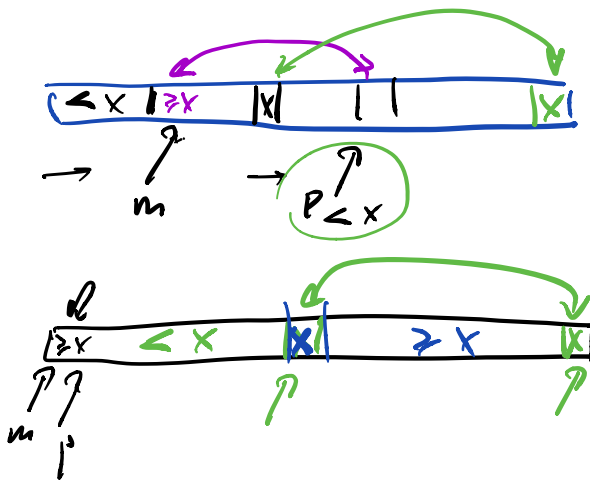
    m = left

    for p in range(left, right - 1):
        if array[p] < x:
            swap(array, p, m)
            m += 1

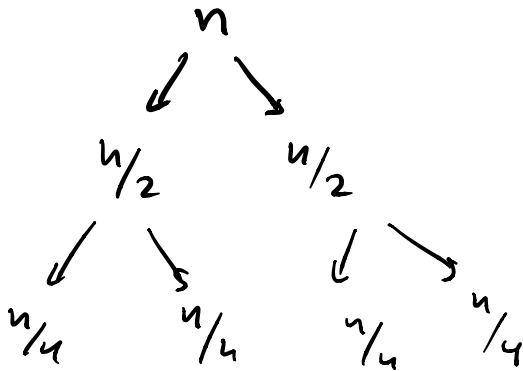
    swap(array, m, right - 1)
    return m

```

$O(n)$



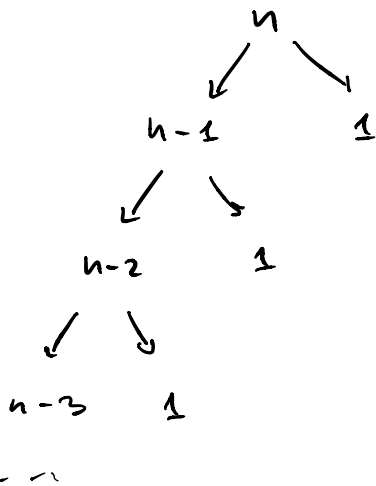
## Идеальный случай



$$T(n) = 2T(n/2) + O(n)$$

$$T(n) = O(n \log n)$$

## Худший случай



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

$$T(n) = \Theta(n^2)$$

Худший случай достигается:

pivot = left

и вырожденный массив

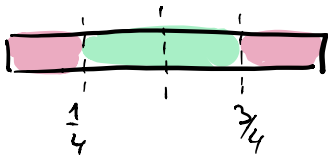
Если выбрать фикс. эл-т (левый, правый, середину, medianу из этих трёх), то всё равно будет  $\Theta(n^2)$  в худшем

Идее:

Давайте выбирать pivot случайно

= Word-RAM with access to random bits

Хорошее разделение

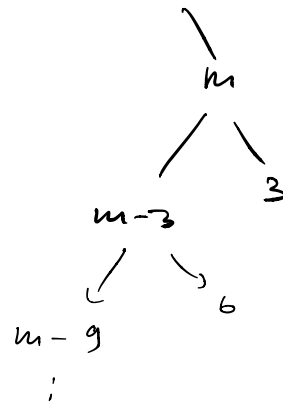


$$T(n) \leq 2T\left(\frac{3}{4}n\right) + O(n)$$

Max массив  $\leq \frac{3}{4}$  исходного

$$P[\text{pivot} \Rightarrow \text{хорошее разделение}] = \frac{1}{2}$$

$E[\# \text{выборов pivot до хорошего разделения}] = ?$



$E[\# \text{повторов попыток до появления оптим. разделения}] = 2$

$$E[\# \dots] = 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \dots = 2 \quad \leftarrow$$

$$\begin{aligned} E[\# \dots] &= \frac{1}{2} \cdot 1 + \frac{1}{2} (1 + E[\# \dots]) = \\ &= 1 + \frac{1}{2} E \quad \Rightarrow \quad \frac{1}{2} E = 1 \quad \Rightarrow \quad E = 2 \end{aligned}$$

$$\sum_{k=0}^{\infty} a^k = \frac{1}{1-a}$$

$$a < 1$$

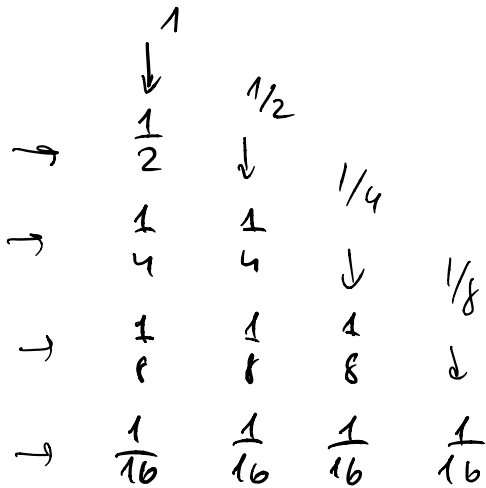
$$f(x) = \sum_{k=0}^{\infty} x^k$$

$$f(x) = \frac{1}{1-x} \quad x < 1$$

$$f'(x) = \frac{1}{(1-x)^2}$$

$$f'(x) = \sum_{k=0}^{\infty} k \cdot x^{k-1}$$

$$\begin{aligned} g(x) &= x \cdot f'(x) \\ g(x) &= \sum_{k=0}^{\infty} k \cdot x^k = \frac{x}{(1-x)^2} \end{aligned}$$



Следствие:

Ожидание глубины рекурсии для  
конкретного  $n$ -го —  $O(\log n)$

