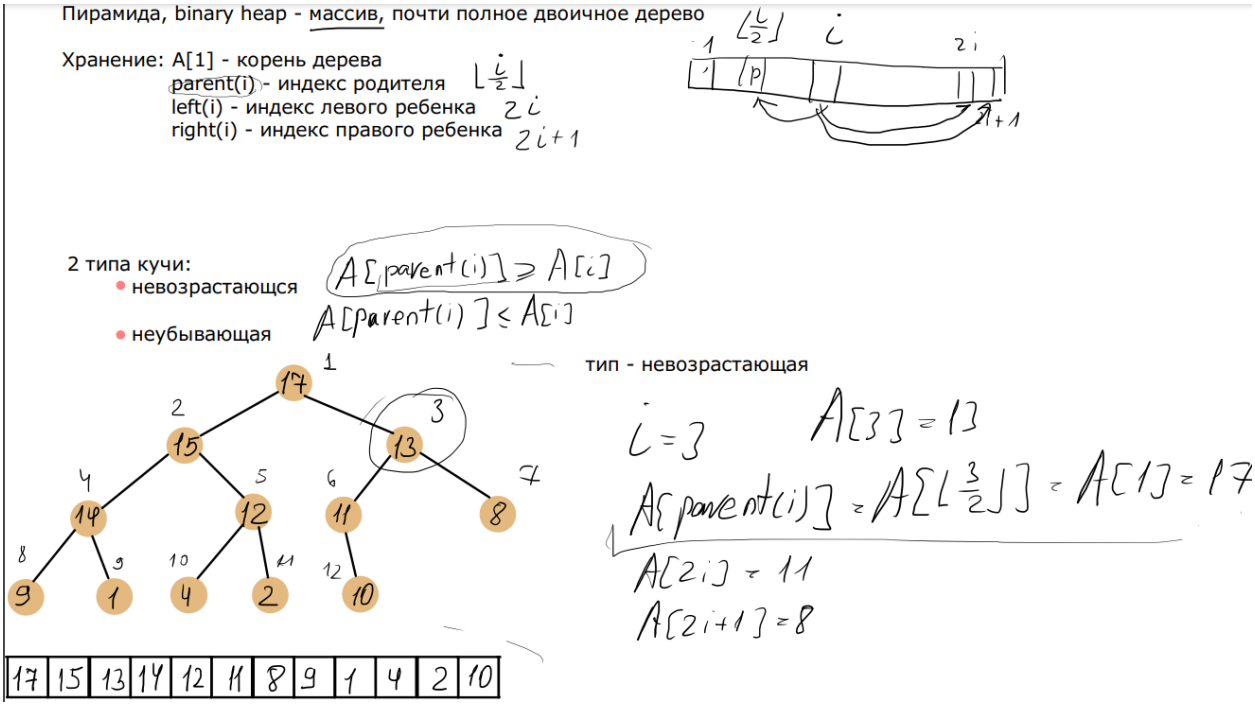


Восстановление свойств кучи (неубывающей)

Если в куче изменяется один из элементов, то она может перестать удовлетворять свойству упорядоченности. Для восстановления этого свойства служат процедуры siftDown (просеивание вниз) и siftUp (просеивание вверх).

siftDown

Если значение измененного элемента увеличивается, то свойства кучи восстанавливаются функцией siftDownsiftDown.

Работа процедуры: если ii-й элемент меньше, чем его сыновья, всё поддерево уже является кучей, и делать ничего не надо. В противном случае меняем местами ii-й элемент с наименьшим из его сыновей, после чего выполняем siftDownsiftDown для этого сына.

