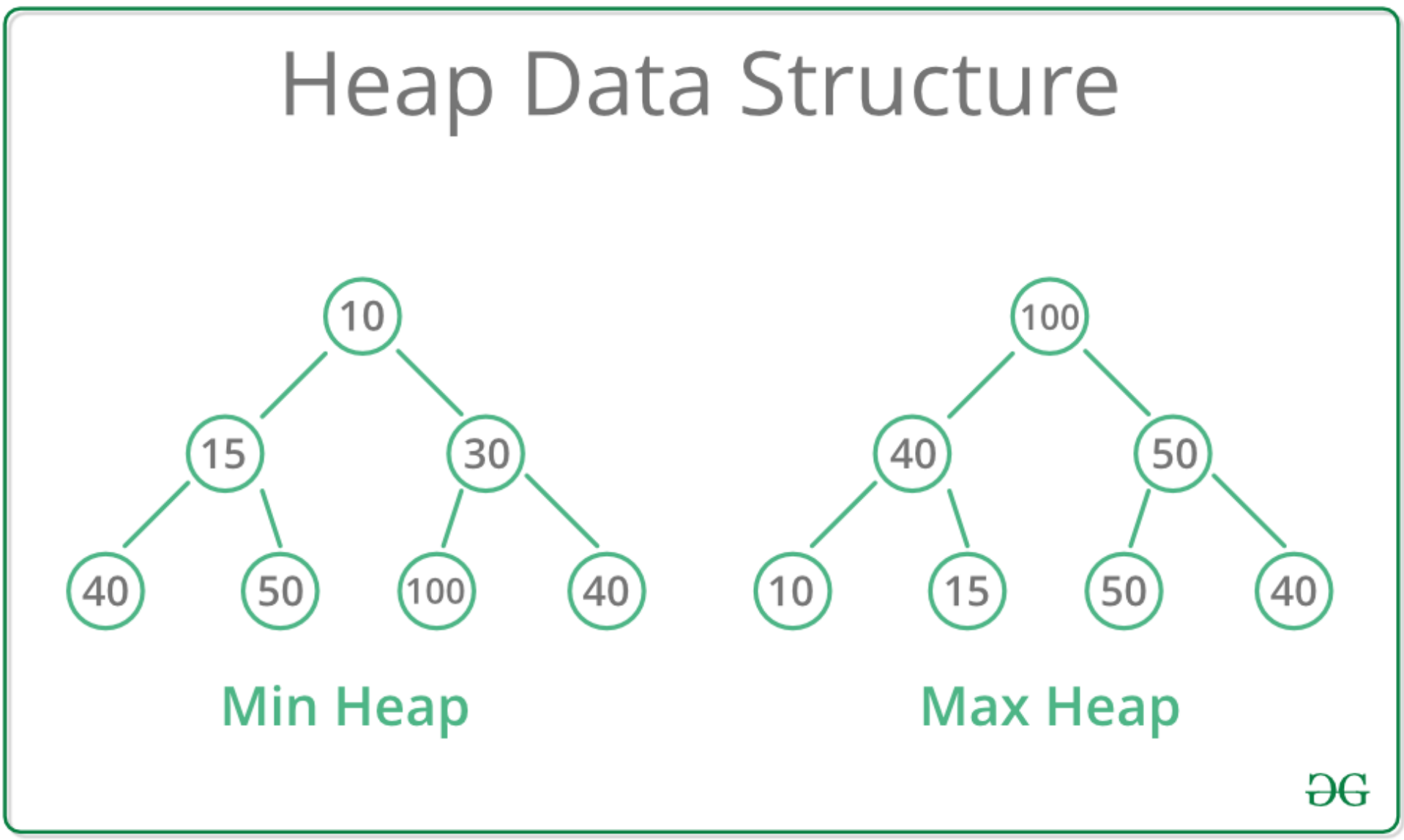
3. Heap-Queues Notes

From What's Next

#queues #heap #bfs

What is Heap Data Structure?

A Heap is a special Tree-based data structure in which the tree is a complete binary tree.



