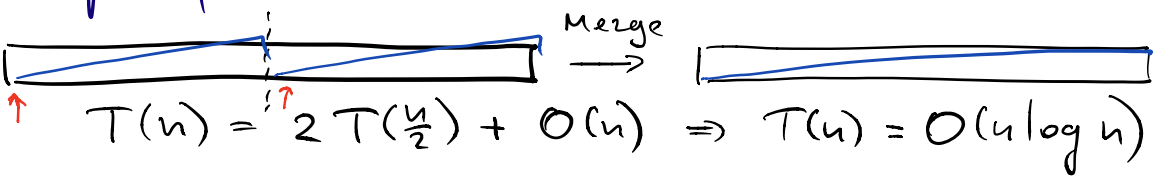
Сортировка слиянием



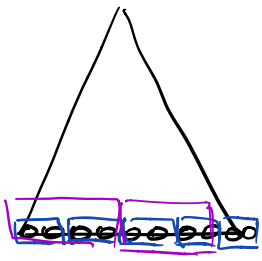

```python
# сортировка слиянием
def merge_sort(array):
    n = len(array)

    # пустой и одноэлементный массивы сортировать не нужно
    if n < 2:
        return array

    # рекурсивно сортируем левую и правую половины массива
    array1 = merge_sort(array[:n/2])
    array2 = merge_sort(array[n/2:])

    # возвращаем результат слияние двух упорядоченных массивов
    return merge(array1, array2)

# слияние двух упорядоченных массивов
def merge(array1, array2):
    n, m = len(array1), len(array2)

    # переменные цикла
    i, j = 0, 0

    # массив для результата слияния
    result = []

    # перебираем элементы массивов
    while i < n and j < m:
        # на каждой итерации выбираем минимальный из текущих
        # и записываем его в массив с результатом
        if array1[i] < array2[j]:
            result.append(array1[i])
            i = i + 1
        else:
            result.append(array2[j])
            j = j + 1

    # добавляем в конец оставшиеся элементы
    while i < n:
        result.append(array1[i])
        i = i + 1

    while j < m:
        result.append(array2[j])
        j = j + 1

    return result
```

Память $O(n)$

Нерек.!

Очередь

Заполняем массивы размера 1



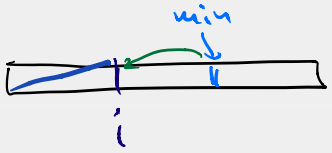
2.1. Квадратичные сортировки

```python
def swap(array, i, j):
    array[i], array[j] = array[j], array[i]
```

Перестановка

```python
def selection_sort(array):
    n = len(array)

    for i in range(n):
        min_pos = i
        for j in range(i + 1, n):
            if array[min_pos] > array[j]:
                min_pos = j

        swap(array, i, min_pos)

    return array
```

Сортировка выбором

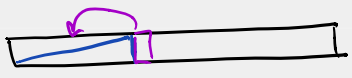


$\sum_{i=0}^{n-1} (n-i) = \Theta(n^2)$

Перестановок $O(n)$

```python
def insertion_sort(array):
    n = len(array)

    for i in range(n):
        for j in range(i, 0, -1):
            if array[j - 1] > array[j]:
                swap(array, j - 1, j)

    return array
```

Сортировка вставкой

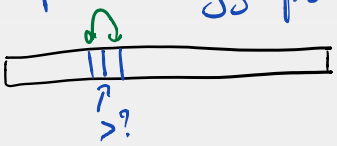


Время $\Theta(n^2)$

Бин Поиск $\Rightarrow$ $O(n \log n)$ сравнений

```python
def bubble_sort(array):
    n = len(array)

    for i in range(n - 1):
        for j in range(n - i - 1):
            if array[j] > array[j + 1]:
                swap(array, j, j + 1)

    return array
```

