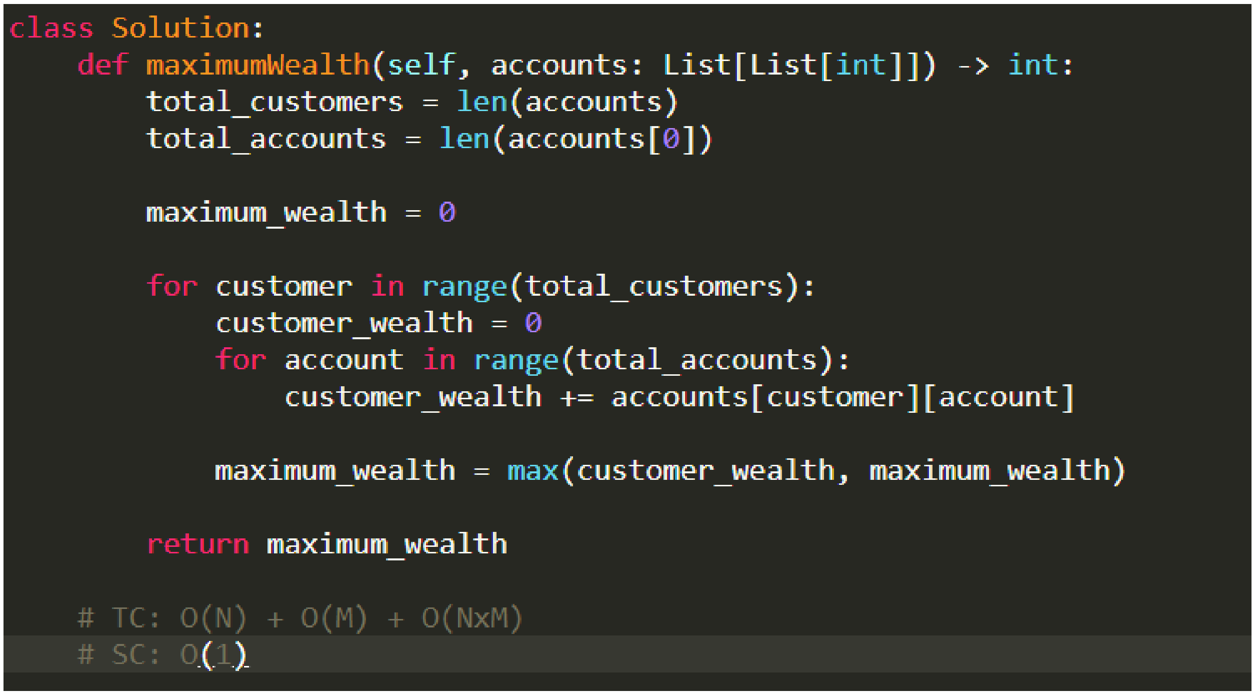
1. https://leetcode.com/problems/richest-customer-wealth/

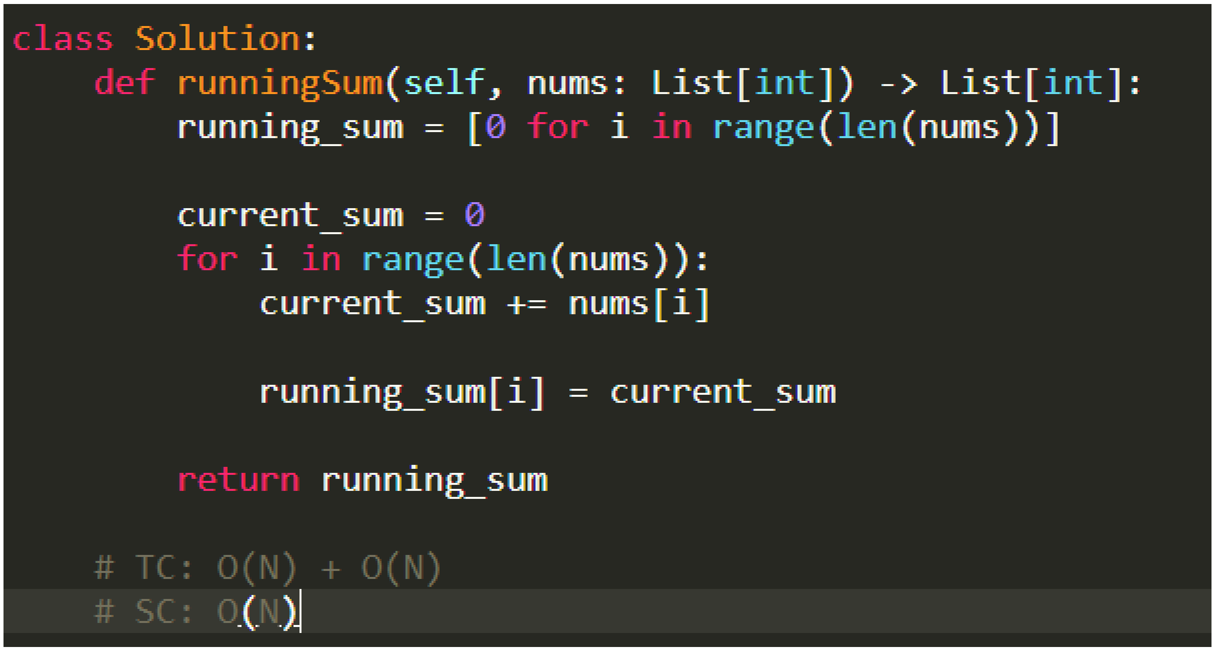
Brute force:
-> compare every row sum and find the maximum.



```
class Solution:
    def maximumWealth(self, accounts: List[List[int]]) -> int:
        total_customers = len(accounts)
        total_accounts = len(accounts[0])

        maximum_wealth = 0

        for customer in range(total_customers):
            customer_wealth = 0
            for account in range(total_accounts):
                customer_wealth += accounts[customer][account]

            maximum_wealth = max(customer_wealth, maximum_wealth)

        return maximum_wealth

    # TC: O(N) + O(M) + O(NxM)
    # SC: O(1)
```