Operations of Heap Data Structure:

- **Heapify:** a process of creating a heap from an array.
- **Insertion:** process to insert an element in existing heap time complexity O(log N).
- **Deletion:** deleting the top element of the heap or the highest priority element, and then organizing the heap and returning the element with time complexity O(log N).
- **Peek:** to check or find the first (or can say the top) element of the heap.

Types of Heap Data Structure

Generally, Heaps can be of two types:

1. **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it’s children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it’s children. The same property must be recursively true for all sub-trees in that Binary Tree.

Understanding Heap

Suppose you're given a list of numbers, it doesn't matter if they are sorted or unsorted, but you're asked to find out the 3^{rd} largest or the 2^{nd} largest number or k^{th} largest number in that list.

1. In a sorted list, the k^{th} element is the k^{th} largest element. So, that'll do.
2. What do you do for an unsorted list?
    - Yes, the answer is you create a ***Min Heap***, where root node is always the smallest.
      A ***Heap*** is a Queue-like Data Structure so it follows ***FIFO*** *(First in First Out)*

- Then you pop elements until the size of the list is **k** .e

Now imagine if you had to calculate that manually.

Heapq in Python

- Python doesn't have Max Heap.
- It only implements Min Heap.
- The workaround is to convert the list of integers to negative integers and apply heapify function.

1. Leetcode - Kth Largest Element In a Stream

#element_in_stream

Easy

Design a class to find the kth largest element in a stream. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Implement KthLargest class:

KthLargest(int k, int[] nums) Initializes the object with the integer k and the stream of integers nums.
int add(int val) Appends the integer val to the stream and returns the element representing the kth largest element in the stream.

Example 1:

Input
["KthLargest", "add", "add", "add", "add", "add"]
[[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]
Output
[null, 4, 5, 5, 8, 8]

Explanation:

```
KthLargest kthLargest = new KthLargest(3, [4, 5, 8, 2]);
kthLargest.add(3);   // return 4
kthLargest.add(5);   // return 5
kthLargest.add(10);  // return 5
kthLargest.add(9);   // return 8
kthLargest.add(4);   // return 8
```

Constraints:

```
1 <= k <= 104
0 <= nums.length <= 104
-104 <= nums\[i] <= 104
-104 <= val <= 104
```

At most 104 calls will be made to add.
It is guaranteed that there will be at least k elements in the array when you search for the kth element.

Code

```
import heapq
class KthLargest:
        def __init__(self, k, nums):
                self.minHeap, self.k = nums, k
                heapq.heapify(self.minHeap)
                while len(self.minHeap)>k:
                        heapq.heappop(self.minHeap)
        def add(self, val):
                heapq.heappush(self.minHeap, val)
                if len(self.minHeap)>self.k:
```

```
                        heapq.heappop(self.minHeap)
                return self.minHeap[0]

param1= ["KthLargest", "add", "add", "add", "add", "add"]
param2 = [[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]
output = []
for i in range(len(param1)):
        if param1[i] == "KthLargest":
                obj = KthLargest(param2[i][0], param2[i][1])
                output.append(None)
        elif param1[i] == "add":
                output.append(obj.add(param2[i][0]))
print(output)
```

