Module7-HW

June 20, 2024

Problem1-LeetcodeQ 703-Kth Largest Element in a 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.

0.0.1 Q703 Psuedocode

[ ]:
```
initialize:
    - k as integer
    - heap as empty list
    For each num in nums:
        Push num into heap
```

```
        If size of heap is greater than k:
            Pop smallest element from heap

function add(val):
    Push val into heap
    If size of heap is greater than k:
        Pop smallest element from heap
    Return smallest element from heap (which is heap[0])
```

0.0.2 Q703 Code.py

[4]:
```
from typing import List
import heapq

class KthLargest:
    def __init__(self, k: int, nums: List[int]):
        self.heap = []
        self.k = k
        for num in nums:
            heapq.heappush(self.heap, num)
            if len(self.heap) > k:
                heapq.heappop(self.heap)

    def add(self, val: int) -> int:
        heapq.heappush(self.heap, val)
        if len(self.heap) > self.k:
            heapq.heappop(self.heap)
        return self.heap[0]


kth_largest = KthLargest(3, [4, 5, 8, 2])
print(kth_largest.add(3))  # Output: 4
print(kth_largest.add(5))  # Output: 5
print(kth_largest.add(10))  # Output: 5
print(kth_largest.add(9))  # Output: 8
print(kth_largest.add(4))  # Output: 8
```

```
4
5
5
8
8
```

Problem2-LeetcodeQ 1046- 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

0.0.3 Q1046 Psuedocode

[ ]:
```
function last_stone_weight(stones):
Create a max heap of negative values of stone heap = []
for stone in stones:
    heap.push(-stone)

Convert the list into a heap in-place heapify(heap)

While there are more than 1 stone left in the heap

Extract the two largest stones (in negative form)
If they are not equal, smash them and push the difference back to the heap
If there is one stone left, return its weight (convert back from negative)
If no stones are left, return 0
```

0.0.4 Q1046 Code.py

[9]:
```
import heapq

class Solution:
    def last_stone_weight(self, stones):
```

```
        heap = [-stone for stone in stones]
        heapq.heapify(heap)

        while len(heap) > 1:
            x = heapq.heappop(heap)
            y = heapq.heappop(heap)

            if x != y:
                heapq.heappush(heap, x - y)

        if heap:
            return -heap[0]
        else:
            return 0

solution = Solution()
stones = [2, 7, 4, 1, 8, 1]
print(solution.last_stone_weight(stones))
```

```
1
```

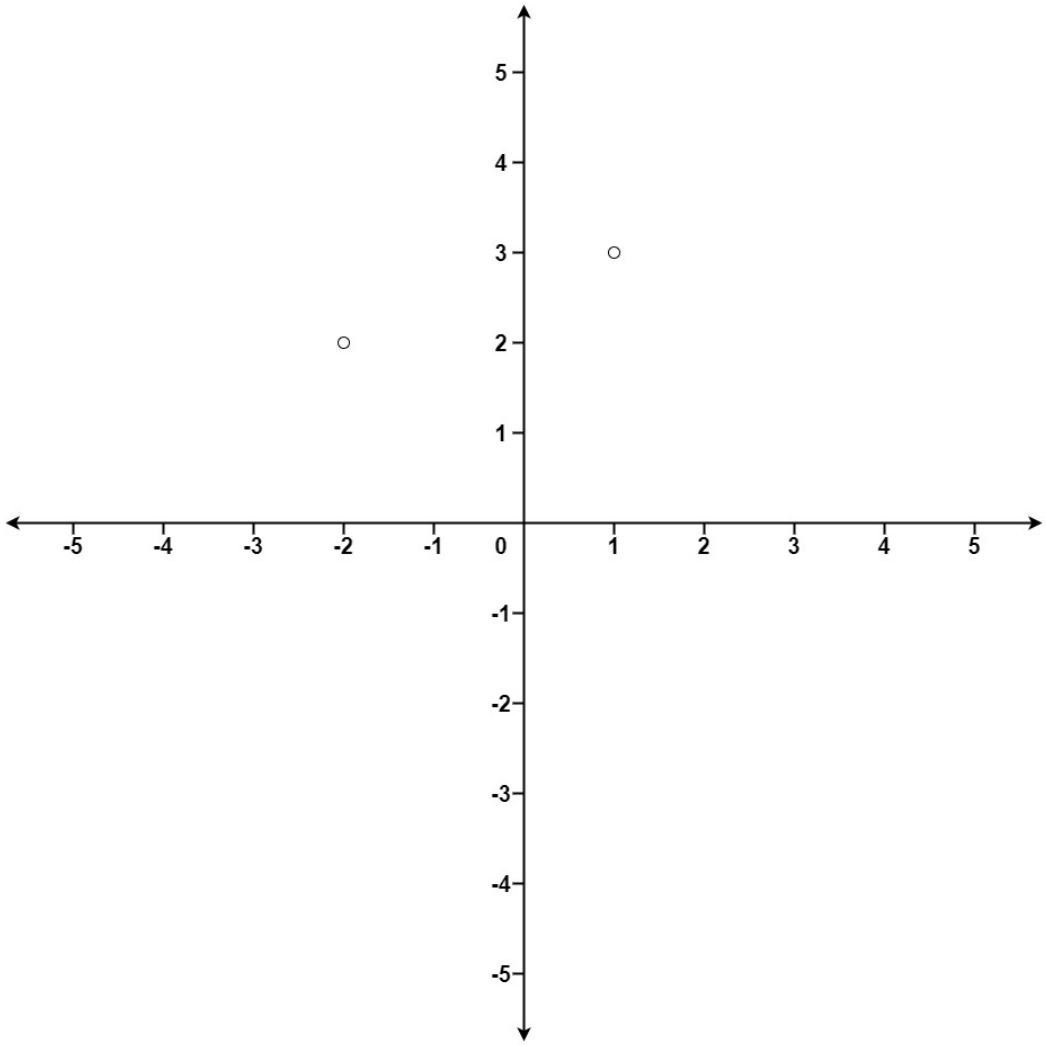
[ ]:

Problem3-LeetcodeQ 973. 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.

[ ]:

0.0.5 Q973 Psuedocode

[ ]:
```
for point in points:
Calculate squared distance from origin
    Push point with negative distance to maintain min heap
Replace the farthest point if current point is closer
Extract points from heap and return as list
```

0.0.6 Q973 Code.py

[7]:
```
import heapq
class Solution:
    def kClosest(self, points, k):
        heap = []
        for point in points:
            distance = sum(map(lambda x: x ** 2, point))
            if len(heap) < k:
                heapq.heappush(heap, [-distance, point])
            else:
                if -distance > heap[0][0]:
                    heapq.heappop(heap)
                    heapq.heappush(heap, [-distance, point])
        return [point for distance, point in heap]


solution = Solution()
points = [[1, 3], [-2, 2]]
k = 1
print(solution.kClosest(points, k))
```

```
[[-2, 2]]
```

Problem4-LeetcodeQ 215. 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

0.0.7 Q215 Psuedocode

[ ]:
```
define function maxHeapify with parameters arr (list of integers), i (integer), end (integer)
set j to 2 * i + 1
while j is less than or equal to end
    if j + 1 is less than or equal to end and arr[j + 1] is greater than arr[j]
        increment j by 1
    if arr[i] is less than arr[j]
        swap arr[i] and arr[j]
        set i to j
        set j to 2 * i + 1
    else
        break

set n to the length of nums
for i from n // 2 - 1 down to 0 (inclusive)
    call maxHeapify with nums, i, and n - 1

for j from n - 1 down to n - k (inclusive)
    swap nums[0] and nums[j]
    call maxHeapify with nums, 0, and j - 1

return nums[-k]
```

0.0.8 Q215 Code.py

[8]:
```python
from typing import List

class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        def maxHepify(arr, i, end):
            j = 2 * i + 1
            while j <= end:
                if j + 1 <= end and arr[j + 1] > arr[j]:
                    j += 1
                if arr[i] < arr[j]:
                    arr[i], arr[j] = arr[j], arr[i]
                    i = j
                    j = 2 * i + 1
                else:
                    break

        n = len(nums)
        for i in range(n // 2 - 1, -1, -1):
            maxHepify(nums, i, n - 1)
```

```
        for j in range(n - 1, n - k - 1, -1):
            nums[0], nums[j] = nums[j], nums[0]
            maxHepify(nums, 0, j - 1)

        return nums[-k]


nums = [3, 2, 1, 5, 6, 4]
k = 2
solution = Solution()
print(solution.findKthLargest(nums, k))  # Output should be 5
```

```
5
```

[ ]: