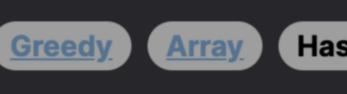
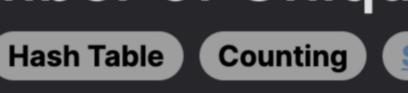
1481. Least Number of Unique Integers after K Removals



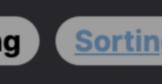


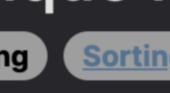


element's occurrences from the array.

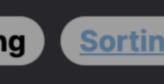














Leetcode Link

Problem Description

Given an array arr of integers and an integer k, the task is to determine the minimum number of unique integers that remain in the array after precisely k elements have been removed. The key to solving this problem lies in effectively selecting which elements to remove to achieve the least number of unique integers possible.

Intuition

To solve this problem, we should consider removing elements that appear more frequently first since this will not reduce the unique count as quickly. Ideally, we want to remove the numbers that occur the least amount of times last. We can apply a strategy that consists of the following steps:

2. Sort these unique integers by their frequency in ascending order. This way, the elements that appear less frequently are at the

1. Count the frequency of each unique integer in the array using a hash table (or Counter in Python).

- beginning of the sorted list. 3. Traverse the sorted list, and with each step, decrease k by the frequency of the current element. This simulates removing that
- 4. If k becomes negative, it means we can't remove all occurrences of the current element without exceeding the allowed k removals. Therefore, the current count of unique integers minus the number of elements we've been able to fully remove by this
- point gives us the answer. 5. If we go through the entire list without k becoming negative, it means we managed to remove all occurrences of certain elements, and we end up with 0 unique integers.
- Having this strategy in place allows us to reach the solution in an efficient manner.

The implementation of the solution follows a straightforward approach outlined in previous steps, which uses common data

integers with fewer occurrences are considered first when we start removing elements.

structures and algorithms: 1. Hash Table (Counter): We employ a hash table which is native to Python called Counter from the collections module. This

Solution Approach

structure automatically counts the frequency of each unique integer, which is essential to our strategy. It simplifies the process of determining how many times each integer appears in the array.

- 1 cnt = Counter(arr) 2. Sorting: After counting the occurrences of each integer, we sort these counts. This sorting is ascending, meaning that unique
- 1 sorted(cnt.values())
- 3. Traversal and Subtraction: With the sorted frequencies, we traverse the list. For each frequency value v, we subtract k by v, simulating the removal of v occurrences from the array.

1 for i, v in enumerate(sorted(cnt.values())):

total number of unique integers (the length of cnt):

remove all occurrences of certain elements leading to zero unique integers left.

1. Count Frequency: We first count how many times each integer appears in the array:

1 if k < 0: return len(cnt) - i 4. Return Result: If we are able to traverse the entire sorted list of frequencies without k becoming negative, we have managed to

If k becomes less than zero during the iteration, it indicates that we cannot remove all occurrences of the current element as

it would exceed k. Hence, the minimal number of unique integers is obtained by subtracting the current index i from the

```
1 return 0
```

linear traversals and basic arithmetic operations.

Let's use a small example to illustrate the solution approach:

Suppose arr = [4, 3, 1, 1, 2, 3, 3] and k = 3. Our goal is to remove k elements from arr in such a way that the number of

The overall complexity of the solution is determined mainly by the sorting operation and the counting operation, with the rest being

unique integers remaining is minimal.

Using a hash table:

to 2.

decreases k to 1.

2 sorted_counts = [1, 1, 2, 3]

4 for i, v in enumerate(sorted_counts):

3 are still present).

k -= v

if k < 0:

3 k = 3

Example Walkthrough

1 appears 2 times 2 appears 1 time

3 appears 3 times 4 appears 1 time

```
2. Sort by Frequency: We sort these counts in ascending order based on frequency:
  Sorted counts: [1, 1, 2, 3]
  The number of unique integers is initially 4.
3. Traverse and Remove: We then traverse the sorted list and remove k elements.
    • We start with the first element (frequency of 1). We remove one instance of the integer with a count of 1, which decreases k
```

cnt = $\{1: 2, 2: 1, 3: 3, 4: 1\}$

∘ With the third element (frequency of 2), we can remove both instances of the integer 1, which would decrease k to -1. However, since we can't go negative, it means we can't remove all instances of 1. At this point, we have removed two unique

def findLeastNumOfUniqueInts(self, arr: List[int], k: int) -> int:

Reduce the count of deletable elements by the current count value

If k becomes negative, we can't delete anymore unique integers

Return the count of remaining unique integers

If k is not negative after trying to remove all, return 0

because all elements can be removed to achieve k deletion

Create a counter for all elements in the array

Sort the counts of each unique integer

return len(counter) - index

Here's the traversal in action: 1 cnt = {1: 2, 2: 1, 3: 3, 4: 1}

integers (those with the initial frequency of 1), resulting in a remaining unique integer count of 2 (those with frequencies of 2 and

