**Q1. Minimum Weight** :

import heapq

def push(heap, value):

heapq.heappush(heap, -value)

def pop(heap):

return -heapq.heappop(heap)

def solve(weights, d):

heap = []

for weight in weights:

push(heap, weight)

for \_ in range(d):

t = pop(heap)

t //= 2

push(heap, t)

ans = 0

for value in heap:

ans += -value

return ans

# Example usage:

weights = [9, 7, 8, 3]

d = 3

print(solve(weights, d))

**Q2. Counting Triplets :**

from collections import defaultdict

class Solution:

def divisibleTripletCount(self, nums, d):

cnt = defaultdict(int)

ans = 0

n = len(nums)

for j in range(n):

for k in range(j + 1, n):

x = (d - (nums[j] + nums[k]) % d) % d

ans += cnt[x]

cnt[nums[j] % d] += 1

return ans

# Example usage:

nums = [3, 6, 9, 12]

d = 3

solution = Solution()

print(solution.divisibleTripletCount(nums, d))

**Q3. Complementary Pairs :**

from collections import defaultdict

def getCount(N, s):

mp = defaultdict(int)

ans = 0

for i in range(N):

a = [0] \* 26

for char in s[i]:

a[ord(char) - ord('a')] += 1

for j in range(26):

a[j] %= 2

ans += mp[tuple(a)]

for j in range(26):

changedCount = a[:]

changedCount[j] = 1 - changedCount[j]

ans += mp[tuple(changedCount)

mp[tuple(a)] += 1

return ans

# Example usage:

N = 3

s = ["abc", "bca", "cab"]

print(getCount(N, s))

**Q4. Efficient Study :**

def knapSackTopDownCode(val, wt, W, n):

mat = [[0] \* (W + 1) for \_ in range(n + 1)]

for i in range(1, n + 1):

for j in range(1, W + 1):

if wt[i - 1] <= j:

mat[i][j] = max(val[i - 1] + mat[i - 1][j - wt[i - 1]], mat[i - 1][j])

else:

mat[i][j] = mat[i - 1][j]

return mat[n][W]

def maximumLearning(iv, articles, p):

size = len(articles)

art = [0] \* size

ivs = [0] \* size

for i in range(size):

art[i] = articles[i] \* 2

ivs[i] = iv[i]

return knapSackTopDownCode(ivs, art, p, size)

# Example usage:

iv = [60, 100, 120]

articles = [10, 20, 30]

p = 50

print(maximumLearning(iv, articles, p))

**Q5. Swap Parity :**

def solve():

str\_input = input().strip()

N = len(str\_input)

ans = ""

freq = [0] \* 10

left = 0

while left < N:

right = left

while right < (N - 1) and (int(str\_input[right]) % 2 == int(str\_input[right + 1]) % 2):

freq[int(str\_input[right])] += 1

right += 1

freq[int(str\_input[right])] += 1

left = right + 1

print("Frequency of digits:", freq)

# Example usage:

solve()

**Q6. Largest String :**

def solve():

s = input().strip()

N = len(s)

K = int(input().strip())

mp = {}

for ch in s:

if ch in mp:

mp[ch] += 1

else:

mp[ch] = 1

c = 0

a = []

while mp:

it = max(mp)

if c == K:

c = 0

it = sorted(mp.keys())[-2]

a.append(it)

mp[it] -= 1

c += 1

if mp[it] <= 0:

del mp[it]

j = len(a) - 1

while j > 0 and a[j] == a[j - 1]:

j -= 1

print(''.join(a[:j + K]))

# Example usage:

solve()

**Q7. Two Operations Redux :**

def solve():

A, B, C = map(int, input().split())

if C < abs(A - B):

print(-1)

else:

print(max(A, B) + (C - abs(A - B)) // 2)

# Example usage:

solve()

**Q8. Task Scheduling :**

def solve():

N, M = map(int, input().split())

a = list(map(int, input().split()))

mp = {}

for i in range(N):

b = int(input())

if b not in mp:

mp[b] = []

mp[b].append(a[i])

c = 0

for key, value in mp.items():

value.sort()

i, j = 0, len(value) - 1

while i < j:

if value[i] + value[j] > M:

c += 1

j -= 1

else:

i += 1

j -= 1

c += 1

if i == j:

c += 1

print(c)

# Example usage:

solve()

**Q9. Biased Die Simulator :**

from functools import cache

MOD = 1000000007

def dieSimulator(n, rollMax):

@cache

def dfs(i, j, x):

if i >= n:

return 1

ans = 0

for k in range(1, 7):

if k != j:

ans += dfs(i + 1, k, 1)

elif x < rollMax[j - 1]:

ans += dfs(i + 1, j, x + 1)

return ans % MOD

return dfs(0, 0, 0)

# Example usage:

n = 3 # Number of rolls

dice = [1, 1, 2, 2, 3, 3]

print(dieSimulator(n, dice))