Explanation

```
Operation 1: "KthLargest"
obj = KthLargest( k = 3, nums = [4, 5, 8, 2]):
                def __init__ (self, 3, [4, 5, 8, 2]):
                        heapq.heapify([4, 5, 8, 2]) >> nums = [2, 4, 8, 5]
                        while len(nums)>k:
                                heapq.heappop([2, 4, 8, 5]) >> nums = [4,8,5]
output.append(None)
output: [None]

Operation 2: "add"
output.append(obj.add(3):
                                heapq.heappush(3) >> nums = [3, 4, 5, 8]
                                if len([3, 4, 5, 8]) > 3: #True
                                        heapq.heappop([3, 4, 5, 8]) >> nums = [4,5,8]
                                return nums[0] >> 4
                        )
output: [None, 4]

Operation 3: "add"
output.append(obj.add(5):
                                heapq.heappush(5) >> nums = [4, 5, 5, 8]
                                if len([4, 5, 5, 8]) > 3: #True
                                        heapq.heappop([4, 5, 5, 8]) >> nums = [5,5,8]
                                return nums[0] >> 5
                        )
output: [None, 4, 5]

Operation 4: "add"
output.append(obj.add(10):
                                heapq.heappush(10) >> nums = [5, 5, 8, 10]
                                if len([5, 5, 8, 10]) > 3: #True
                                        heapq.heappop([4, 5, 5, 8]) >> nums = [5, 8, 10]
                                return nums[0] >> 5
                        )
output: [None, 4, 5, 5]

Operation 5: "add"
output.append(obj.add(9):
                                heapq.heappush(9) >> nums = [5, 8, 10, 9]
                                if len([5, 8, 10, 9]) > 3: #True
                                        heapq.heappop([5, 8, 10, 9]) >> nums = [8, 9, 10]
                                return nums[0] >> 8
                        )
output: [None, 4, 5, 5, 8]

Operation 6: "add"
output.append(obj.add(4):
                                heapq.heappush(9) >> nums = [4, 8, 10, 9]
                                if len([4, 8, 10, 9]) > 3: #True
                                        heapq.heappop([4, 8, 10, 9]) >> nums = [8, 9, 10]
                                return nums[0] >> 8
                        )
output: [None, 4, 5, 5, 8, 8]
```

2. Leetcode - Last Stone Weight

#last_stone_weight

Easy

You are given an array of integers `stones` where `stones[i]` is the weight of the `ith` stone.

We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights `x` and `y` with `x <= y`. The result of this smash is:

- If `x == y`, both stones are destroyed, and
- If `x != y`, the stone of weight `x` is destroyed, and the stone of weight `y` has new weight `y - x`.

At the end of the game, there is **at most one** stone left.

Return *the weight of the last remaining stone*. If there are no stones left, return `0`.

Example 1:

Input: stones = [2,7,4,1,8,1]
Output: 1
Explanation:
We combine 7 and 8 to get 1 so the array converts to [2,4,1,1,1] then,
we combine 2 and 4 to get 2 so the array converts to [2,1,1,1] then,
we combine 2 and 1 to get 1 so the array converts to [1,1,1] then,
we combine 1 and 1 to get 0 so the array converts to [1] then that's the value of the last stone.

Example 2:

Input: stones = [1]
Output: 1

Constraints:

- `1 <= stones.length <= 30`
- `1 <= stones[i] <= 1000`

Code

```
import heapq
def LastStoneWeight(stones):
    """
    :type stones: List[int]
    :rtype: int
    """
    stones = [-stone for stone in stones] #max heap
    heapq.heapify(stones)
    while len(stones) > 1:
        first = heapq.heappop(stones)
        second = heapq.heappop(stones)
        if second > first:
            heapq.heappush(stones, first-second) #first one is the largest so anyway -8 -(-7) = -1
    stones.append(0)
    return abs(stones[0])
print(LastStoneWeight([3,1]))
print(LastStoneWeight(stones = [2,7,4,1,8,1]))
```

Explanation

```
def LastStoneWeight([3,1]):
        stones:  [-3, -1]
        while len([-3, -1])>1:
                first:  -3
                second:  -1
                -1 > -3:
                        push: -2
                        stones:  [-2]
                        #len(stones):  1
```

```
        stones.append(0) >> stones: [-2, 0]
        return result:  2

def LastStoneWeight([2,7,4,1,8,1]):
        stones:  [-8, -7, -4, -1, -2, -1]
        while len(stones)>1:
                #iter 1
                first:  -8
                second:  -7
                -7 > - 8:
                        pushing -1
                        stones:  [-4, -2, -1, -1, -1]
                # len(stones) = 5
                #iter2
                first:  -4
                second:  -2
                -2 > 4:
                        pushing -2
                        stones:  [-2, -1, -1, -1]
                # len(stones) = 4

                first:  -2
                second:  -1
                -1 > -2:
                        pushing -1
                        stones:  [-1, -1, -1]
                #len(stones):  3

                first:  -1
                second:  -1
                -1 > -1 ? >> False
                #len(stones):  1
                #while loop ends
        stones.append(0) >> stones: [-1,-1,-1, 0]
        result:  1
```