• We move to the next element (another frequency of 1). We remove one instance of another integer with a count of 1, which

- return 4 i # number of unique integers initially minus the index 4. Return Result: With the loop broken, we know we've removed instances of unique integers only until the array's k becomes negative. This means we return 4 - 2, which equals 2.
 - sorted_counts = sorted(counter.values()) # Go through the counts starting from the smallest for index, value in enumerate(sorted_counts):

Therefore, the minimum number of unique integers remaining in the array arr after removing exactly k elements is 2.

```
1 from collections import Counter
  from typing import List
  class Solution:
```

counter = Counter(arr)

k -= value

if k < 0:

return 0

Python Solution

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

24

26

25 }

```
25
Java Solution
   class Solution {
       public int findLeastNumOfUniqueInts(int[] arr, int k) {
           // Create a hashmap to store the frequency of each integer in the array
           Map<Integer, Integer> frequencyMap = new HashMap<>();
           // Populate the frequency map
           for (int num : arr) {
               frequencyMap.merge(num, 1, Integer::sum); // Increment the count for each occurrence of a number
           // Create a list to store the frequencies only
9
           List<Integer> frequencies = new ArrayList<>(frequencyMap.values());
10
           // Sort the frequencies in ascending order
11
12
           Collections.sort(frequencies);
13
           // Iterate over the list of frequencies
           for (int i = 0, totalUniqueNumbers = frequencies.size(); i < totalUniqueNumbers; ++i) {</pre>
14
15
               k -= frequencies.get(i); // Subtract the frequency from 'k'
               if (k < 0) {
17
                   // If 'k' becomes negative, the current frequency can't be fully removed
                   // so return the number of remaining unique integers
19
                   return totalUniqueNumbers - i;
20
21
           // If all frequencies have been removed with 'k' operations, return 0 as there are no unique integers left
23
           return 0;
```

public:

class Solution {

C++ Solution

1 #include <vector>

2 #include <unordered_map>

#include <algorithm>

```
int findLeastNumOfUniqueInts(vector<int>& arr, int k) {
           // Create a hashmap to count the occurrence of each integer in the array
           unordered_map<int, int> frequencyMap;
            for (int number : arr) {
10
                ++frequencyMap[number];
12
13
           // Extract the frequencies and sort them in ascending order
14
            vector<int> frequencies;
15
            for (auto& [number, count] : frequencyMap) {
16
                frequencies.push_back(count);
17
19
            sort(frequencies.begin(), frequencies.end());
20
21
           // Determine the least number of unique integers by removing k occurrences
            int uniqueIntegers = frequencies.size(); // start with all unique integers
22
23
            for (int i = 0; i < frequencies.size(); ++i) {</pre>
                // Subtract the frequency of the current number from k
25
                k -= frequencies[i];
26
27
               // If k becomes negative, we can't remove any more numbers
28
               if (k < 0) {
29
                    return uniqueIntegers - i; // Return the remaining number of unique integers
30
31
32
33
           // If k is non-negative after all removals, we've removed all duplicates
34
           return 0;
35
36 };
37
Typescript Solution
```

```
// Extract the frequency values from the map and store them in an array
       const frequencies: number[] = [];
10
       for (const frequency of frequencyMap.values()) {
11
12
           frequencies.push(frequency);
13
14
       // Sort the frequencies array in ascending order
15
       frequencies.sort((a, b) => a - b);
16
17
18
       // Iterate over the sorted frequencies
       for (let i = 0; i < frequencies.length; ++i) {</pre>
19
20
           // Decrement k by the current frequency
           k -= frequencies[i];
           // If k becomes negative, we've used up k removals, so we return
           // the number of unique integers left, which is the length of the
24
           // frequencies array minus the current index
25
               return frequencies.length - i;
26
27
28
29
       // If we've processed all frequencies and haven't used up k removals,
30
31
       // all integers have been removed and 0 unique integers are left
32
       return 0;
33 }
34
Time and Space Complexity
```

function findLeastNumOfUniqueInts(arr: number[], k: number): number {

const frequencyMap: Map<number, number> = new Map();

for (const number of arr) {

6

9

and worst case.

// Iterate over the array and populate the frequency map

// Create a map to hold the frequency of each integer in the array

frequencyMap.set(number, (frequencyMap.get(number) || 0) + 1);

The time complexity of the given code is $0(n \log n)$. This complexity arises from the sorting operation sorted(cnt.values()) where cnt.values() represents the counts of unique integers in the array arr; sorting these counts requires 0(n log n) time since sorting

is typically done using comparison-based algorithms like quicksort or mergesort which have 0(n log n) complexity in the average

The space complexity of the code is O(n) because we are storing counts of the elements in the array in a dictionary cnt. In the worst case, if all elements are unique, it will contain n key-value pairs which relate directly to the size of the input array arr.