2. https://leetcode.com/problems/running-sum-of-1d-array/

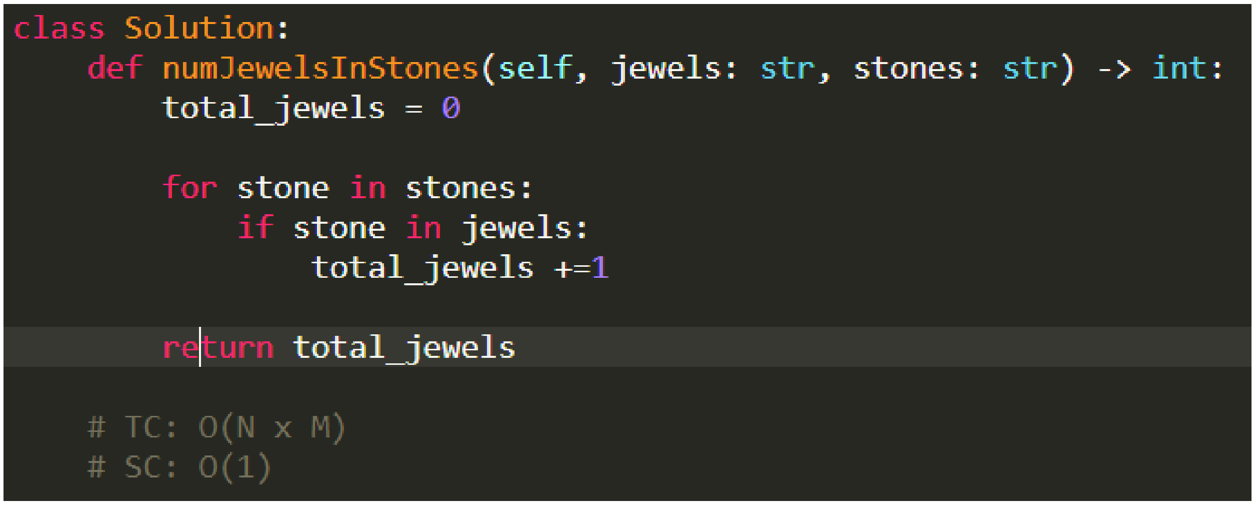
Brute force:
-> maintain sum up to ith index in an array.



```
class Solution:
    def runningSum(self, nums: List[int]) -> List[int]:
        running_sum = [0 for i in range(len(nums))]

        current_sum = 0
        for i in range(len(nums)):
            current_sum += nums[i]

            running_sum[i] = current_sum

        return running_sum

    # TC: O(N) + O(N)
    # SC: O(N)
```

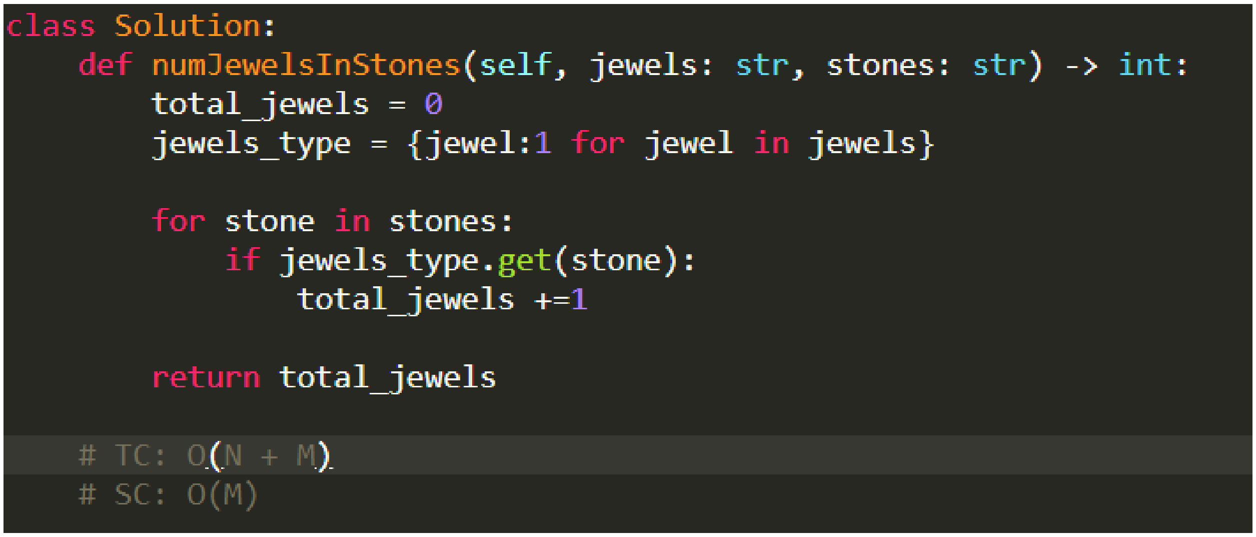
3. https://leetcode.com/problems/jewels-and-stones/

Brute force:
-> for every stone check if it is a jewel.



```
class Solution:
    def numJewelsInStones(self, jewels: str, stones: str) -> int:
        total_jewels = 0

        for stone in stones:
            if stone in jewels:
                total_jewels +=1

        return total_jewels

    # TC: O(N x M)
    # SC: O(1)
```