Сортировка пузырьком

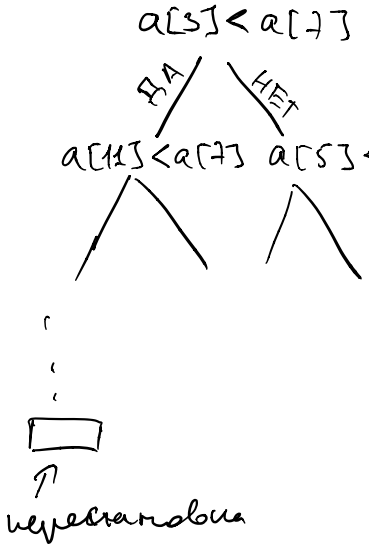


Сравнение и перед: $O(n^2)$

Нижняя оценка на сортировку сравнениями

Th. $\forall$ сортировка сравнениями совершает $\Omega(n \log n)$ сравнений на массиве длины n.

Дерево сравнений



Утв: ∀ дерево сравнений, кот. сортирует массив длины n имеет $\geq n!$ листьев

перестановок n эл-ов

3	7	5
1	3	2

массив → | 3 | 5 | 7 |

Следствие: Глубина дерева $\geq \log n!$

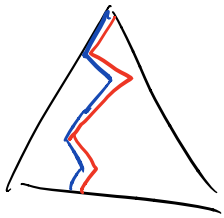
Полное дерево глубина $d \Rightarrow 2^d$ листьев

$$\log n! = \log 1 + \log 2 + \log 3 + \ldots + \log n$$

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

$$\log n! \geq \frac{n}{2} \cdot \log \frac{n}{2} = \Omega(n \log n)$$

Утв: По ∀ алгоритму сорт. ср. и длине n можно построить сортирующ. дерево сравнений

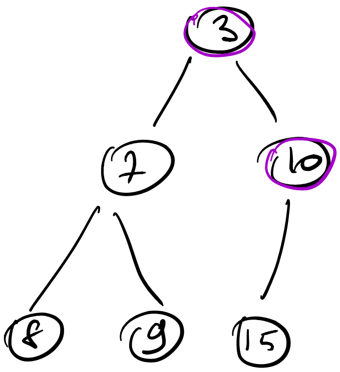


$\leq k$ вопросов

Вариантов ответов $\leq 2^k$

$2^k \geq n!$ $\quad k = \Omega(n \log n)$

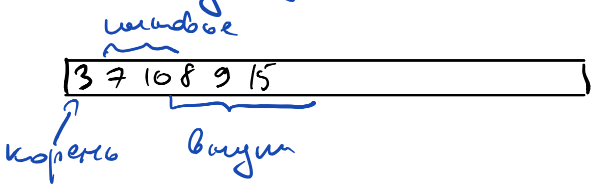
Куча (Пирамида)   Heap
(Двоичная куча)   Binary heap.



≡ Двоичная min куча — это полное двоичное дерево, где для каждого эл-та выполн. свойство кучи:

$value(x) \geq value(parent(x))$

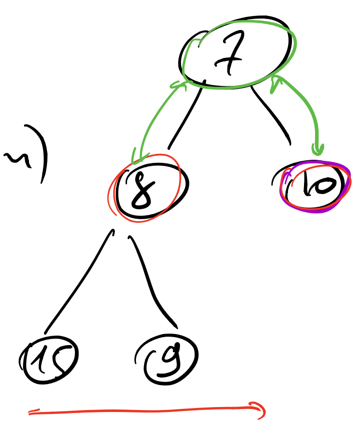
Реализация на массиве



Нумерация с 1	Нумерация с 0
root = 1	root = 0
$left(i) = 2i$	$left(i) = 2i+1$
$right(i) = 2i+1$	$right(i) = 2i+2$
$parent(i) = \lfloor i/2 \rfloor$	$parent(i) = \lfloor \frac{i-1}{2} \rfloor$

