Разбор задач

Задача 1. Мойка автомобиля

Стоимость мытья машины по второму тарифу равна TY, а по третьему — VZ. Из этих величин, а также из стоимости мытья по фиксированному тарифу X, нужно взять минимум.

Пример решения.

```
x = int(input())
y = int(input())
z = int(input())
t = int(input())
v = int(input())
print(min(x, t * y, v * z))
```

Задача 2. Политическая борьба

Рассмотрим решение для частного случая $a < b$ и $c = 0$ (делегаты только двух партий, противников больше, чем сторонников).

Если делегата первой партии поселить с двумя делегатами другой партии, то его переманят оппоненты. Если же двух делегатов первой партии поселить с делегатом второй партии, то количество членов первой партии увеличится на 1. Поэтому делегатов первой партии нужно селить парами, добавляя к ним по одному делегату второй партии. Если a чётное, то делегаты первой партии смогут привлечь $a/2$ новых сторонников.

Если a нечётное, то один делегат первой партии перейдёт во вторую партию, если его поселить одного. Этого можно избежать, если трёх делегатов первой партии поселить в один номер, но тогда два других делегата не смогут привлечь одного нового сторонника. Можно поступить любым способом, результат будет одинаковым. Количество переманенных делегатов второй партии будет равно $(a - 3)/2$.

Пример решения для этого случая.

```
a = int(input())
b = int(input())
c = int(input())
n = (a + b + c) // 3
if a % 2 == 0:
    print(a + a // 2)
else:
    print(a + (a - 3) // 2)
```

Заметим, что это решение работает и в случае $a = 1$, тогда ответ равен 0.

Теперь рассмотрим полное решение. Определим количество комнат $n = \frac{a+b+c}{3}$. Если a не меньше $2n$, то в каждый номер можно поселить не менее двух делегатов первой партии, тогда наутро все делегаты станут членами первой партии. Ответ в этом случае $3n$.

Иначе будем селить делегатов первой партии по двое, чтобы каждые два делегата первой партии переманили одного делегата другой партии. Тогда количество переманенных делегатов будет равно a // 2 (целочисленное деление a на 2).

Если a нечётное, то одного делегата первой партии придётся поселить в номер с оппонентами. Переманивания этого делегата удастся избежать, если его поселить с представителями разных партий, что возможно, если $b > 0$ и $c > 0$. Если же a нечётно, и хотя бы одно из чисел b или c равно 0, то этого делегата сохранить не удастся — ответ будет меньше на 1 (как и ранее, его можно поселить в номер с двумя делегатами своей партии, но это тоже приведёт к уменьшению ответа на 1, так как уменьшится число переманенных членов других партий).

Пример решения для общего случая.

```
a = int(input())
```

```
b = int(input())
c = int(input())
n = (a + b + c) // 3
if a >= 2 * n:
    print(3 * n)
elif a % 2 == 1 and (b == 0 or c == 0):
    print(a + a // 2 - 1)
else:
    print(a + a // 2)
```

Задача 3. Путь зигзагом

Будем увеличивать координаты чередуя их, пока не придём в нужную точку ($x = a$, $y = b$). Если же одна координата уже приняла необходимое значение, а её нужно изменить, уменьшим её на 1, потом она снова увеличится на 1, то есть эта координата будет меняться вблизи нужного нам значения: a, $a - 1$, a, $a - 1$ и т.д., пока вторая координата не достигнет конечного значения.

В этом случае, возможно, мы сделаем лишний ход. Чтобы избежать этого лишнего хода, первое движение нужно делать в том направлении, в котором нам нужно переместиться на большее расстояние, то есть если $a > b$, то на первом шаге нужно менять x, а если $b > a$ — то y. Тогда мы получим наилучший ответ.

Докажем это. Пусть $m = \max(a, b)$. Тогда мы должны сделать как минимум m шагов в одном направлении, и ответ не может быть меньше $m + (m - 1)$, т.к. в другом направлении мы должны сделать как минимум $m - 1$ шагов. Более того, если a и b — одной чётности, то количества выполненных шагов в каждом направлении также должны быть одной чётности, и общее число шагов будет не менее $2m$. То есть общее число шагов не меньше $2m$, если a и b одной чётности, и $2m - 1$, если разной чётности.