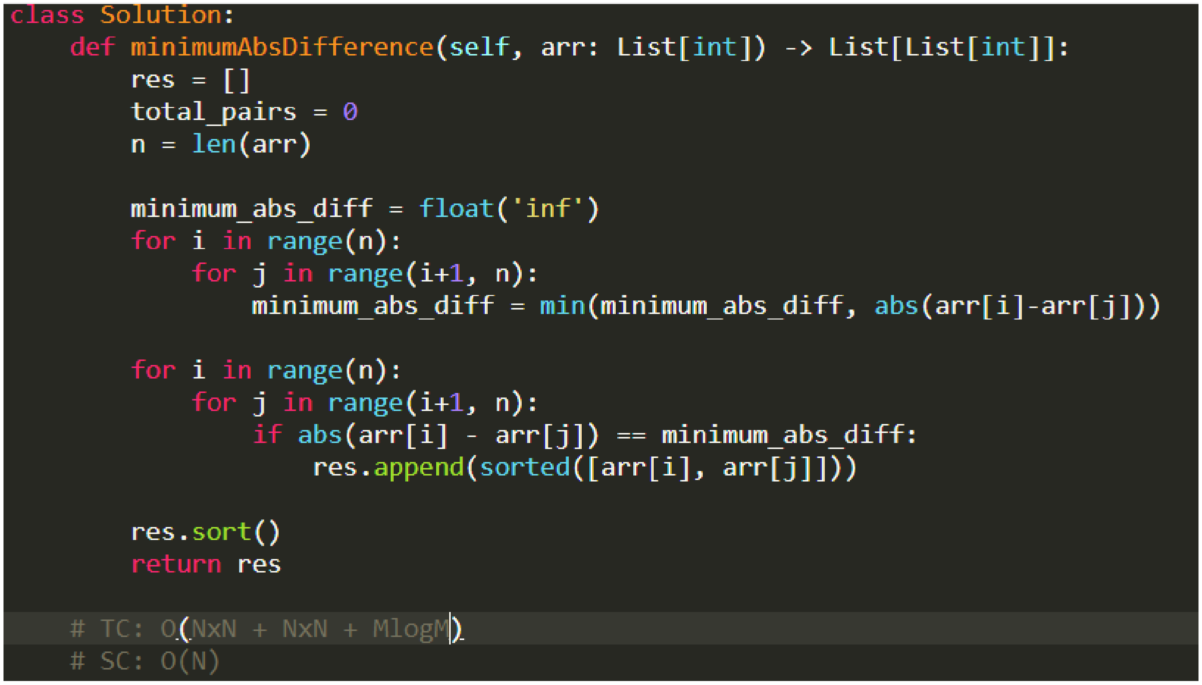
Better:
-> maintain a hash map to store type of jewels, so the lookup time for checking a stone is a jewel or not becomes O(1).



```
class Solution:
    def numJewelsInStones(self, jewels: str, stones: str) -> int:
        total_jewels = 0
        jewels_type = {jewel:1 for jewel in jewels}

        for stone in stones:
            if jewels_type.get(stone):
                total_jewels +=1

        return total_jewels

    # TC: O(N + M)
    # SC: O(M)
```

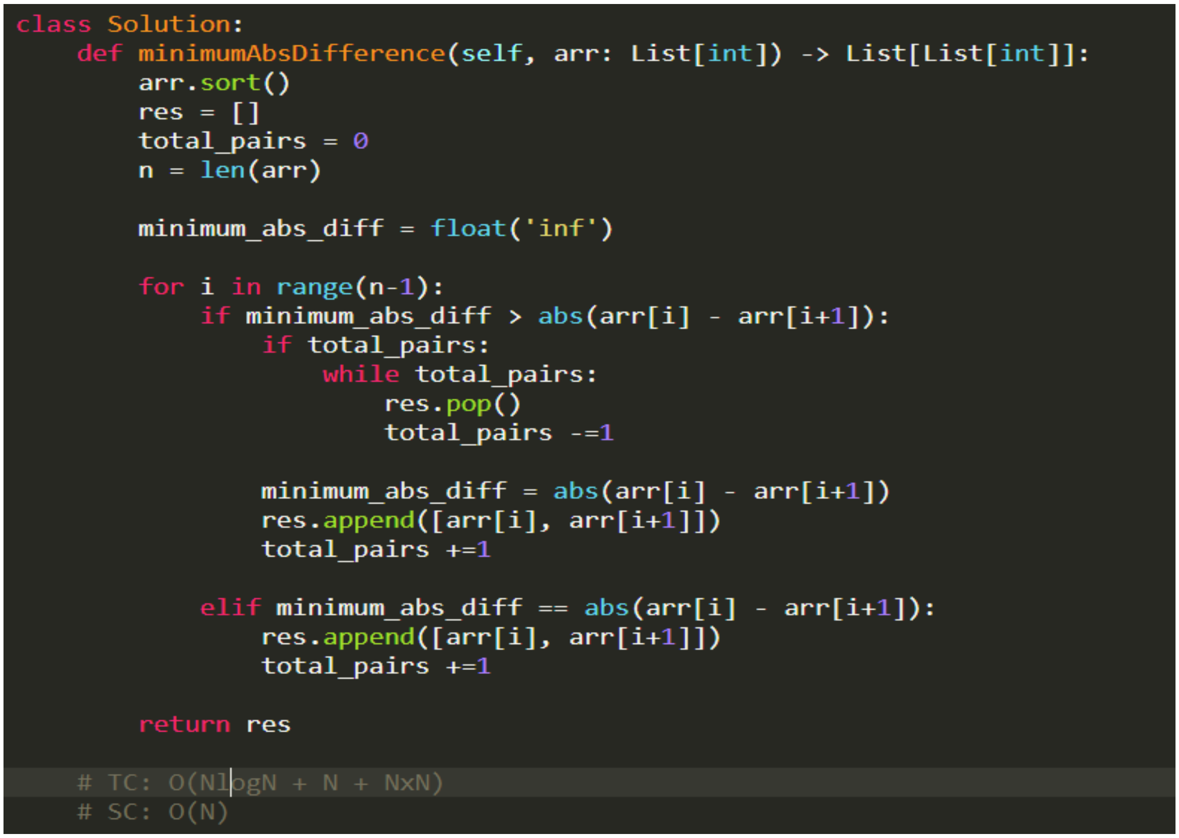
4. https://leetcode.com/problems/minimum-absolute-difference/

Brute force:
-> generate all pairs and find the minimum absolute difference.
-> again generate all pairs and those with minimum absolute difference, sort them and store them in result, then again sort the resultant pairs.



