Heap

Mustafa Muhammad

12th October 2021

List of Questions:

1. General format
2. Kth smallest element
3. Return K largest elements in array
4. Sort a K sorted array—Sort nearly sorted array
5. K closest numbers
6. Top K frequent elements
7. Frequency sort
8. Connect ropes to minimize the cost
9. Sum of elements between k1 smallest and k2 smallest numbers

1 General format and identification

Identification 1, K will be mentioned, or any other limited range of numbers.

2, Smallest or largest.

1, Min Heap maps to k + Largest

2, Max Heap maps to k + Smallest

Sorting is nlog(n), using heap for K elements is nlog(K)

2 Kth Smallest Element

Default heap is a min heap in python, to make it max just multiply by -1.

```
import heapq
class Solution:
    def k_smallest_element(self, arr, k):
        heap = []

        for i in range(0, len(arr)):
            heapq.heappush(heap, -arr[i])

            if len(heap) > k:
                heapq.heappop(heap)

        #Max heap so the smallest will be at the start of the array.
        return heap[0]*-1
```

3 Return K largest elements in an array

```
import heapq
class Solution:
    def k_largest_element(self, arr, k):
        heap = []

        for i in range(0, len(arr)):
            heapq.heappush(heap, arr[i])

            if len(heap) > k:
                heapq.heappop(heap)

        #Max heap so the smallest will be at the start of the array.
        return heap
```

4 Sort a K sorted array — nearly sorted array

Given an array of n elements, where each element is at most k away from its target position, devise an algorithm that sorts in O(n log k) time. For example, let us consider k is 2, an element at index 7 in the sorted array, can be at indexes 5, 6, 7, 8, 9 in the given array.

We sort K values at a time, hence when we go above k length of heap, we pop and add to the result array. We proceed to add all the remaining values at the end.

```
def nearly_sorted_array(array, k):
    heap = []
    res = []
    for i in range(0, len(arr)):
        heapq.heappush(heap, arr[i])

        if len(heap) > k:
            res.append(heapq.pop(heap))

    for j in range(0, len(heap)):
        res.append(heapq.heapop(heap))

    return res
```

5 K closest numbers — K closest elements

Given a sorted integer array arr, two integers k and x, return the k closest integers to x in the array. The result should also be sorted in ascending order. An integer a is closer to x than an integer b if:

|a - x| < |b - x|, or

|a - x| == |b - x| and a < b

We put -arr[i] because the second key is compared if the difference is same.

```
import heapq

class Solution:
    def findClosestElements(self, arr: List[int], k: int, x: int) -> List[int]:
        heap = []
        for i in range(0, len(arr)):
            diff = abs(arr[i]-x)

            heapq.heappush(heap, (-diff, -arr[i]))

            if len(heap) > k:
                heapq.heappop(heap)

        return sorted([h[1]*-1 for h in heap])
```

6 Top K frequent numbers/elements

Given an integer array nums and an integer k, return the k most frequent elements. You may return the answer in any order.

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

```
import heapq
class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        hash_map = {}
        for i in range(0, len(nums)):
            if nums[i] in hash_map:
                hash_map[nums[i]] += 1
            else:
                hash_map[nums[i]] = 1
        heap = []
        for keys, vals in hash_map.items():
            heapq.heappush(heap, (vals, keys))
            if len(heap) > k:
                heapq.heappop(heap)
        return [h[1] for h in heap]
```

7 Frequency sort

Sort on array by the frequency of its elements

```
import heapq
class Solution:
    def frequencySort(self, nums: List[int]) -> List[int]:
        hash_map = {}
        heap = []
        res = []

        for i in range(0, len(nums)):
            if nums[i] not in hash_map:
                hash_map[nums[i]] = 1
            else:
                hash_map[nums[i]] += 1
        for key, val in hash_map.items():
            heapq.heappush(heap, (val, -key))

        for j in range(0, len(heap)):
            tup = heapq.heappop(heap)
            count = tup[0]
            key = tup[1]*-1
            for j in range(0, count):
                res.append(key)
        return res
```

8 Find K closest points to origin

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). You may return the answer in any order. The answer is guaranteed to be unique

```
import heapq
class Solution:
    def kClosest(self, points: List[List[int]], k: int) -> List[List[int]]:
        heap = []
        for coord in points:
            x = coord[0]
            y = coord[1]
            dist = (x**2+y**2)**0.5
            heapq.heappush(heap, (-dist, (x,y)))
            if len(heap) > k:
                heapq.heappop(heap)
        return [h[1] for h in heap]
```

9 Connect ropes to minimize cost

There are given N ropes of different lengths, we need to connect these ropes into one rope. The cost to connect two ropes is equal to sum of their lengths. The task is to connect the ropes with minimum cost. Input: n = 4 arr[] = 4, 3, 2, 6 Output: 29 Explanation: For example if we are given 4 ropes of lengths 4, 3, 2 and 6. We can connect the ropes in following ways. 1) First connect ropes of lengths 2 and 3. Now we have three ropes of lengths 4, 6 and 5. 2) Now connect ropes of lengths 4 and 5. Now we have two ropes of lengths 6 and 9. 3) Finally connect the two ropes and all ropes have connected. Total cost for connecting all ropes is 5 + 9 + 15 = 29. This is the optimized cost for connecting ropes. Other ways of connecting ropes would always have same or more cost. For example, if we connect 4 and 6 first (we get three strings of 3, 2 and 10), then connect 10 and 3 (we get two strings of 13 and 2). Finally we connect 13 and 2. Total cost in this way is 10 + 13 + 15 = 38.

```
import heapq
class Solution:
    def minCost(self,arr,n) :
        heap = []
        total = 0
        for i in range(0, len(arr)):
            heapq.heappush(heap, arr[i])
        while len(heap) >= 2:
            val1 = heapq.heappop(heap)
            val2 = heapq.heappop(heap)
            total += val1+val2
            heapq.heappush(heap, val1+val2)
        return total
```

10 Sum of Elements b/w k1 and k2 smallest numbers

Given an array A[] of N positive integers and two positive integers K1 and K2. Find the sum of all elements between K1th and K2th smallest elements of the array. It may be assumed that (1 ¡= k1 ¡ k2 ¡= n). Input: N = 7 A[] = 20, 8, 22, 4, 12, 10, 14 K1 = 3, K2 = 6 Output: 26 Explanation: 3rd smallest element is 10 6th smallest element is 20 Sum of all element between K1 & K2 is 12 + 14 = 26

```
import heapq
class Solution:
    def sumBetweenTwoKth(self, A, N, K1, K2):
        def get_kth(arr, k):
            heap = []
            for i in range(0, len(arr)):
                heapq.heappush(heap, -arr[i])
                if len(heap) > k:
                    heapq.heappop(heap)
            return heap[0]
        v1 = get_kth(A, K1)*-1
        v2 = get_kth(A, K2)*-1
        total = 0
        for j in range(0, len(A)):
            if A[j] > v1 and A[j] < v2:
                total += A[j]
        return total
```