3. Leetcode - K Closest Points to Origin

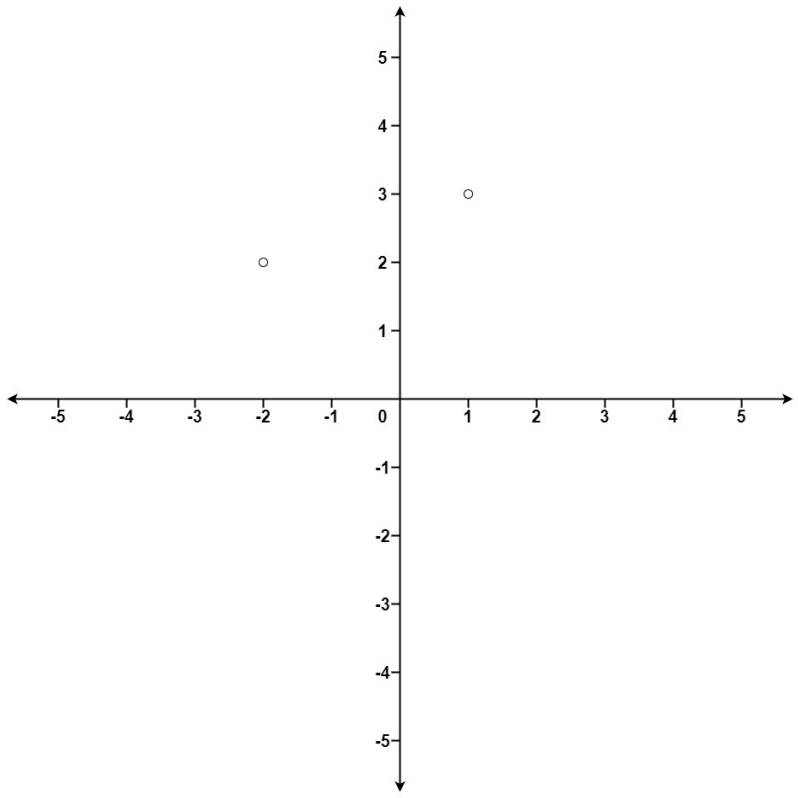
Medium

Given an array of `points` where `points[i] = [xi, yi]` represents a point on the **X-Y** plane and an integer `k`, return the `k` closest points to the origin `(0, 0)`.

The distance between two points on the **X-Y** plane is the Euclidean distance (i.e., `√(x1 - x2)2 + (y1 - y2)2`).

You may return the answer in **any order**. The answer is **guaranteed** to be **unique** (except for the order that it is in).

Example 1:



Input: points = [[1,3],[-2,2]], k = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).

The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest k = 1 points from the origin, so the answer is just [[-2,2]].

Example 2:

Input: points = [[3,3],[5,-1],[-2,4]], k = 2
Output: [[3,3],[-2,4]]
Explanation: The answer [[-2,4],[3,3]] would also be accepted.

Constraints:

- `1 <= k <= points.length <= 104`
- `-104 <= xi, yi <= 104`

Code

```
import heapq

def kClosest(points, k):
        minHeap = []
        for point in points:
                x, y = point
                distance = (x**2) + (y**2)
                minHeap.append([distance,x,y])
                heapq.heapify(minHeap)
        res = []
        while k>0:
                dist, x, y = heapq.heappop(minHeap)
                res.append([x,y])
                k-=1
        return res
if __name__ == '__main__':
        points = [[0,1],[1,0]]
        k = 2
        print(kClosest(points,k))
```

Explanation

```
def kClosest([[0, 1], [1, 0]], 2):
        minHeap = []
        for point in [[0, 1], [1, 0]]:
                x, y = [0, 1] >>                x = 0, y = 1
                distance = (0**2) + (1**2) >> 1
                [].append([1, 0, 1]) >> minHeap = [[1, 0, 1]]
                heapq.heapify([[1, 0, 1]]) >> minHeap = [[1, 0, 1]]
                x, y = [1, 0] >>                x = 1, y = 0
                distance = (1**2) + (0**2) >> 1
                [[1, 0, 1]].append([1, 1, 0]) >> minHeap = [[1, 0, 1], [1, 1, 0]]
                heapq.heapify([[1, 0, 1], [1, 1, 0]]) >> minHeap = [[1, 0, 1], [1, 1, 0]]
        res = []
        while k > 0:
                dist, x, y = heapq.heappop({minHeap}) >> dist = 1, x = 0, y = 1
                res.append([0, 1]) >> res = [[0, 1]]
                k-=1 >> k = 1
                dist, x, y = heapq.heappop({minHeap}) >> dist = 1, x = 1, y = 0
                res.append([1, 0]) >> res = [[0, 1], [1, 0]]
                k-=1 >> k = 0
        return [[0, 1], [1, 0]]
```

4. Leetcode - Kth Largest Element In An Array

Medium

Given an integer array `nums` and an integer `k`, return *the* `kth` *largest element in the array*.

Note that it is the `kth` largest element in the sorted order, not the `kth` distinct element.

Can you solve it without sorting?

Example 1:
Input: nums = [3,2,1,5,6,4], k = 2

Output: 5
Example 2:

Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4

Constraints:

- `1 <= k <= nums.length <= 105`
- `-104 <= nums[i] <= 104`

Code

```
def findKthLargest(nums, k):
        k = len(nums)-k
        def quickSelect(l, r):
                pivot, p = nums[r], l
                for i in range(l, r):
                        if nums[i] <= pivot:
                                nums[p], nums[i] = nums[i], nums[p]
                                p += 1
                nums[p], nums[r] = nums[r], nums[p]
                if p > k: return quickSelect(l, p - 1)
                elif p < k: return quickSelect(p+1, r)
                else: return nums[p]
        return quickSelect(0, len(nums)-1)
print(findKthLargest([3,2,1,5,6,4], 2))
```

Explanation

```
def findKthLargest([3, 2, 1, 5, 6, 4], 2):
        k = 6-2 >> k = 4
        return quickSelect(0, 5):
                pivot, p = 4, 0 >> pivot = 4, p = 0
                for i in range(0, 5):
                        # i = 0
                        if nums[0] <= 4: --> True
                                nums[0], nums[0] = 3, 3 >> nums = [3, 2, 1, 5, 6, 4]
                                p+=1 >> p = 1


                # i = 1
                if nums[1] <= 4: --> True
                        nums[1], nums[1] = 2, 2 >> nums = [3, 2, 1, 5, 6, 4]
                        p+=1 >> p = 2


                # i = 2
                if nums[2] <= 4: --> True
                        nums[2], nums[2] = 1, 1 >> nums = [3, 2, 1, 5, 6, 4]
                        p+=1 >> p = 3


                # i = 3
                if nums[3] <= 4: --> False


                # i = 4
                if nums[4] <= 4: --> False


    nums[3], nums[5] = 4, 5 >> nums = [3, 2, 1, 4, 6, 5]
```

```
                    if 3 > 4: --> False
            elif 3 < 4: --> True
                    return quickSelect(4, 5):
```

```
pivot, p = 5, 4 >> pivot = 5, p = 4
for i in range(4, 5):
        # i = 4
        if nums[4] <= 5: --> False
```

```
nums[4], nums[5] = 5, 6 >> nums = [3, 2, 1, 4, 5, 6]
if 4 > 4: --> False
elif 4 < 4: --> False
else: --> True
        return 5
```

5. Leetcode - Task Scheduler

Medium

Given a characters array `tasks`, representing the tasks a CPU needs to do, where each letter represents a different task. Tasks could be done in any order. Each task is done in one unit of time. For each unit of time, the CPU could complete either one task or just be idle.

However, there is a non-negative integer `n` that represents the cooldown period between two **same tasks** (the same letter in the array), that is that there must be at least `n` units of time between any two same tasks.

Return *the least number of units of times that the CPU will take to finish all the given tasks*.

Example 1:

```
Input: tasks = ["A","A","A","B","B","B"], n = 2
Output: 8
```

Explanation:

A -> B -> idle -> A -> B -> idle -> A -> B
There is at least 2 units of time between any two same tasks.

Example 2:

```
Input: tasks = ["A","A","A","B","B","B"], n = 0
Output: 6
```

Explanation: On this case any permutation of size 6 would work since n = 0.

```
["A","A","A","B","B","B"]
["A","B","A","B","A","B"]
["B","B","B","A","A","A"]
```

...
And so on.