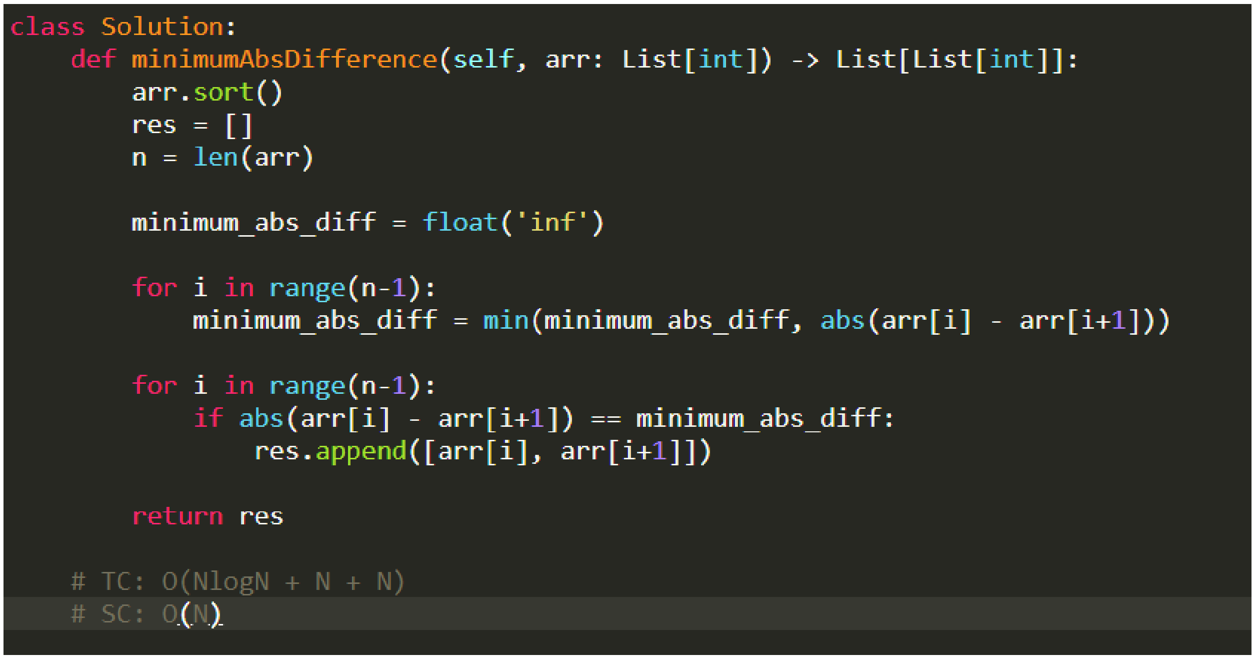
```
class Solution:
    def minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:
        res = []
        total_pairs = 0
        n = len(arr)

        minimum_abs_diff = float('inf')
        for i in range(n):
            for j in range(i+1, n):
                minimum_abs_diff = min(minimum_abs_diff, abs(arr[i]-arr[j]))

        for i in range(n):
            for j in range(i+1, n):
                if abs(arr[i] - arr[j]) == minimum_abs_diff:
                    res.append(sorted([arr[i], arr[j]]))

        res.sort()
        return res

    # TC: O(NxN + NxN + MlogM)
    # SC: O(N)
```

Better:
-> sort the array and maintain an array where resultant pairs will be stored.
-> iterate over the sorted array, check if the ith and (i+1)th element has minimum absolute difference till now, if yes, remove every pair from the resultant array if any present and store this pair.



```
class Solution:
    def minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:
        arr.sort()
        res = []
        total_pairs = 0
        n = len(arr)

        minimum_abs_diff = float('inf')

        for i in range(n-1):
            if minimum_abs_diff > abs(arr[i] - arr[i+1]):
                if total_pairs:
                    while total_pairs:
                        res.pop()
                        total_pairs -=1

                minimum_abs_diff = abs(arr[i] - arr[i+1])
                res.append([arr[i], arr[i+1]])
                total_pairs +=1

            elif minimum_abs_diff == abs(arr[i] - arr[i+1]):
                res.append([arr[i], arr[i+1]])
                total_pairs +=1

        return res

    # TC: O(NlogN + N + NxN)
    # SC: O(N)
```