Пусть $a > b$. Тогда выполнив a шагов по координате x мы окажемся либо в точке (a, b), либо в точке $(a, b-1)$, так как по координате y робот будет «бегать» между $y = b$ и $y = b-1$. Поскольку мы начали движение с координаты x, по оси y мы сделали на одно перемещение меньше, и координаты робота будут разной чётности. В какой именно точке он окажется, зависит именно от чётности b, поэтому в случае разных чётностей a и b он окажется в точке (a, b), и алгоритм завершит работу. Если же a и b одной чётности, то робот окажется в точке $(a, b - 1)$, и ему понадобится сделать ещё один шаг. Случай $a < b$ рассматривается аналогично, в случае $a = b$ наше решение достигнет цели ровно за $a + b$ шагов.

Пример решения. В этом решении в переменной move_x хранится логическое значение (True или False), означающее, что робот делает очередной шаг вдоль оси OX. Это значение будет меняться на противоположное на каждом шаге цикла.

```
a = int(input())
b = int(input())
move_x = a > b
x = 0
y = 0
while x != a or y != b:
    if move_x:
        if x < a:
            x += 1
        else:
            x -= 1
    else:
        if y < b:
            y += 1
        else:
            y -= 1
```

```
    print(x, y)
    move_x = not move_x
```

Задача 4. ВелоForces

Значение k может быть от 1 до n, т.к. слон Семён совершил хотя бы 1 заезд длиной 1 км, а всего он совершил n заездов.

Можно перебрать значения k от 1 до n и для каждого k посчитать, сколько в данном массиве чисел, которые не меньше k. Если это количество также не меньше k, то слон Семён достиг уровня k или выше. Запомним наибольшее из таких подходящих k. Пример такого решения.

```
n = int(input())
a = [int(input()) for i in range(n)]
ans = 0
for k in range(1, n + 1):
    cnt = 0
    for val in a:
        if val >= k:
            cnt += 1
    if cnt >= k:
        ans = k
print(ans)
```

Такое решение имеет сложность $O(n^2)$ и набирает 50 баллов.

Для полного решения заметим, что подсчитывать числа, которые не меньше определённого значения удобно, если отсортировать значения по неубыванию. Пусть массив отсортирован и индексы элементов массива начинаются с нуля. Если нулевой элемент массива не меньше n, то все остальные элементы массива тоже не меньше n и поэтому уровень слона Семёна будет равен n. Если это не так, но значение элемента массива с индексом 1 не меньше $n-1$, то уровень слона Семёна будет равен $n-1$. Если и это неверно, но при этом элемент массива с индексом 2 не меньше $n-2$, то уровень слона Семёна будет $n-2$. То есть нам нужно найти такое минимальное k, что a[k]>=n−k, тогда уровень слона Семёна будет $n-k$.

Такое решение имеет сложность $O(n \log n)$ (ввиду использования быстрой сортировки) и набирает 100 баллов.

```
n = int(input())
a = [int(input()) for i in range(n)]
a.sort()
k = 0
while a[k] < n - k:
    k += 1
print(n - k)
```

Задача 5. Хлеб и зрелище

В этой задаче в данном массиве из n элементов нужно найти два отрезка длины t каждый с минимальной общей суммой, при этом между отрезками должен быть хотя бы один элемент, не принадлежащий этим отрезкам.