Example 3:

```
Input: tasks = ["A","A","A","A","A","A","B","C","D","E","F","G"], n = 2
Output: 16
```

Explanation:
One possible solution is
A -> B -> C -> A -> D -> E -> A -> F -> G -> A -> idle -> idle -> A -> idle -> idle -> A

Constraints:

- `1 <= task.length <= 104`
- `tasks[i]` is upper-case English letter.
- The integer `n` is in the range `[0, 100]`.

Code

```
import heapq
from collections import Counter, deque
def leastInterval(tasks, n):
        count = Counter(tasks)
        maxHeap = [-cnt for cnt in count.values()]
        heapq.heapify(maxHeap)
        time = 0
        q = deque() #pairs of [-cnt, idleTime]
        while maxHeap or q:
                time += 1
                if maxHeap:
                        cnt = 1+heapq.heappop(maxHeap)
                        if cnt:
                                q.append([cnt, time + n]) #if count is non-zero, we append it to the queue
                if q and q[0][1] == time: #if current time matches the idleTime of the first item in queue, then we push the count
back to our heap.
                        heapq.heappush(maxHeap,q.popleft()[0])
        return time
tasks = ["A","A","A","B","B","B"]
n = 2
print(leastInterval(tasks,n))
```

Explanation

```
def leastInterval(['A', 'A', 'A', 'B', 'B', 'B'], 2):
        count = Counter(['A', 'A', 'A', 'B', 'B', 'B']) >> Counter({'A': 3, 'B': 3})
        maxHeap = [-cnt for cnt in Counter({'A': 3, 'B': 3}).values()] >> [-3, -3]
        heapq.heapify([-3, -3]) >> maxHeap = [-3, -3]
        time = 0
        q = deque()
        while maxHeap or q:
                # maxHeap = [-3, -3], q = deque([]), time = 0
                time += 1 >> time = 1
                if maxHeap: --> True
                        cnt = 1+heapq.heappop([-3, -3]) >> cnt = -2
                        if cnt: --> True
                                q.append([cnt, 1 + 2]) >> q = deque([[-2, 3]])
                if deque([[-2, 3]]) and q[0][1] == 1: -- > False
                # maxHeap = [-3], q = deque([[-2, 3]]), time = 1
                time += 1 >> time = 2
                if maxHeap: --> True
                        cnt = 1+heapq.heappop([-3]) >> cnt = -2
                        if cnt: --> True
                                q.append([cnt, 2 + 2]) >> q = deque([[-2, 3], [-2, 4]])
                if deque([[-2, 3], [-2, 4]]) and q[0][1] == 2: -- > False
                # maxHeap = [], q = deque([[-2, 3], [-2, 4]]), time = 2
                time += 1 >> time = 3
                if maxHeap: --> False
                if deque([[-2, 3], [-2, 4]]) and q[0][1] == 3: -- > True
                        heapq.heappush([],deque([[-2, 3], [-2, 4]]).popleft()[0]) >> maxHeap = [-2], q = deque([[-2, 4]])

                # maxHeap = [-2], q = deque([[-2, 4]]), time = 3
                time += 1 >> time = 4
                if maxHeap: --> True
                        cnt = 1+heapq.heappop([-2]) >> cnt = -1
                        if cnt: --> True
                                q.append([cnt, 4 + 2]) >> q = deque([[-2, 4], [-1, 6]])
                if deque([[-2, 4], [-1, 6]]) and q[0][1] == 4: -- > True
                        heapq.heappush([],deque([[-2, 4], [-1, 6]]).popleft()[0]) >> maxHeap = [-2], q = deque([[-1, 6]])

                # maxHeap = [-2], q = deque([[-1, 6]]), time = 4
                time += 1 >> time = 5
                if maxHeap: --> True
                        cnt = 1+heapq.heappop([-2]) >> cnt = -1
                        if cnt: --> True
                                q.append([cnt, 5 + 2]) >> q = deque([[-1, 6], [-1, 7]])
                if deque([[-1, 6], [-1, 7]]) and q[0][1] == 5: -- > False
                # maxHeap = [], q = deque([[-1, 6], [-1, 7]]), time = 5
                time += 1 >> time = 6
                if maxHeap: --> False
                if deque([[-1, 6], [-1, 7]]) and q[0][1] == 6: -- > True
                        heapq.heappush([],deque([[-1, 6], [-1, 7]]).popleft()[0]) >> maxHeap = [-1], q = deque([[-1, 7]])
```

```
            # maxHeap = [-1], q = deque([[-1, 7]]), time = 6
            time += 1 >> time = 7
            if maxHeap: --> True
                    cnt = 1+heapq.heappop([-1]) >> cnt = 0
                    if cnt: --> False
            if deque([[-1, 7]]) and q[0][1] == 7: -- > True
                    heapq.heappush([],deque([[-1, 7]]).popleft()[0]) >> maxHeap = [-1], q = deque([])

            # maxHeap = [-1], q = deque([]), time = 7
            time += 1 >> time = 8
            if maxHeap: --> True
                    cnt = 1+heapq.heappop([-1]) >> cnt = 0
                    if cnt: --> False
            if deque([]) and q[0][1] == 8: -- > False
    return 8
```

6. Leetcode - Design Twitter

#hashset #hashmap #twitter_design #software_design

Medium

Design a simplified version of Twitter where users can post tweets, follow/unfollow another user, and is able to see the `10` most recent tweets in the user's news feed.

Implement the `Twitter` class:

- `Twitter()` Initializes your twitter object.
- `void postTweet(int userId, int tweetId)` Composes a new tweet with ID `tweetId` by the user `userId`. Each call to this function will be made with a unique `tweetId`.
- `List<Integer> getNewsFeed(int userId)` Retrieves the `10` most recent tweet IDs in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user themself. Tweets must be **ordered from most recent to least recent**.
- `void follow(int followerId, int followeeId)` The user with ID `followerId` started following the user with ID `followeeId`.
- `void unfollow(int followerId, int followeeId)` The user with ID `followerId` started unfollowing the user with ID `followeeId`.

Example 1:

```
Input:
--------------------
["Twitter", "postTweet", "getNewsFeed", "follow", "postTweet", "getNewsFeed", "unfollow", "getNewsFeed"]

[[], [1, 5], [1], [1, 2], [2, 6], [1], [1, 2], [1]]

Output:
--------------------
[null, null, [5], null, null, [6, 5], null, [5]]
```

Explanation

```
Twitter twitter = new Twitter();
twitter.postTweet(1, 5); // User 1 posts a new tweet (id = 5).
twitter.getNewsFeed(1);  // User 1's news feed should return a list with 1 tweet id -> [5]. return [5]
twitter.follow(1, 2);    // User 1 follows user 2.
twitter.postTweet(2, 6); // User 2 posts a new tweet (id = 6).
twitter.getNewsFeed(1);  // User 1's news feed should return a list with 2 tweet ids -> [6, 5]. Tweet id 6 should precede tweet id 5
because it is posted after tweet id 5.
twitter.unfollow(1, 2);  // User 1 unfollows user 2.
twitter.getNewsFeed(1);  // User 1's news feed should return a list with 1 tweet id -> [5], since user 1 is no longer following user
2.
```

Constraints:

- `1 <= userId, followerId, followeeId <= 500`
- `0 <= tweetId <= 104`
- All the tweets have **unique** IDs.
- At most `3 * 104` calls will be made to `postTweet`, `getNewsFeed`, `follow`, and `unfollow`.

