Heap (heapq)

```
class element:
    def __init__(self, x): self.x = x
    def __lt__(self, other): return self.x < other.x
    def __repr__(self): return repr(self.x)

from heapq import *
heapify(heap)
heappush(heap, k)
heappop(heap)
heappushpop(heap, k)  # -> push -> pop
heapreplace(heap, k)  # -> pop -> push

nlargest( 3, (element(x) for x in range(10)) )
nsmallest( 3, (element(x) for x in range(10)) )
single_sorted_iterables = heapq.merge(*list_of_sorted_iterables)
```

max/min/k-th/median

Kth Largest Element in a Stream `min heap of size K`
Kth Largest Element in an Array `min heap of size K`
Find Median from Data Stream `two heaps`

top-k `nlargest`

Top K Frequent Elements `nlargest: number_count`
Top K Frequent Words `nlargest: word_count`
Sort Characters By Frequency `nlargest: char_count`

sorted by row & col

Kth Smallest Element in a Sorted Matrix `heapq.merge`
Find K Pairs with Smallest Sums `heapq.merge`

BFS with heap-queue

K-th Ugly Number II `BFS heap[1]`
K-th Super Ugly Number `BFS heap[1]`
Swim in Rising Water `BFS heap[(g00, 0, 0)]`
Trapping Rain Water I (1D) `BFS heap[boundary]`
Trapping Rain Water II (2D) `BFS heap[boundary]`

max/min/k-th/median @ sliding window

Sliding Window Maximum `hard`
Sliding Window Median `hard`

```
### sliding window maximum ###
def maxSlidingWindow(self, nums, k):
    if not nums: return []
    if k == 1: return nums
    deq = collections.deque()
    result = []

    for i in range(k):
        while len(deq) != 0:
            if nums[i] > nums[deq[-1]]: deq.pop()
            else: break
        deq.append(i)
    result.append(nums[deq[0]])

    for i in range(k, len(nums)):
        if deq[0] < i - k + 1: deq.popleft()
        while len(deq) != 0:
            if nums[i] > nums[deq[-1]]: deq.pop()
            else: break
        deq.append(i)
        result.append(nums[deq[0]])

    return result
```

还没做

Minimum Cost to Hire K Workers
Split Array into Consecutive Subsequences
Reachable Nodes In Subdivided Graph

hard

Find K-th Smallest Pair Distance `hard`
K-th Smallest Prime Fraction `hard`
Minimum Number of Refueling Stops `hard`
IPO `hard`

The Skyline Problem `hard`

```
def getSkyline(self, buildings):
        events = sorted(list((L, -H, R) for L, R, H in buildings)
                        + list((R, H, None) for L, R, H in buildings))
        res, maxH = [], [(0, math.inf)]
        for x, h, r in events:
            while x > maxH[0][1]: heapq.heappop(maxH)
            if h < 0:  # start
                if -h > -maxH[0][0]:  res.append([x, -h])
                heapq.heappush(maxH, (h, r))
            else:      # close
                if h == -maxH[0][0]:
                    heapq.heappop(maxH)
                    while x >= maxH[0][1]: heapq.heappop(maxH)
                    if -maxH[0][0] < h: res.append([x, -maxH[0][0]])
        return res
```