Куча

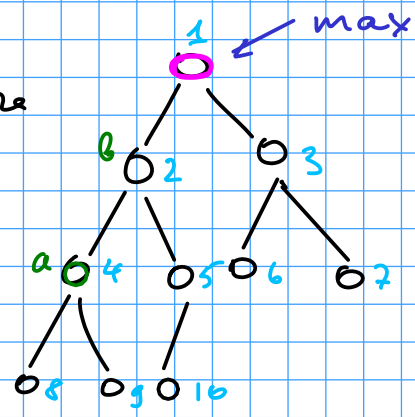
Двоичная куча

≡ полное двоичное дерево

→ $\forall v: V \le parent(V) \Rightarrow$ max-куча

"Свойство кучи"

→ $\forall v: V \ge parent(V) \Rightarrow$ min-куча



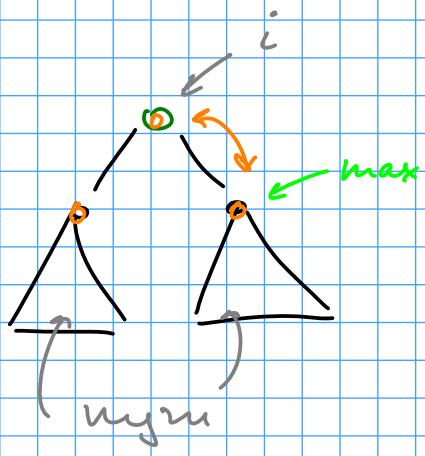
Реализация на массиве

H - массив [1 : n]

Left(i) = 2i

right(i) = 2i + 1

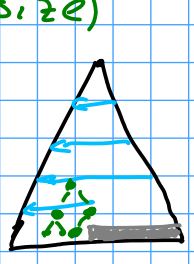
parent(i) = ⌊i/2⌋




```
Heapify(i):
    largest = i
    if left(i) ≤ size &&
        value(left(i)) > value(i):
        largest = left(i)
    if right(i) ≤ size &&
        value(right(i)) > value(largest):
        largest = right(i)
    if largest ≠ i:
        swap(value(i), value(largest))
        heapify(largest)
```

O(height)

```
make_heap():
    for i = ⌊size/2⌋ to 1
        Heapify(i)
```



„Конечная оценка" $O(n \log n)$ $= 2$

$$T(n) = \sum_{h=1}^{\log n} \frac{n}{2^h} \cdot O(h) \leq n \cdot O\left(\sum_{h=1}^{\infty} \frac{h}{2^h}\right)$$

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

$$\left(\sum_{k=0}^{\infty} a^k\right)' = \sum_{k=0}^{\infty} k a^{k-1} = \frac{1}{(1-a)^2} \Rightarrow$$

$$\left(\frac{1}{1-a}\right)' = \frac{1}{(1-a)^2} \Rightarrow \sum_{k=0}^{\infty} k a^k = \frac{a}{(1-a)^2}$$

Extract_max ():

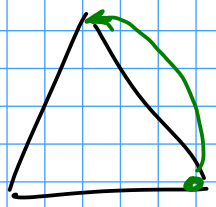
max = value(1)

value(1) = value(size)

size = size − 1

Heapify (1)

return max



$O(\text{height}) = O(\log n)$

Insert (v):

size = size + 1

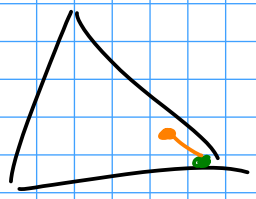
value(size) = v

s = size

* while s > 1 && value(parent(s)) < value(s):

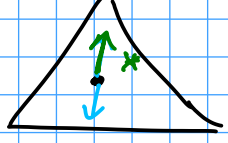
swap (value(s), value(parent(s)))

s = parent(s)



$O(\log n)$

Increase key / Decrease key (i, v)



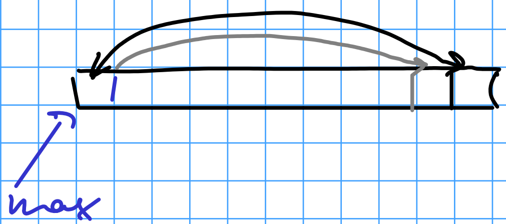
Применение:

1. Очередь с приоритетами

```
enqueue(p, v):   Insert((p, v))     ← ключ для кучи
dequeue():
    (p, v) = ExtractMax()
    return v
```

2. Сортировка кучей (Heap Sort)

```
HeapSort(A):
    Make_Heap(A)
    while (heap-size > 0):
  *   | max = ExtractMax()
      | A[heap-size + 1] = max
```




```
  *   | swap(A[1], A[heap-size])
      | heap_size = heap-size - 1
      | Heapify(1)
```

Плюсы: $O(n \log n)$, $O(1)$ доп. памяти $O(n \log n)$

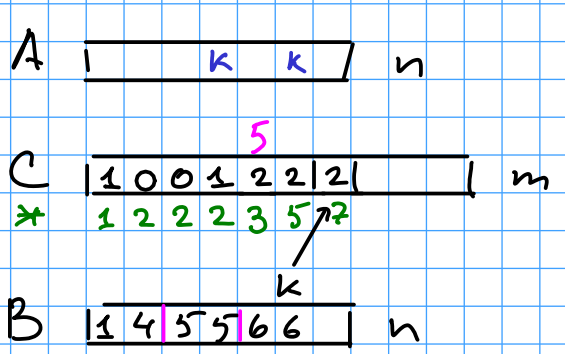
Минусы: не стабильная

Линейные сортировки

1. Counting Sort (сортировка подсчётом)

Пусть в массив A все числа из инт. $[1, m]$ (целые числа)

```
Counting Sort (A[1:n])
    C[1:m]
    for i = 1 to n:
        C[A[i]] += 1
    k = 1
    for i = 1 to m:
        while C[i] > 0:
            A[k] = i
            k = k + 1
            C[i] -= 1
    B[1:n]
    s = 1
    for i = 1 to m:
        s += C[i]
        C[i] = s - C[i]
    for i = 1 to n:
        B[C[A[i]]] = A[i]
        C[A[i]] += 1
```



Наивная реализация

$O(n+m)$

2. Поразрядная сортировка (Radix sort)

d - разрядность / длина макс. строки

0	2	3	1	7	5	4	
1	2	3	3	4	7	8	
4	5	7	4	3	2	1	
2	1	2	8	5	0	6	
4	3	2	1	1	9	8	
3	1	4	1	5	9	0	

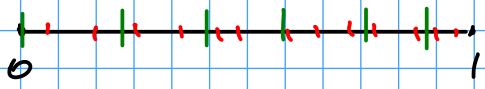
3 2 1

Последовательная стабильная сортировка по разрядам справа налево ⇒ сортировка

$O(n \cdot d)$ при исп. CountingSort

3. Bucket Sort

Для равномерно распределённых на отр.



В каждом bucket с одинаковое # эл-ов.
Сортируем # bucket и вписываем
все эл-ты.