Code

```
import heapq
from collections import defaultdict
class Twitter(object):
    def __init__(self):
        self.count = 0
        self.tweetMap = defaultdict(list)
        self.followMap = defaultdict(set)

    def postTweet(self, userId, tweetId):
        self.tweetMap[userId].append([self.count,tweetId])
        self.count -=1
    def getNewsFeed(self, userId):
        res = []
        minHeap = []
        self.followMap[userId].add(userId)
        for followeeId in self.followMap[userId]:
            if followeeId in self.tweetMap:
                idx = len(self.tweetMap[followeeId])-1
                count, tweetId = self.tweetMap[followeeId][idx]
                minHeap.append([count,tweetId, followeeId, idx-1])
        heapq.heapify(minHeap)
        while minHeap and len(res)<10:
            count,tweetId, followeeId, idx = heapq.heappop(minHeap)
            res.append(tweetId)
            if idx>=0:
                count, tweetId = self.tweetMap[followeeId][idx]
                heapq.heappush(minHeap, [count, tweetId, followeeId, idx-1])
        return res

    def follow(self, followerId, followeeId):
        self.followMap[followerId].add(followeeId)

    def unfollow(self, followerId, followeeId):
        if followeeId in self.followMap[followerId]:
            self.followMap[followerId].remove(followeeId)


obj = Twitter()
param1 = ["Twitter", "postTweet", "getNewsFeed", "follow", "postTweet", "getNewsFeed", "unfollow", "getNewsFeed"]

param2 = [[], [1, 5], [1], [1, 2], [2, 6], [1], [1, 2], [1]]
output = []
operations={
                  "Twitter":None,
                  "postTweet": obj.postTweet,
                  "getNewsFeed": obj.getNewsFeed,
                  "follow": obj.follow,
                  "unfollow": obj.unfollow
                  }
for i in range(len(param1)):
        method = operations[param1[i]]
        if method is not None:
           if len(param2[i])>1:
               output.append(method(param2[i][0],param2[i][1]))
           else:
               output.append(method(param2[i][0]))
        else:
                output.append(None)
print(output)
```

7. Leetcode - Find Median From Data Stream

Hard

The **median** is the middle value in an ordered integer list. If the size of the list is even, there is no middle value, and the median is the mean of the two middle values.

- For example, for `arr = [2,3,4]`, the median is `3`.

- For example, for `arr = [2,3]`, the median is `(2 + 3) / 2 = 2.5`.

Implement the MedianFinder class:

- `MedianFinder()` initializes the `MedianFinder` object.
- `void addNum(int num)` adds the integer `num` from the data stream to the data structure.
- `double findMedian()` returns the median of all elements so far. Answers within `10-5` of the actual answer will be accepted.

Example 1:

Input

```
["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]
[[], [1], [2], [], [3], []]
```

Output

```
[null, null, null, 1.5, null, 2.0]
```

Explanation

```
MedianFinder medianFinder = new MedianFinder();
medianFinder.addNum(1);    // arr = [1]
medianFinder.addNum(2);    // arr = [1, 2]
medianFinder.findMedian(); // return 1.5 (i.e., (1 + 2) / 2)
medianFinder.addNum(3);    // arr[1, 2, 3]
medianFinder.findMedian(); // return 2.0
```

Constraints:

- `-105 <= num <= 105`
- There will be at least one element in the data structure before calling `findMedian`.
- At most `5 * 104` calls will be made to `addNum` and `findMedian`.

Follow up:

- If all integer numbers from the stream are in the range `[0, 100]`, how would you optimize your solution?
- If `99%` of all integer numbers from the stream are in the range `[0, 100]`, how would you optimize your solution?

Code

```python
import heapq

class MedianFinder(object):

    def __init__(self):
        self.small, self.large = [], []

    def addNum(self, num):
        if not self.small or num <= -self.small[0]:
            heapq.heappush(self.small, -num)
        else:
            heapq.heappush(self.large, num)

        # Balance the sizes of small and large
        if len(self.small) > len(self.large) + 1:
            val = -heapq.heappop(self.small)
            heapq.heappush(self.large, val)
        elif len(self.large) > len(self.small):
            val = heapq.heappop(self.large)
            heapq.heappush(self.small, -val)

    def findMedian(self):
        if len(self.small) == len(self.large):
```

```
            return (-self.small[0] + self.large[0]) / 2.0
        else:
            return -self.small[0] / 1.0

# Input and method calls
commands = ["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]
values = [[], [1], [2], [], [3], []]

median_finder = None
output = []

for i in range(len(commands)):
    command = commands[i]
    value = values[i][0] if values[i] else None

    if command == "MedianFinder":
        median_finder = MedianFinder()
        output.append(None)
    elif command == "addNum":
        median_finder.addNum(value)
        output.append(None)
    elif command == "findMedian":
        result = median_finder.findMedian()
        output.append(result)

print(output)  # Output will contain the results of method calls
```

End