## 1. Написать программу для решения задачи минимизации суммарного штрафа.

Постановка: Имеется набор из n работ. Для каждой работы известны:

* C[i] — **штраф за единицу времени**, на которое выполнение откладывается (например, за каждую минуту просрочки);
* T[i] — **время выполнения** работы.

Нужно найти **такой порядок выполнения работ**, чтобы **суммарный штраф за ожидание** был минимальным.

Смысл: Каждая следующая работа «ждёт», пока предыдущие завершатся. Чем дольше она ждёт — тем больше суммарный штраф. Задача — **сначала делать те работы, для которых ожидание обходится дороже**.

Алгоритм: Используется **жадная стратегия**:

* Сортировать работы по убыванию отношения **штраф/время**: C[i] / T[i]
* Это обеспечивает наименьшую "стоимость" ожидания за каждую минуту.

def SumOfQueue(mas):

sum = 0

tempSum = 0

for i in range(9):

sum += tempSum \* mas[i][0]

tempSum += mas[i][1]

return sum

C = [7, 11, 9, 12, 15, 9, 10, 8, 4]

T = [14, 18, 15, 20, 35, 19, 13, 16, 12]

mas = [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0],

[0, 0, 0], [0, 0, 0]]

for i in range(len(C)):

mas[i][0] = C[i]

mas[i][1] = T[i]

mas[i][2] = C[i] / T[i]

print("Не отсортированный массив: ", mas)

beforeSort = SumOfQueue(mas)

print("Сумма до сортировки: ", beforeSort)

mas.sort(key=lambda x: x[2], reverse=True)

print("Отсортированный массив: ", mas)

afterSort = SumOfQueue(mas)

print("Сумма после сортировки: ", afterSort)

print("Разница: ", abs(beforeSort - afterSort))

## 2. Написать программу для решения задачи «директора» (одного станка).

Постановка:

На одном станке нужно последовательно выполнить несколько заданий. Каждое имеет своё **время обработки**.

Цель — **минимизировать общее время ожидания всех работ**, то есть суммарное время, пока работа ожидает начала выполнения.

Алгоритм:

Используется **жадная стратегия**:

* Сначала выполнять те задачи, которые **самые короткие**.
* Сортировка по возрастанию времени работы.

Этот принцип называется **"Shortest Job First" (SJF)**.

def sumQueue(mas):

sum = 0

tempSum = 0

for i in range(9):

sum += tempSum

tempSum += mas[i][0]

return sum

T=[(26,1),(35,2),(30,3),(15,4),(25,5),(10,6),(45,7),(50,8),(40,9)]

print(T)

print('Сумма до сортировки: ',sumQueue(T))

T.sort()

print(T)

print('Сумма после сортировки: ',sumQueue(T))

## 3. Написать программу для решения задачи двух станков

Постановка:

Есть два станка **A** и **B**. Каждое задание можно выполнить на любом станке, но с разным временем.

Цель — **распределить задачи так, чтобы общее среднее время выполнения на станках было минимальным**.

Метод:

Применяется **модифицированный алгоритм Джонсона**:

* Ищем наименьшее из двух времён A[i] и B[i];
* Если минимальное время на станке **A**, задачу ставим ближе к началу;
* Если на **B**, задачу ставим ближе к концу;
* Отмечаем, что задача уже использована, заменяя её временно на 1000.

Это эвристика: она не гарантирует абсолютный минимум, но даёт **приближённое эффективное решение**.

A = [62, 85, 45, 37, 28, 54, 37, 30, 29]

B = [42, 53, 35, 40, 25, 33, 35, 26, 17]

l = 0;

r = len(A) - 1

C = [(0, 0) for i in range(len(A))]

sumA = []; quanA = 0; sumB = []; quanB = 0

for i in range(len(A)):

if (min(A) < min(B)):

minIndex = A.index(min(A))

C[l] = (A[minIndex], B[minIndex])

sumA.append(A[minIndex])

quanA += 1

A[minIndex] = 1000; B[minIndex] = 1000

l += 1

else:

minIndex = B.index(min(B))

C[r] = (A[minIndex], B[minIndex])

sumB.append(B[minIndex])

quanB += 1

A[minIndex] = 1000; B[minIndex] = 1000

r -= 1

print(C)

avgA = sum(sumA) / quanA;

avgB = sum(sumB) / quanB

print('Элементы A', sumA, 'Среднее время обслуживания на станке А: ', avgA, '\n', 'Элементы B', sumB,

'Среднее время обслуживания на станке B:', avgB, '\n', 'Среднее время', (avgA + avgB) / 2)