Процедура выполняется за время O(logn)

```
function siftDown(i : int):
    while 2 * i + 1 < a.heapSize      // heapSize — количество элементов в
куче
        left = 2 * i + 1              // left — левый сын
        right = 2 * i + 2             // right — правый сын
        j = left
        if right < a.heapSize and a[right] < a[left]
            j = right
        if a[i] <= a[j]
            break
        swap(a[i], a[j])
        i = j
```

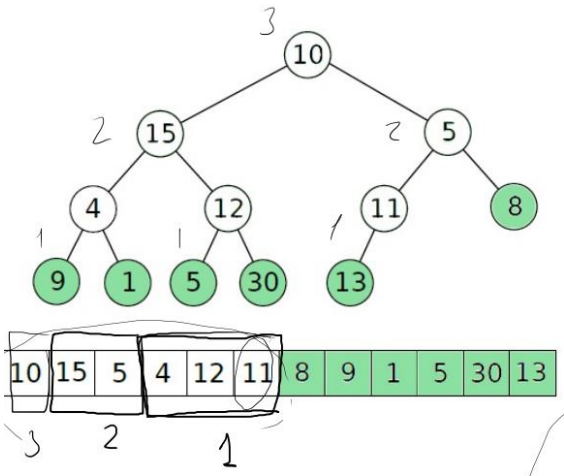
siftUp

Если значение измененного элемента уменьшается, то свойства кучи восстанавливаются функцией siftUpsiftUp.

Работа процедуры: если элемент больше своего отца, условие 1 соблюдено для всего дерева, и больше ничего делать не нужно. Иначе, мы меняем местами его с отцом. После

чего выполняем siftUpsiftUp для этого отца. Иными словами, слишком маленький элемент всплывает наверх. Процедура выполняется за время $O(\log n)$

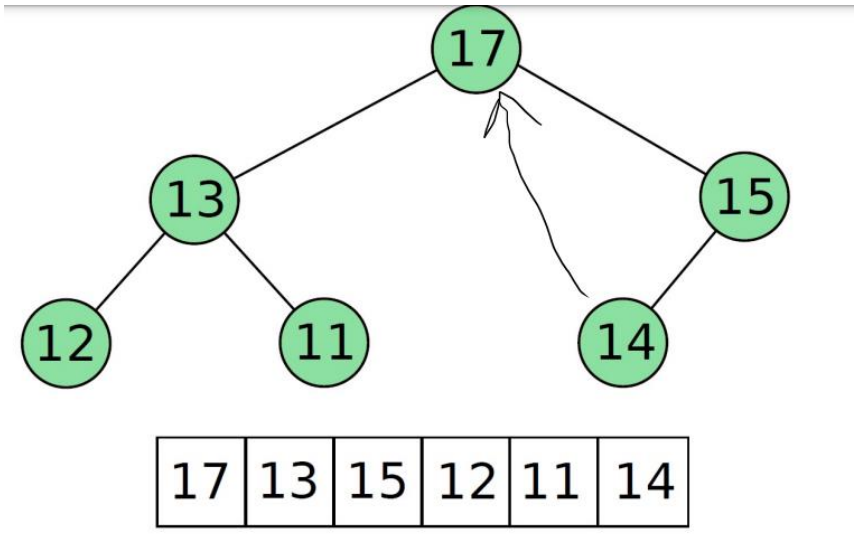
```
function siftUp(i : int):
    while a[i] < a[(i - 1) / 2]      // i  0 — мы в корне
        swap(a[i], a[(i - 1) / 2])
        i = (i - 1) / 2
```



BUILD-MAX-HEAP(A)
1 $A.heap\text{-}size = A.length$
2 **for** $i = \lfloor A.length/2 \rfloor$ **downto** 1
3 MAX-HEAPIFY(A, i)

$\frac{n}{2} \cdot 0 + \frac{n}{4} \cdot 1 + \frac{n}{8} \cdot 2 + \frac{n}{16} \cdot 3 + \dots$

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



HEAPSORT(A) $O(n)$
1 BUILD-MAX-HEAP(A)
2 **for** $i = A.length$ **downto** 2
3 Обменять $A[1]$ с $A[i]$
4 $A.heap\text{-}size = A.heap\text{-}size - 1$
5 MAX-HEAPIFY($A, 1$)

Очередь с приоритетами на основе кучи

Очередь, которая просматривает (удаляет) самый лучший (с наивысшим приоритетом) элемент

Каждый элемент содержит ключ, который определяет его приоритет

Операции:
1) Heap_Maximum(A) - сообщает ключ элемента с наибольшим приоритетом
2) Extract_Max(A) - извлекает из структуры элемент с наибольшим приоритетом
3) Increase_Key(A, i, key) - меняет приоритет элемента A(i) на значение key
4) Insert(A, key) - добавляет в кучу новый элемент с приоритетом key