- Min за $O(1)$
- ExtractMin(): $O(\log n)$

```
res = value[root]
value[root] = value[n-1]
n = n - 1
Heapify(root)
```



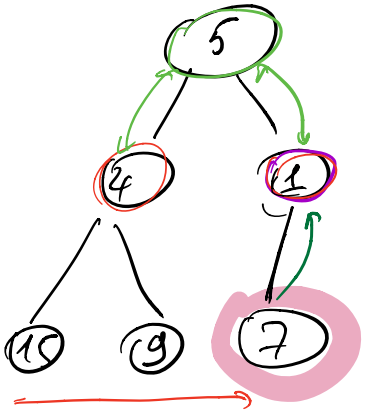
return res

```
Heapify(i):
    min = i
    if left(i) < n and value[left(i)] < value[i]:
        min = left(i)
    if right(i) < n and value[right(i)] < value[min]:
        min = right(i)
    if i ≠ min:
        swap(value[i], value[min])
        Heapify(min)
```

Время: $O(h) = O(\log n)$

- Insert(x): $O(\log n)$

```
value[n] = x
i = n
n = n + 1
DecreaseKey(i)
```



- DecreaseKey(i):

```
while value[i] < value[parent(i)]:
    swap(value[i], value[parent(i)])
    i = parent(i)
```

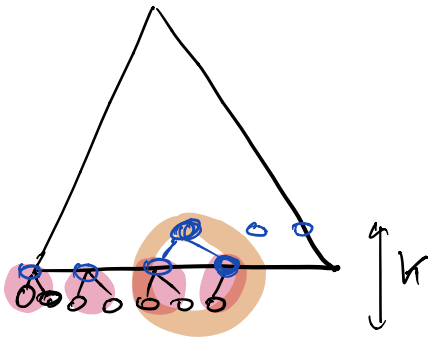
$O(\log n)$

Построение за $O(n \log n)$ — очевидно

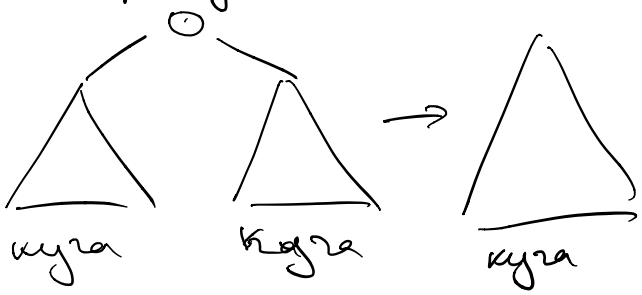
Построение за $O(n)$

```
Make_Heap(A):
    for i = size(A)/2 to 0:
        Heapify(i)
```

Max i: $left(i) < n$



Heapify:



$$\sum_{i=\frac{n}{2}}^{0} O(\log n) = O(n \log n)$$

$$\sum_{k=1}^{\log n} \frac{n}{2^k} O(k) = n \left(\sum_{k=1}^{\log n} \frac{k}{2^k} \right) = O(n)$$

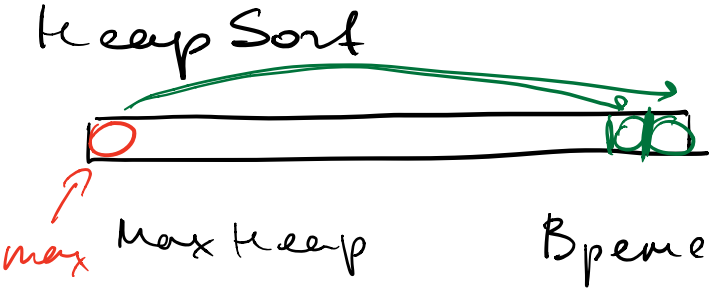
сложность heapify

вершин на уровне h

$$\sum \frac{k}{2^k} = O(1)$$

Упр

Heap Sort



Время: Make Heap +
n. ExtractMax

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

$= O(n \log n)$