Priority Queue

Assignment Solutions

PW SKILLS

Q1. Given a string s, rearrange the characters of s so that any two adjacent characters are not the same.

Return any possible rearrangement of s or return "" if not possible.

Example 1:

Input: s = "aab"
Output: "aba"
Example 2:

Input: s = "aaab"
Output: ""

Solution :

Code : ASS_Code1.java

Output :

```
The desired output is : laslpaslasliphiphcykwaWls
```

Approach :
1. Idea is to store all the characters and their occurrences in a hashmap
2. Sort the map with the help of a max heap by moving all the duplicates to the left side.
3. Initialize a Map.Entry variable to keep track of the previous entry in the priority queue
4. Initialize a string builder to add back the characters in the expected order
5. Run a while loop over the priority queue and poll the element from the priority queue
6. We check if the previous entry is not null and its check whether its value
(the number of occurrences of this character) is greater than 0 and add it to the queue otherwise we add the current entry's key to the string builder and decrement its occurrence by 1.
7. Update the previous entry with current entry (as mentioned above, we could have multiple occurrences of a character in the map, so we want to keep a track of the current entry by assigning it to the previous entry and handle the condition to add this element to the queue or not as pointed out in the previous step.
8. At this point, once the loop is rolled out, we check if the string builder's length is equal to the length of the input string and return the string builder as string otherwise we return "".

Q2. You are given two integer arrays nums1 and nums2 sorted in ascending order and an integer k.
Define a pair (u, v) which consists of one element from the first array and one element from the second array.
Return the k pairs (u1, v1), (u2, v2), ..., (uk, vk) with the smallest sums.

Example 1:

Input: nums1 = [1,7,11], nums2 = [2,4,6], k = 3
Output: [[1,2],[1,4],[1,6]]
Explanation: The first 3 pairs are returned from the sequence: [1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
Example 2:

Input: nums1 = [1,1,2], nums2 = [1,2,3], k = 2
Output: [[1,1],[1,1]]

Explanation: The first 2 pairs are returned from the sequence: [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
Example 3:

Input: nums1 = [1,2], nums2 = [3], k = 3
Output: [[1,3],[2,3]]
Explanation: All possible pairs are returned from the sequence: [1,3],[2,3]

Solution :

Code : ASS_Code2.java

Output :

```
The desired output is :
1 4
2 4
1 5
2 5
```

Approach :

- Use min_heap to keep track of the next minimum pair sum, and we only need to maintain K possible candidates in the data structure.

- For every number in nums1, its best partner(yields min sum) always starts from nums2[0] since arrays are all sorted.
- And for a specific number in nums1, its next candidate should be [this specific number] + nums2[current_associated_index + 1], unless out of boundary.

- Here is a simple example demonstrating how this algorithm works.

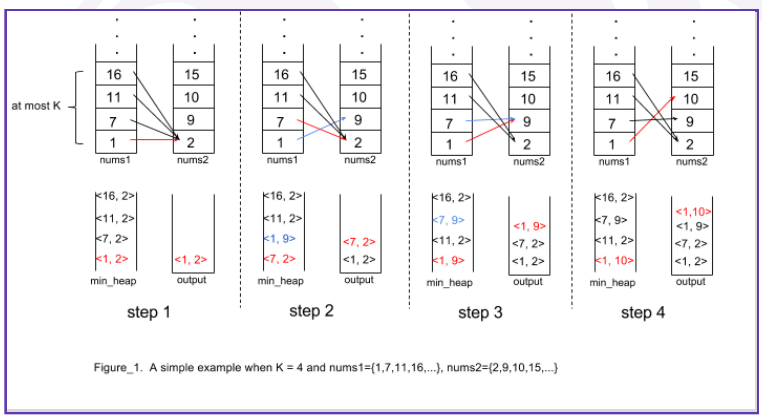

Figure_1. A simple example when K = 4 and nums1={1,7,11,16,...}, nums2={2,9,10,15,...}

Q3. You are playing a solitaire game with three piles of stones of sizes a, b, and c respectively. Each turn you choose two different non-empty piles, take one stone from each, and add 1 point to your score. The game stops when there are fewer than two non-empty piles (meaning there are no more available moves).

Given three integers a, b, and c, return the maximum score you can get.

