LC 697

- There is one loop over the input array. We need to count the frequency and the first occurrence index for each elem. in the input array.

- IF $a \in A$ and is the max freq.,
  let degree = count[a] and
  res = i - first[A[i]] + 1

- IF $a \in A$ is a number w/ max freq.,
  let res = min(res, i - first[A[i]] + 1)

- O(N), O(M) where M is size of diff. numbers.

```
def findShortestSubArray(self, A):
    first, count, res, degree = {}, {}, 0, 0
    for i, a in enumerate(A):
        first.setdefault(a, i)
        count[a] = count.get(a, 0) + 1
        if count[a] > degree:
            degree = count[a]
            res = i - first[a] + 1
        elif count[a] == degree:
            res = min(res, i - first[a] + 1)

    return res.
```

// nums

if a doesn't exist in the set, then set a to i.

if a ∉ count, then count[a] = 0. else count = count + 1

Probably could use namedTuples for the dictionaries.

LC 697

(697) 7/12

Input: list[int] nums where nums[i] $\in \mathbb{Z}^+ \forall i$

Output: int

Return the length of subarray w/ the smallest contiguous same degree as nums.

Ex1 nums = [1, 2, 2, 3, 1]

deg(nums) = 2

smallest = [2, 2] $\Rightarrow$ length = 2 ←

Ex2 nums = [1, 2, 2, 3, 1, 4, 2]

deg(nums) = 3 ←

sm = [2, 2, 3, 1, 4, 2], length = 6 ←

Must be contiguous.

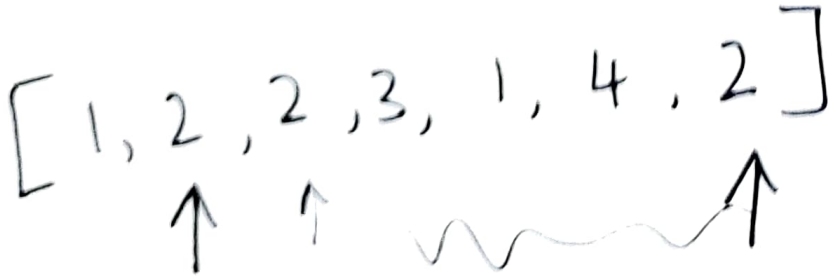
First Naive approach is to generate all subarrays that are contiguous and return the smallest.

↳ O(n) space

↳ Before appending to new list we need to find the position of the repeating value.

We can also use hash map to keep track of the count for each elem.

- The better approach would be to use ~~two~~ pointers.
  ↳ where there are ptrs on the start, curr, and end of the max freq. value.



→ O(n), O(1)

LC 1560

7/12/22

```
import bisect

def f(self, n:int, rounds:List[int])->List[int]:
    first, last = rounds[0], rounds[-1]

    if last >= first:
        return [i for i in range(first, last+1)]

    else:
        rec = []
        for i in range(n):
            if first >= i+1 or last <= i+1:
                if rec:
                    bisect.insort(rec, i+1)
                else:
                    rec.append(i+1)

    return rec
```

Append in a sorted way