1.

import random as rd

class Solution(object):

def threeSumClosest(self, nums, target):

nums.sort()

n = len(nums)

closest\_sum = float('inf')

for i in range(n - 2):

l = i + 1

r = n - 1

while l < r:

current\_sum = nums[i] + nums[l] + nums[r]

if current\_sum == target:

return current\_sum

if abs(current\_sum - target) < abs(closest\_sum - target):

closest\_sum = current\_sum

if current\_sum < target:

l += 1

else:

r -= 1

return closest\_sum

2.

class Solution(object):

def merge(self, intervals):

intervals.sort(key=lambda x: x[0])

merged = []

for interval in intervals:

if not merged or merged[-1][1] < interval[0]:

merged.append(interval)

else:

merged[-1][1] = max(merged[-1][1], interval[1])

return merged

3.class Solution(object):

def merge(self, nums1, m, nums2, n):

p1 = m - 1

p2 = n - 1

p = m + n - 1

while p1 >= 0 and p2 >= 0:

if nums1[p1] > nums2[p2]:

nums1[p] = nums1[p1]

p1 -= 1

else:

nums1[p] = nums2[p2]

p2 -= 1

p -= 1

while p2 >= 0:

nums1[p] = nums2[p2]

p2 -= 1

p -= 1

return nums1

4.class Solution(object):

def intersection(self, nums1, nums2):

lst = []

for i in range(0, len(nums1)):

for j in range(0, len(nums2)):

if nums1[i] == nums2[j]:

lst.append(nums1[i])

return list(set(lst))

5.class Solution(object):

def findMinArrowShots(self, points):

if not points:

return 0

points.sort(key=lambda x: x[1])

arrows = 1

end = points[0][1]

for i in range(1, len(points)):

if points[i][0] > end:

arrows += 1

end = points[i][1]

return arrows

6.class Solution(object):

def minMoves2(self, nums):

nums.sort()

n = len(nums)

median = nums[n // 2]

moves = sum(abs(num - median) for num in nums)

return moves

7.

class Solution(object):

def findMinDifference(self, timePoints):

def timeToMinutes(time):

hours, minutes = map(int, time.split(':'))

return hours \* 60 + minutes

minutes = [timeToMinutes(time) for time in timePoints]

minutes.sort()

min\_diff = float('inf')

for i in range(1, len(minutes)):

min\_diff = min(min\_diff, minutes[i] - minutes[i - 1])

min\_diff = min(min\_diff, 1440 + minutes[0] - minutes[-1])

return min\_diff

8.class Solution(object):

def maximumProduct(self, nums):

nums.sort()

max1 = nums[-1] \* nums[-2] \* nums[-3]

max2 = nums[0] \* nums[1] \* nums[-1]

return max(max1, max2)

9.

class Solution(object):

def dominantIndex(self, nums):

if len(nums) == 0:

return -1

max\_num = max(nums)

max\_index = nums.index(max\_num)

for i, num in enumerate(nums):

if i != max\_index and max\_num < 2 \* num:

return -1

return max\_index

10.

class Solution(object):

def customSortString(self, order, s):

count\_s = {}

for char in s:

if char in count\_s:

count\_s[char] += 1

else:

count\_s[char] = 1

result = []

for char in order:

if char in count\_s:

result.append(char \* count\_s[char])

del count\_s[char]

for char in count\_s:

result.append(char \* count\_s[char])

return ''.join(result)

11.

class Solution(object):

def findReplaceString(self, s, indices, sources, targets):

replacements = []

for i in range(len(indices)):

index = indices[i]

source = sources[i]

target = targets[i]

if s[index:index+len(source)] == source:

replacements.append((index, source, target))

replacements.sort()

result = []

prev\_index = 0

for index, source, target in replacements:

result.append(s[prev\_index:index])

result.append(target)

prev\_index = index + len(source)

result.append(s[prev\_index:])

return ''.join(result)

12.

even\_index = 0

odd\_index = 1

n = len(nums)

while even\_index < n and odd\_index < n:

if nums[even\_index] % 2 == 0:

even\_index += 2

elif nums[odd\_index] % 2 == 1:

odd\_index += 2

else:

# Swap nums[even\_index] and nums[odd\_index]

nums[even\_index], nums[odd\_index] = nums[odd\_index], nums[even\_index]

even\_index += 2

odd\_index += 2

return nums

13.

n = len(nums)

result = [0] \* n

left, right = 0, n - 1

pos = n - 1

while left <= right:

left\_square = nums[left] \*\* 2

right\_square = nums[right] \*\* 2

if left\_square > right\_square:

result[pos] = left\_square

left += 1

else:

result[pos] = right\_square

right -= 1

pos -= 1

return result

14.

class Solution(object):

def heightChecker(self, heights):

expected = sorted(heights)

count = 0

for i in range(len(heights)):

if heights[i] != expected[i]:

count += 1

return count

15.

class Solution(object):

def removeCoveredIntervals(self, intervals):

intervals.sort(key=lambda x: (x[0], -x[1]))

count = 0

end = 0

for interval in intervals:

if interval[1] > end:

count += 1

end = interval[1]

return count

16.

class Solution(object):

def minimumAbsDifference(self, arr):

arr.sort()

min\_diff = float('inf')

for i in range(1, len(arr)):

min\_diff = min(min\_diff, arr[i] - arr[i - 1])

result = []

for i in range(1, len(arr)):

if arr[i] - arr[i - 1] == min\_diff:

result.append([arr[i - 1], arr[i]])

return result

17.

class Solution(object):

def canMakeArithmeticProgression(self, arr):

arr.sort()

diff = arr[1] - arr[0]

for i in range(2, len(arr)):

if arr[i] - arr[i - 1] != diff:

return False

return True

18.

class Solution(object):

def average(self, salary):

min\_salary = float('inf')

max\_salary = float('-inf')

total\_sum = 0

for s in salary:

if s < min\_salary:

min\_salary = s

if s > max\_salary:

max\_salary = s

total\_sum += s

total\_sum -= min\_salary + max\_salary

average\_salary = total\_sum / (len(salary) - 2)

return average\_salary

19.

class Solution(object):

def sortJumbled(self, mapping, nums):

def transform\_number(num, mapping):

num\_str = str(num)

mapped\_num = []

for char in num\_str:

digit = int(char)

mapped\_num.append(mapping[digit])

return int(''.join(map(str, mapped\_num)))

mapped\_values = [(num, transform\_number(num, mapping)) for num in nums]

sorted\_nums = sorted(mapped\_values, key=lambda x: x[1])

return [num for num, \_ in sorted\_nums]

20.

class Solution(object):

def minMovesToSeat(self, seats, students):

seats.sort()

students.sort()

n = len(seats)

total\_moves = 0

for i in range(n):

total\_moves += abs(students[i] - seats[i])

return total\_moves