

Top 'K' Frequent Numbers (medium)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity

Problem Statement

Given an unsorted array of numbers, find the top 'K' frequently occurring numbers in it.

Example 1:





Input: [1, 3, 5, 12, 11, 12, 11], K = 2
Output: [12, 11]
Explanation: Both '11' and '12' appeared twice.

Example 2:

Input: [5, 12, 11, 3, 11], K = 2
Output: [11, 5] or [11, 12] or [11, 3]
Explanation: Only '11' appeared twice, all other numbers appeared once.



Try it yourself





Try solving this question here:

 Java	 Python3	 JS	 C++
--	---	--	---

```
1 def find_k_frequent_numbers(nums, k):
2     topNumbers = []
3     # TODO: Write your code here
4     return topNumbers
5
6
7 def main():
8
9     print("Here are the K frequent numbers: " +
10         str(find_k_frequent_numbers([1, 3, 5, 12, 11, 12, 11], 2)))
11
12     print("Here are the K frequent numbers: " +
13         str(find_k_frequent_numbers([5, 12, 11, 3, 11], 2)))
14
15
16 main()
```

1/
18

Solution





This problem follows Top 'K' Numbers

(<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5728885882748928/>). The only difference is that in this problem, we need to find the most frequently occurring number compared to finding the largest numbers.

We can follow the same approach as discussed in the **Top K Elements** problem. However, in this problem, we first need to know the frequency of each number, for which we can use a **HashMap**. Once we have the frequency map, we can use a **Min Heap** to find the 'K' most frequently occurring number. In the **Min Heap**, instead of comparing numbers we will compare their frequencies in order to get frequently occurring numbers

Code


Here is what our algorithm will look like:



 Java
 Python3
 C++
 JS

```

1  from heapq import *
2
3
4  def find_k_frequent_numbers(nums, k):
5
6      # find the frequency of each number
7      numFrequencyMap = {}
8      for num in nums:
9          numFrequencyMap[num] = numFrequencyMap.get(num, 0) + 1
10
11     minHeap = []
12
13     # go through all numbers of the numFrequencyMap and push them in the minHeap, which will
14     # top k frequent numbers. If the heap size is more than k, we remove the smallest(top) n
15     for num, frequency in numFrequencyMap.items():
16         heappush(minHeap, (frequency, num))
17         if len(minHeap) > k:
18             heappop(minHeap)
19
20     # create a list of top k numbers
21     topNumbers = []
22     while minHeap:
23         topNumbers.append(heappop(minHeap)[1])
24
25     return topNumbers
26
27
28 def main():

```



Time complexity

The time complexity of the above algorithm is $O(N + N * \log K)$.



The space complexity will be $O(N)$. Even though we are storing only 'K' numbers in the heap. For the frequency map, however, we need to store all the 'N' numbers.

[← Back](#)[Next →](#)

Connect Ropes (easy)

Frequency Sort (medium)

☒ **Mark as Completed**[Report an Issue](#)[Ask a Question](#)

(https://discuss.educative.io/tag/top-k-frequent-numbers-medium__pattern-top-k-elements__grokking-the-coding-interview-patterns-for-coding-questions)