Notebook

August 2, 2024

[ ]:
```
from typing import List
```

[ ]:
```
# https://leetcode.com/problems/minimize-the-maximum-difference-of-pairs/

"""
Input: nums = [10,1,2,7,1,3], p = 2
Output: 1
Explanation: The first pair is formed from the indices 1 and 4, and the second␣
 ↪pair is formed from the indices 2 and 5.
The maximum difference is max(|nums[1] - nums[4]|, |nums[2] - nums[5]|) =␣
 ↪max(0, 1) = 1. Therefore, we return 1.
"""

class Solution:
    def minimizeMax(self, nums: List[int], p: int) -> int:
        nums.sort()

        def check(nums, mid, p):
            i = 1
            while i < len(nums):
                if nums[i]-nums[i-1] <= mid:
                    p -= 1
                    i += 1
                    if p <= 0 : return True
                i += 1
            return p <= 0

        l, r = 0, nums[-1] - nums[0]
        ans = -1
        while l <= r:
            mid = l + (r - l) // 2
            if check(nums, mid, p):
                # print(mid)
                ans = mid
                r = mid-1
            else:
                l = mid+1
```

```
        return ans
```

[ ]:
```
# https://leetcode.com/problems/house-robber-iv/solutions/3143741/
 ↪binary-search-c-with-similar-problems/
# Minimum of maximum
"""
Input: nums = [2,7,9,3,1], k = 2
Output: 2
Explanation: There are 7 ways to rob the houses. The way which leads to minimum␣
 ↪capability is to rob the house at index 0 and 4. Return max(nums[0],␣
 ↪nums[4]) = 2.
"""

class Solution:
    def minCapability(self, nums: List[int], k: int) -> int:
        def helper(nums, mid, k):
            i, n = 0, len(nums)
            while i < n:
                if nums[i] <= mid:
                    k -= 1
                    i += 2
                else:
                    i += 1
                if k == 0: return True
            return k <= 0
        l, h = 0, 1000000000
        while l < h:
            mid = l + (h - l) // 2
            if (helper(nums, mid, k)):
                h = mid
            else:
                l = mid+1
        return l
```

[ ]:
```
# https://leetcode.com/problems/find-peak-element/description/
# return any peak element which is greater than its neighbors

# use binary search to find the peak element
# Time complexity: O(logn)
class Solution:
    def findPeakElement(self, nums: List[int]) -> int:
        left, right = 0, len(nums)-1
        while left < right-1:
            mid = (left + right) // 2
            if nums[mid] > nums[mid+1] and nums[mid] > nums[mid-1]:
                return mid
```

```
            if nums[mid] < nums[mid+1]: left = mid+1
            else: right = mid-1
        return left if nums[left] >= nums[right] else right


# https://leetcode.com/problems/find-a-peak-element-ii/description/
# return peak in 2D array matrix

# Time complexity: O(m*logn), m is the number of rows, n is the number of␣
 ↪columns
class Solution(object):
    def findPeakGrid(self, mat):
        startCol = 0
        endCol = len(mat[0])-1

        while startCol <= endCol:
            maxRow = 0
            midCol = (endCol+startCol)//2

            for row in range(len(mat)):
                maxRow = row if (mat[row][midCol] >= mat[maxRow][midCol]) else␣
 ↪maxRow

            leftIsBig    =   midCol-1 >= startCol  and  mat[maxRow][midCol-1] >␣
 ↪mat[maxRow][midCol]
            rightIsBig   =   midCol+1 <= endCol    and  mat[maxRow][midCol+1] >␣
 ↪mat[maxRow][midCol]

            if (not leftIsBig) and (not rightIsBig):   # we have found the peak␣
 ↪element
                return [maxRow, midCol]
            elif rightIsBig:              # if rightIsBig, then there is an␣
 ↪element in 'right' that is bigger than all the elements in the 'midCol',
                startCol = midCol+1     # so 'midCol' cannot have 'peakPlane'
            else:                           # leftIsBig
                endCol = midCol-1

        return []


# https://leetcode.com/problems/peak-index-in-a-mountain-array/description/
# here peak where values are increasing and then decreasing arr[i] < arr[i+1] >␣
 ↪arr[i+2]

class Solution:
```

```python
    def peakIndexInMountainArray(self, arr: List[int]) -> int:
        left, right = 0, len(arr)-1
        while left < right:
            mid = (left + right) // 2
            if arr[mid] < arr[mid+1]:
                left = mid+1
            else:
                right = mid
        return left
```

[ ]:
```python
# https://github.com/doocs/leetcode/blob/main/solution/2100-2199/2137.Pour%20Water%20Between%20Buckets%20to%20Make%20Water%20Levels%20Equal/README_EN.md
# Pour Water Between Buckets to Make Water Levels Equal

"""
Input: buckets = [1,2,7], loss = 80       Output: 2.00000
Explanation: Pour 5 gallons of water from buckets[2] to buckets[0].
5 * 80% = 4 gallons are spilled and buckets[0] only receives 5 - 4 = 1 gallon of water.
All buckets have 2 gallons of water in them so return 2.

Input: buckets = [2,4,6], loss = 50      Output: 3.50000
Explanation: Pour 0.5 gallons of water from buckets[1] to buckets[0].
0.5 * 50% = 0.25 gallons are spilled and buckets[0] only receives 0.5 - 0.25 = 0.25 gallons of water.
Now, buckets = [2.25, 3.5, 6].
Pour 2.5 gallons of water from buckets[2] to buckets[0].
2.5 * 50% = 1.25 gallons are spilled and buckets[0] only receives 2.5 - 1.25 = 1.25 gallons of water.
All buckets have 3.5 gallons of water in them so return 3.5.
"""
# answer upto 1e-5 accepted
class Solution:
    def equalizeWater(self, buckets: List[int], loss: int) -> float:
        def check(v):
            a = b = 0
            for x in buckets:
                if x >= v:
                    a += x - v
                else:
                    b += (v - x) * 100 / (100 - loss)
            return a >= b

        l, r = 0, max(buckets)
        while r - l > 1e-5:
            mid = (l + r) / 2
```

```
            if check(mid):
                l = mid
            else:
                r = mid
        return l
```