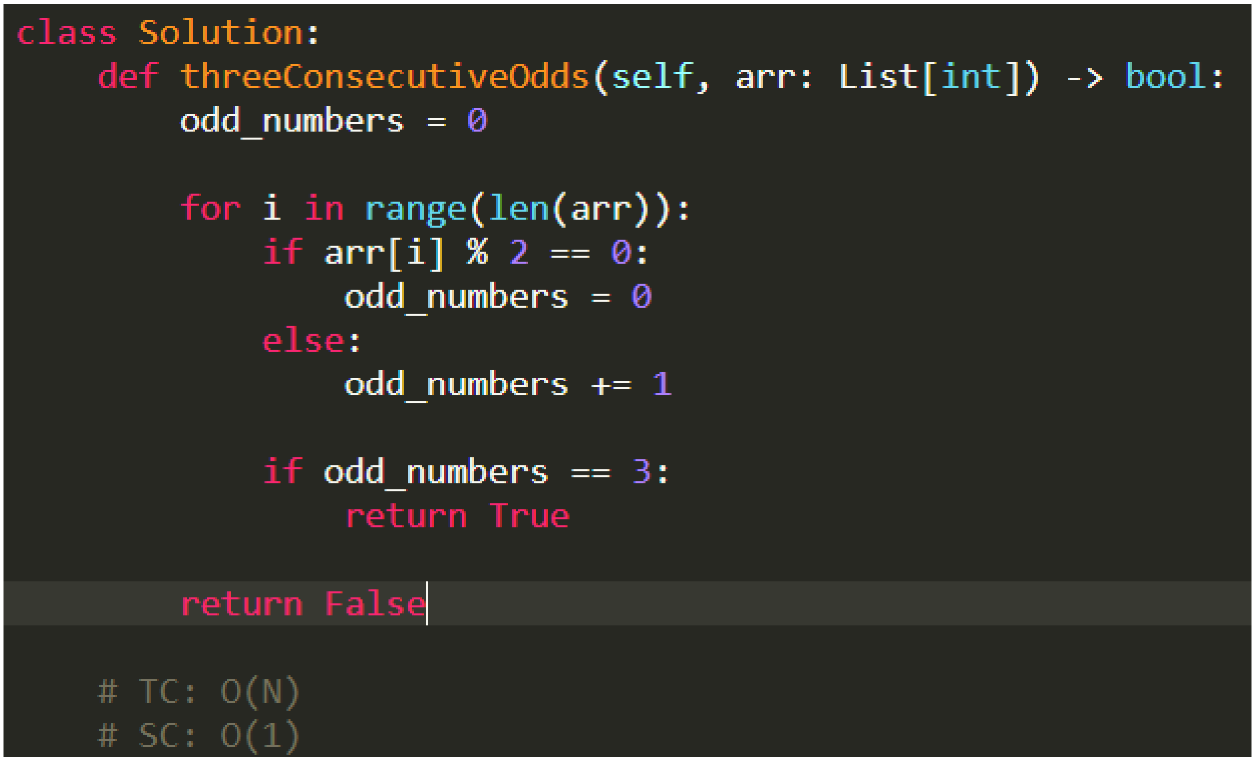
Optimal:
-> sort the array, find the minimum absolute difference by iterating on the array once.
-> then again iterate through array and store those pairs in resultant array.



```
class Solution:
    def minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:
        arr.sort()
        res = []
        n = len(arr)

        minimum_abs_diff = float('inf')

        for i in range(n-1):
            minimum_abs_diff = min(minimum_abs_diff, abs(arr[i] - arr[i+1]))

        for i in range(n-1):
            if abs(arr[i] - arr[i+1]) == minimum_abs_diff:
                res.append([arr[i], arr[i+1]])

        return res

    # TC: O(NlogN + N + N)
    # SC: O(N)
```

5. https://leetcode.com/problems/three-consecutive-odds/

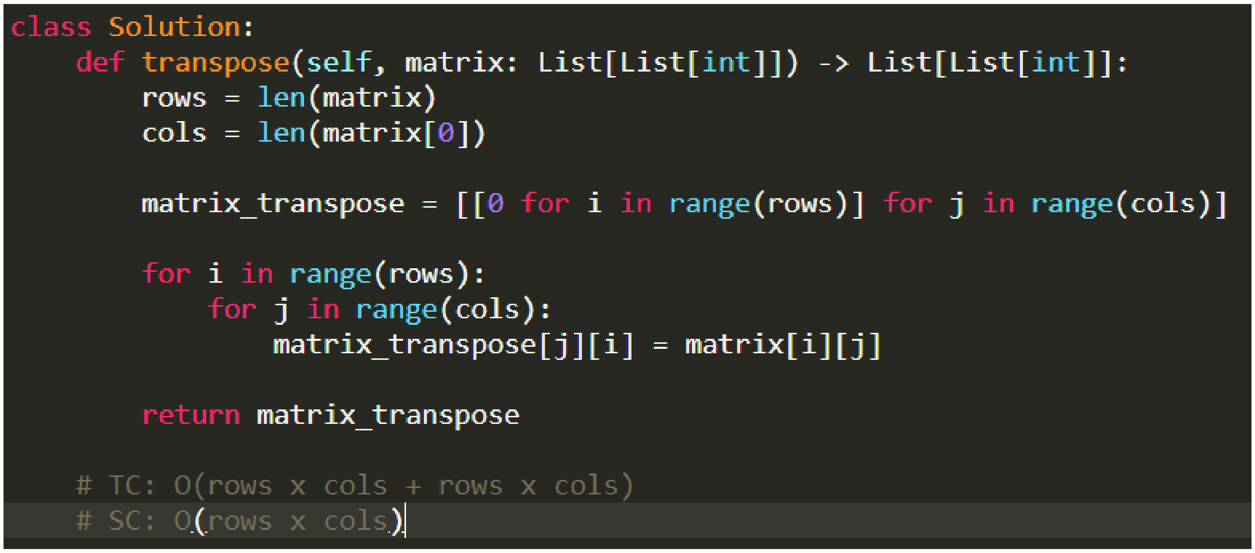
Brute force:
-> simply maintain a count of consecutive odd numbers, if it reaches 3 break the loop or if even number comes in between make it 0.



```
class Solution:
    def threeConsecutiveOdds(self, arr: List[int]) -> bool:
        odd_numbers = 0

        for i in range(len(arr)):
            if arr[i] % 2 == 0:
                odd_numbers = 0
            else:
                odd_numbers += 1

            if odd_numbers == 3:
                return True

        return False

    # TC: O(N)
    # SC: O(1)
```

6. https://leetcode.com/problems/transpose-matrix/

Brute force:
-> create another matrix which represents the transpose of this matrix, and arr[row][col] = transpose[col][row].