Example 1:

Input: a = 2, b = 4, c = 6
Output: 6
Explanation: The starting state is (2, 4, 6). One optimal set of moves is:
- Take from 1st and 3rd piles, state is now (1, 4, 5)

Take from 1st and 3rd piles, state is now (0, 4, 4)
- Take from 2nd and 3rd piles, state is now (0, 3, 3)
- Take from 2nd and 3rd piles, state is now (0, 2, 2)
- Take from 2nd and 3rd piles, state is now (0, 1, 1)
- Take from 2nd and 3rd piles, state is now (0, 0, 0)

There are fewer than two non-empty piles, so the game ends. Total: 6 points.
Example 2:

Input: a = 4, b = 4, c = 6
Output: 7
Explanation: The starting state is (4, 4, 6). One optimal set of moves is:
- Take from 1st and 2nd piles, state is now (3, 3, 6)
- Take from 1st and 3rd piles, state is now (2, 3, 5)
- Take from 1st and 3rd piles, state is now (1, 3, 4)
- Take from 1st and 3rd piles, state is now (0, 3, 3)
- Take from 2nd and 3rd piles, state is now (0, 2, 2)
- Take from 2nd and 3rd piles, state is now (0, 1, 1)
- Take from 2nd and 3rd piles, state is now (0, 0, 0)

There are fewer than two non-empty piles, so the game ends. Total: 7 points.

Solution :

Code : ASS_Code3.java

Output :

```
The desired output is : 8
```

Approach :
- The idea is to get the two stones with maximum value and decrease the two stones value by 1.
- We do this until we don't find two non zero valued stones.

Q4. You are given an m x n matrix that has its rows sorted in non-decreasing order and an integer k.

You are allowed to choose exactly one element from each row to form an array.
Return the kth smallest array sum among all possible arrays.

Example 1:

Input: mat = [[1,3,11],[2,4,6]], k = 5
Output: 7
Explanation: Choosing one element from each row, the first k smallest sum are:
[1,2], [1,4], [3,2], [3,4], [1,6]. Where the 5th sum is 7.
Example 2:

Input: mat = [[1,3,11],[2,4,6]], k = 9
Output: 17
Example 3:

Input: mat = [[1,10,10],[1,4,5],[2,3,6]], k = 7
Output: 9
Explanation: Choosing one element from each row, the first k smallest sum are:
[1,1,2], [1,1,3], [1,4,2], [1,4,3], [1,1,6], [1,5,2], [1,5,3]. Where the 7th sum is 9.

Solution :
Code : ASS_Code4.java

Output :

```
The desired output is : 7
```

Approach :

- Calculate max priority queue of size k for the first row.
- Add the rest rows one by one to the max priority queue and make sure that max priority queue size is less than or equal to k.

Q5. Given that integers are read from a data stream. Find the median of elements read so for in an efficient way. For simplicity assume, there are no duplicates. For example, let us consider the streams 5, 15, 1, 3 ...

After reading 1st element of stream - 5 -> median - 5
After reading 2nd element of stream - 5, 15 -> median - 10
After reading 3rd element of stream - 5, 15, 1 -> median - 5
After reading the 4th element of stream - 5, 15, 1, 3 -> median - 4, so on.

Solution :
Code : ASS_Code5.java

Output :

```
The median is : 3.0
The median is : 3.0
The median is : 4.0
```

Approach :

- The basic idea is to maintain two heaps: a max-heap and a min-heap. The max heap stores the smaller half of all numbers while the min heap stores the larger half.
- The sizes of two heaps need to be balanced each time when a new number is inserted so that their size will not be different by more than 1.
- Therefore each time when findMedian() is called we check if two heaps have the same size.
- If they do, we should return the average of the two top values of heaps. Otherwise we return the top of the heap which has one more element.
- There are several possible situations when a new number is inserted:

1)If both heaps are empty, meaning that we are inserting the first number, we just arbitrarily inserted it into a heap, let's say, the min-heap.

2)If the min-heap has more elements (later we will argue that the size won't be different by more than 1), we need to compare the new number with the top of the min-heap. If it is larger than that, then the new number belongs to the larger half and it should be added to the min-heap. But since we have to balance the heap,

we should move the top element of the min-heap to the max-heap. For the min-heap, we inserted a new number but removed the original top, its size won't change. For the max-heap, we inserted a new element (the top of the min-heap) so its size will increase by 1.

3)If max-heap has more elements, we did the similar thing as 2).

4)If they have the same size, we just compare the new number with one of the top to determine which heap the new number should be inserted. We just simply inserted it there.

- It can be seen that for each insertion if it was in situation 1) and 4), then after insertion the heap size difference will be 1. For 2) and 3), the size of the heap with fewer elements will increase by 1 to catch up with the heap with more elements. Hence their sizes are well-balanced and the difference will never exceed 1.

- Obviously, the median will be the top element of the heap which has one more element (if max-heap and min-heap have different sizes), or the average of the two tops (if max-heap and min-heap have equal sizes).