$$\begin{aligned}
 E[\ ] &= \log_{4/3} n - E[\# \text{ выходов до уровня} \\
 &\quad \text{глубины}] \\
 &= 2 \cdot \log_{4/3} n
 \end{aligned}$$

Сложность:

$\equiv$  Сложность вероятностного алгоритма —  
 $E[\# \text{ операций}]$   
 (сложность в среднем)

$\boxed{132507}$   
 $13240$

Анализ # сравнений

$$E[\# \text{ сравнений}] = \sum_{1 \leq i \leq j \leq n} E[\# \text{ сравнений } i \sim j]$$

индекс  $i$   
 упоряд. масс.  $\rightarrow$

Утв 1:  $\forall$  пара  $i, j$  сравнивается  $\leq 1$  раз

Утв 2: В  $\forall$  сравнении участвует pivot

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

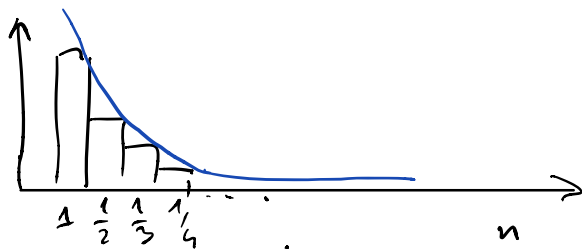


$\leftarrow$  упоряд. массив

$$E[\# \text{ сравнений } i \text{ и } j] = 1 \cdot P[i \text{ ср. с } j] = \frac{2}{j-i+1}$$

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

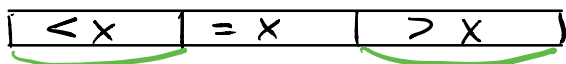
$$= \sum_{i=1}^n \sum_{k=1}^{n-i} \frac{2}{k+1} \leq \sum_{i=1}^n O(\log n) = O(n \log n)$$



$$\sum_{k=1}^n \frac{1}{k} < \int_1^n \frac{1}{x} \cdot dx = \ln n$$

Ув: Мы предполагаем, что все эл-ты массива различны

Запустим QS на массиве  $u$  всех нулей  $\Rightarrow \Theta(n^2)$



```

def quick_sort3(array, left, right):
    if right <= left + 1:
        return

    pivot = random.randint(left, right - 1)

    (m1, m2) = partition3(array, pivot, left, right)

    quick_sort3(array, left, m1)
    quick_sort3(array, m2, right)

    return array

```

```

def partition3(array, pivot, left, right):
    x = array[pivot]

    swap(array, pivot, right - 1)

    m = left
    p = left
    q = right - 1

    while p < q:
        if array[p] == x:
            q -= 1
            swap(array, p, q)
        else:
            if array[p] < x:
                swap(array, p, m)
                m += 1
            p += 1

    for i in range(q, right):
        swap(array, m + i - q, i)

    return m, m + right - q

```