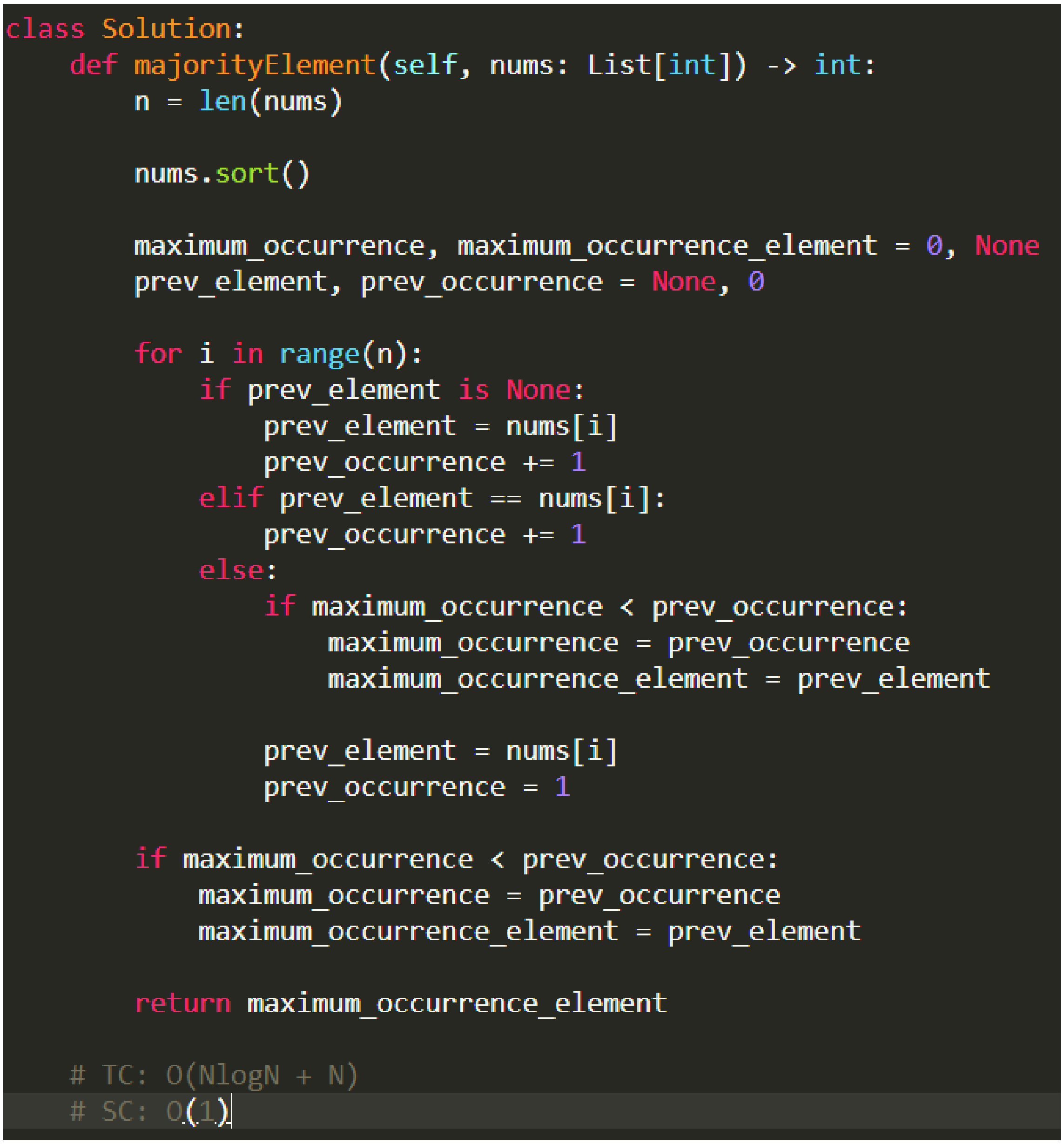


```
class Solution:
    def transpose(self, matrix: List[List[int]]) -> List[List[int]]:
        rows = len(matrix)
        cols = len(matrix[0])

        matrix_transpose = [[0 for i in range(rows)] for j in range(cols)]

        for i in range(rows):
            for j in range(cols):
                matrix_transpose[j][i] = matrix[i][j]

        return matrix_transpose

    # TC: O(rows x cols + rows x cols)
    # SC: O(rows x cols)
```

7. https://leetcode.com/problems/majority-element/

Brute force:
-> iterate through the given array and for each element check it's occurrences and return the element which occurred the maximum number of times.
TC: O(n^2)
SC: O(1)