Самая простая идея — перебрать начала двух отрезков. Если индекс первого элемента левого отрезка равен i, то правый отрезок может иметь начальным элементом отрезок с индексом $j = i+t+1$ или больше, начиная с этого значения и будем перебирать j. Дальше посчитаем суммы элементов на данных отрезках и выберем из них наименьшее. Такое решение будет иметь сложность $O(n^2 t)$.

```
n = int(input())
t = int(input())
d = [int(input()) for i in range(n)]
```

```
ans = 10**9
i = 0
while i + 2 * t + 1 <= n:
    j = i + t + 1
    while j + t <= n:
        ans = min(ans, sum(d[i:i+t]) + sum(d[j:j+t]))
        j += 1
    i += 1
print(ans)
```

Это решение можно улучшить при помощи стандартного приёма: предподсчитаем значения каждой нужной нам суммы. Пусть sums[i] будет равно сумме элементов массива d, начиная с i-го элемента. Один раз вычислим эти суммы, чтобы впоследствии не считать их заново. Сами эти суммы легко посчитать при помощи техники «скользящего окна»: если мы посчитали сумму элементов от d[i] до d[i+t−1], то чтобы перейти к следующей сумме нужно добавить значение d[i+t] и вычесть значение d[i].

Такое решение будет иметь сложность $O(n^2)$.

```
n = int(input())
t = int(input())
d = [int(input()) for i in range(n)]
sums = [0] * (n - t + 1)
sums[0] = sum(d[0:t])
for i in range(0, n - t):
    sums[i+1] = sums[i] + d[i+t] - d[i]
ans = 10**9
i = 0
while i + 2 * t + 1 <= n:
    j = i + t + 1
    while j + t <= n:
        ans = min(ans, sums[i] + sums[j])
        j += 1
    i += 1
print(ans)
```

Наконец, чтобы улучшить и это решение, можно заметить, что если мы выбрали какой-то ответ, то первый отрезок можно двигать на некотором префиксе нашего массива, а второй отрезок — на некотором суффиксе. То есть задача сводится к тому, что мы должны выбирать наименьшее значение среди значений массива sums на некотором его префиксе или суффиксе. Это можно сделать при помощи стандартной техники префиксных минимумов — предподсчитаем минимумы на всех префиксах и всех суффиксах массива. Это можно сделать за $O(n)$. Дальше можно, например, перебрать ту минуту, которую Петя обязательно проведёт в магазине. Нужно рассмотреть префиксы массива, не включающие эту минуту, и выбрать на этом префиксе отрезок из t элементов с минимальной суммой. Затем нужно рассмотреть суффиксы, не включающие эту минуту, и выбрать на этом суффиксе отрезок длины t с минимальной суммой. Используя массивы префиксных и суффиксных минимумов, это можно сделать за $O(1)$, и общее решение будет иметь сложность $O(n)$.

У этой идеи есть разные варианты, например, рассмотрим решение, вообще не использующее вспомогательные массивы. В этом решении мы будем перебирать индекс j начала правого отрезка. А значение i будет равно максимальному возможному началу левого отрезка, на самом деле i=j−t−1, но для простоты будем использовать две переменные. Переменная sum1 будет равна сумме отрезка длины t, начиная с индекса i, а переменная sum2 будет равна сумме отрезка длины t, начиная с индекса j.

При увеличении i и j на 1 переменные sum1 и sum2 пересчитываются добавлением одного и вычитанием другого элемента массива.

При фиксированном значении j в качестве начала левого отрезка можно взять любое значение, не превосходящее i, поэтому для фиксированного j наилучшая сумма, которую можно взять, равна sum2 и наименьшему из значений sum1, которые возникали до этого. Поэтому будем хранить в переменной min_sum1 наименьшее из значений sum1, пересчитывая его каждый раз при сдвиге i.

Такое решение имеет сложность $O(n)$.

```
n = int(input())
t = int(input())
d = [int(input()) for i in range(n)]
i = 0
j = t + 1
sum1 = sum(d[i:i+t])
sum2 = sum(d[j:j+t])
min_sum1 = sum1
ans = sum1 + sum2
while j + t < n:
    sum1 += d[i+t]
    sum1 -= d[i]
    min_sum1 = min(min_sum1, sum1)
    i += 1
    sum2 += d[j+t]
    sum2 -= d[j]
    j += 1
    ans = min(ans, min_sum1 + sum2)
print(ans)
```