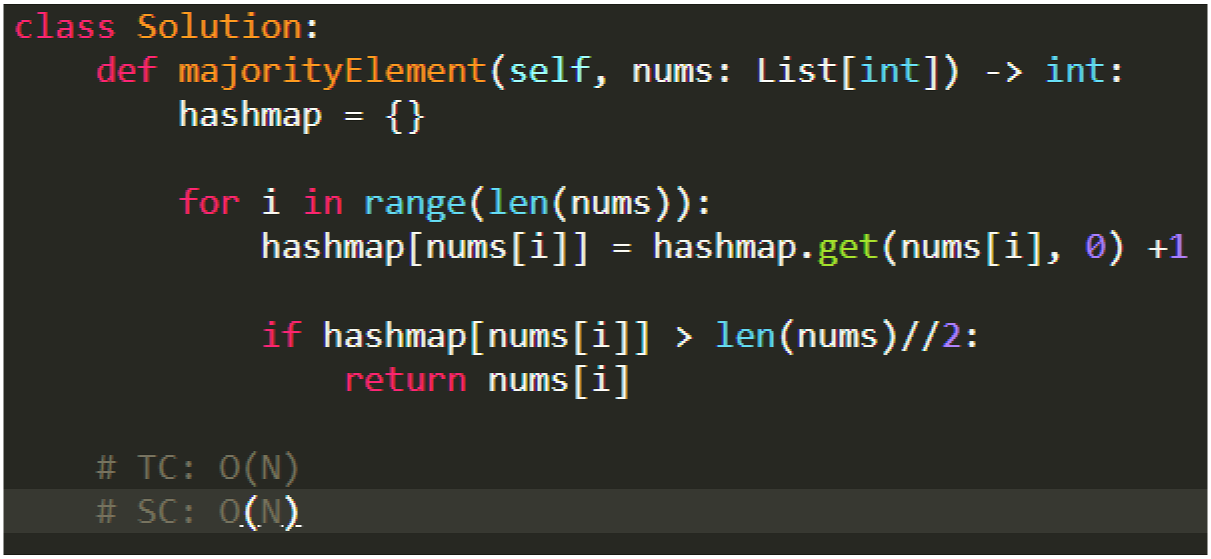
Better:
-> sort the array, then iterate through the array and count the number of occurrences of unique elements and accordingly find the majority element.



```
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        n = len(nums)

        nums.sort()

        maximum_occurrence, maximum_occurrence_element = 0, None
        prev_element, prev_occurrence = None, 0

        for i in range(n):
            if prev_element is None:
                prev_element = nums[i]
                prev_occurrence += 1
            elif prev_element == nums[i]:
                prev_occurrence += 1
            else:
                if maximum_occurrence < prev_occurrence:
                    maximum_occurrence = prev_occurrence
                    maximum_occurrence_element = prev_element

                prev_element = nums[i]
                prev_occurrence = 1

        if maximum_occurrence < prev_occurrence:
            maximum_occurrence = prev_occurrence
            maximum_occurrence_element = prev_element

        return maximum_occurrence_element

    # TC: O(NlogN + N)
    # SC: O(1)
```

Better:

-> iterate over the array and maintain a hash map, to store count of unique elements, at any moment while iterating if any element found more than n/2 times, no need to further iterate.

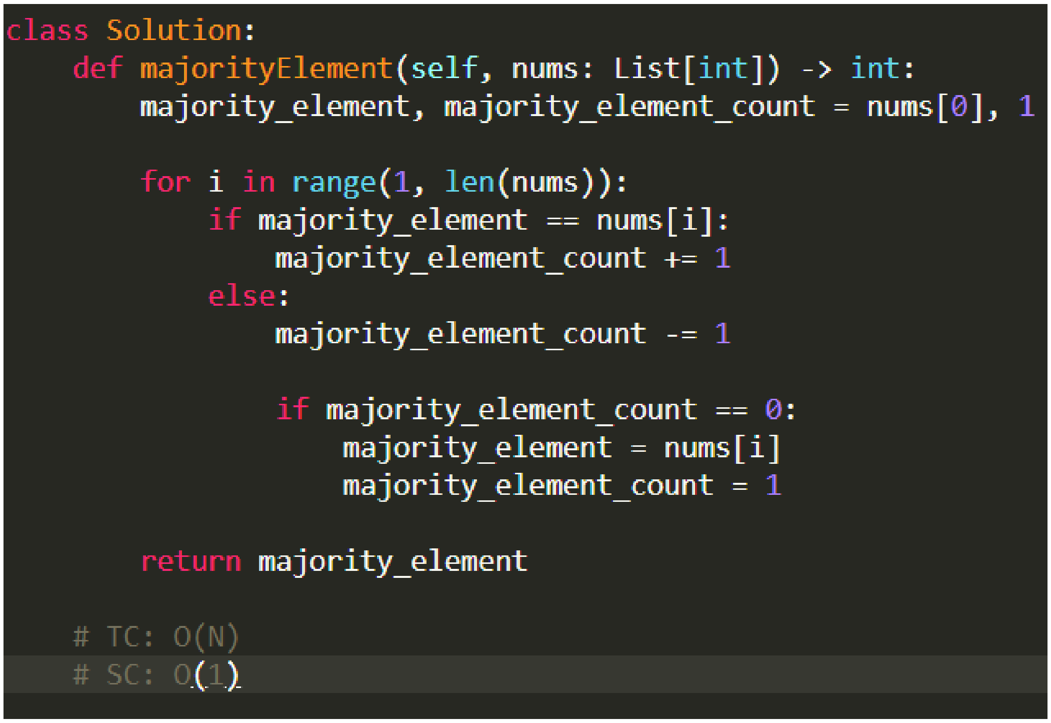


```
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        hashmap = {}

        for i in range(len(nums)):
            hashmap[nums[i]] = hashmap.get(nums[i], 0) +1

            if hashmap[nums[i]] > len(nums)//2:
                return nums[i]

    # TC: O(N)
    # SC: O(N)
```

Optimal:

-> using Moore's algorithm, in which we'll maintain two variables, one specifying majority element and other specifying the count by which the majority element is major.

-> if an element is occurring more than n/2 times then, after iterating the whole array we'll have that element as the majority element.

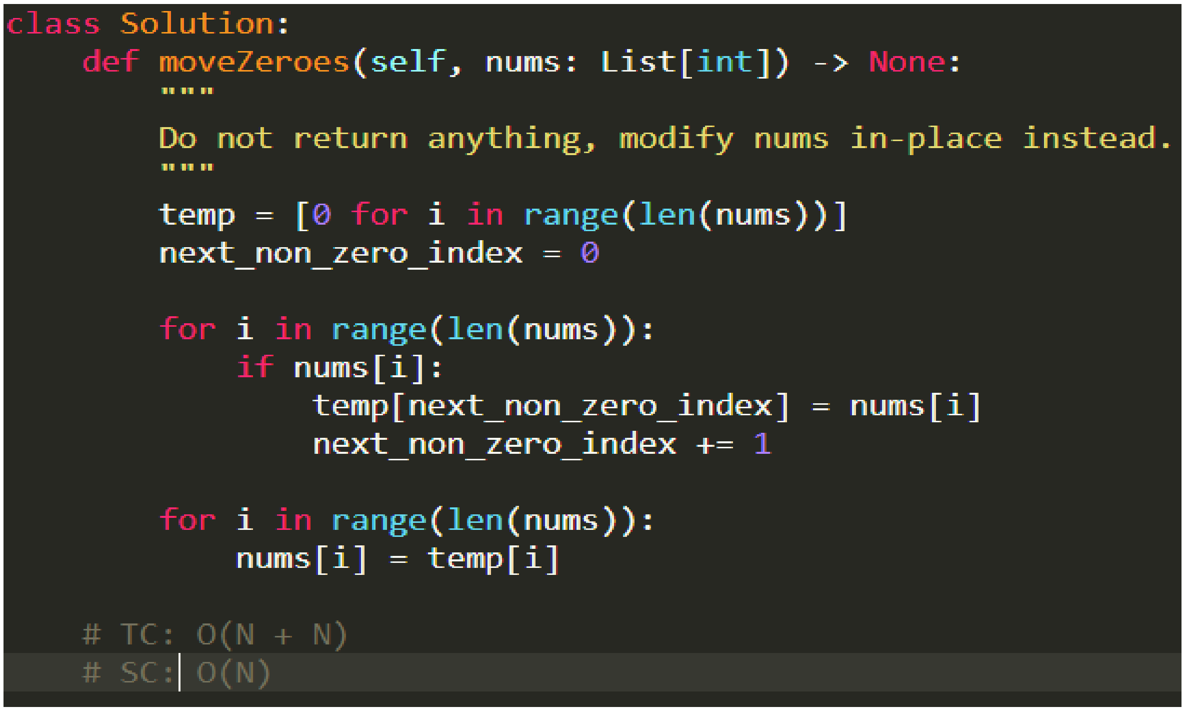


```
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        majority_element, majority_element_count = nums[0], 1

        for i in range(1, len(nums)):
            if majority_element == nums[i]:
                majority_element_count += 1
            else:
                majority_element_count -= 1

                if majority_element_count == 0:
                    majority_element = nums[i]
                    majority_element_count = 1

        return majority_element

    # TC: O(N)
    # SC: O(1)
```

8. https://leetcode.com/problems/move-zeroes/

Brute:

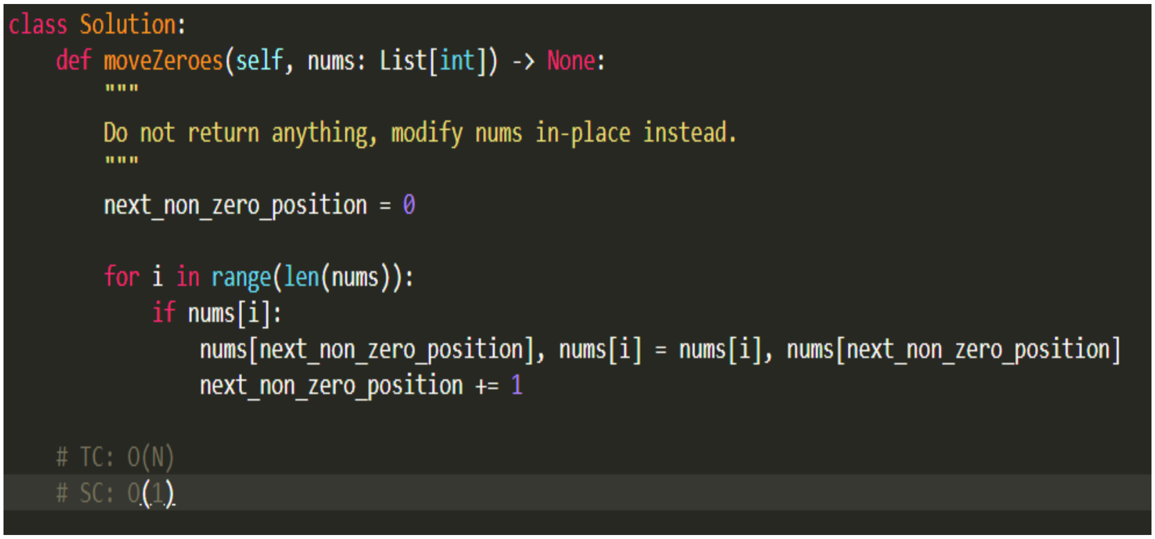
-> maintain a temporary array, place all the non zero element in that array and copy them back to the original array.



```python
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        temp = [0 for i in range(len(nums))]
        next_non_zero_index = 0

        for i in range(len(nums)):
            if nums[i]:
                temp[next_non_zero_index] = nums[i]
                next_non_zero_index += 1

        for i in range(len(nums)):
            nums[i] = temp[i]

    # TC: O(N + N)
    # SC: O(N)
```

Optimal:

-> maintain a variable which signifies the position for a non zero element, as a non zero element is placed at this position, increment it by 1.



```python
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        next_non_zero_position = 0

        for i in range(len(nums)):
            if nums[i]:
                nums[next_non_zero_position], nums[i] = nums[i], nums[next_non_zero_position]
                next_non_zero_position += 1

    # TC: O(N)
    # SC: O(1)
```