Задача A. Место в поезде

Для решения задачи требуется сначала посчитать количество мест в вагоне $4 \cdot k$. Теперь найдем в каком вагоне едет Азат.

```
v = 0
while n > 4 * k:
  n -= 4*k
  v += 1
v += 1
```

Теперь найдем в какой кабинке.

```
while n > 4:
  n -= 4
  cb += 1
cb += 1
```

Осталось написать условие для определения верхнее или нижнее место в поезде.

```
if n == 1 or n == 4:
  print("Lower")
else:
  print("Upper")
```

Данный подход набирал 30 баллов. Для полного решения надо избавиться от циклов.

Запись $\lfloor x \rfloor$ обозначает округление вниз значения x (например, $\lfloor 1.4 \rfloor = 1$, $\lfloor 2 \rfloor = 2$).

Вычислять в каком вагоне едет Азат можно с помощью формулы: $v = \lfloor \frac{n+4 \cdot k-1}{4 \cdot k} \rfloor$. Надо не забыть изменить $n = n - 4 \cdot k \cdot (v-1)$.

Номер кабинки можно вычислить с помощью формулы: $cb = \lfloor \frac{n+3}{4} \rfloor$. Надо не забыть изменить $n = n - 4 \cdot (cb-1)$.

Теперь решение имеет асимптотику $O(1)$.

Задача B. Сложные строки

Заметим, что подстроки, которые подходят это палиндромы. Теперь сожмем строку, то есть подстроки с одинаковыми символами заменяем на пару из символа и количество подряд идущих одинаковых символов. То есть из строки *abbbaaabbb* делаем массив $comp = [(a,1),(b,3),(a,3),(b,3)]$. Теперь осталось пройтись по этому массиву и проверить $comp[i-1][0] == comp[i+1][0]$, если данное условие выполняется то проверяем длиннее ли новая подстрока чем ответ, который уже имеется.

```
s = input()

comp = [[s[0], 1]]

ans = 0
for i in range(1, len(s)):
  if s[i] == comp[-1][0]:
    comp[-1][1] += 1
  else:
    comp.append([s[i], 1])

for i in range(1, len(comp) - 1):
  if comp[i - 1][0] == comp[i + 1][0]:
    ans = max(ans, comp[i - 1][1] + comp[i][1] + comp[i + 1][1])
print(ans)
```

Задача C. Рассадка пассажиров

Подзадача 1

В текущей подзадаче можно создать двумерный массив a размером n на 3 и эмулировать процесс рассадки. Значение $a[i][j]$ равное *true* будет означать, что в ряду i на кресле j уже сидит пассажир, иначе кресло ещё пустое. При посадке очередного пассажира нужно проверить: сидит ли кто-нибудь

между проходом и креслом пассажира, и если на указанных местах есть какие-либо пассажиры, то нужно увеличить ответ на количество таких пассажиров. Асимптотика решения $O(k \cdot m)$, а так как m — это константа, то имеем асимптотику $O(k)$. Количество пассажиров не больше общего количества мест, поэтому $k \leqslant n \cdot m$ и итоговая асимптотика решения $O(n)$.

Подзадача 2

Текущую подзадачу можно решить аналогично подзадаче 1, но нужно создать массив размером n на m. Различие лишь в том, что в подзадаче 1 для каждого места в ряду можно было точечно проверить остальные места в том же ряду, а в текущей подзадаче лучше использовать цикл. Асимптотика решения $O(n \cdot m^2)$.

Подзадача 3

В текущей подзадаче можно отсортировать массив по номеру ряда, при этом для одного и того же ряда нужно сохранить порядок пассажиров, который был в исходном списке. Другими словами, если пассажиры i и j имеют кресла на одном ряду и в изначальном списке пассажир i идёт до пассажира j, то и после сортировки пассажир i должен идти до пассажира j.

Так как мы отсортировали пассажиров по номеру ряда, то мы можем пойти по отсортированному массиву и обрабатывать ряд за рядом, для каждого ряда эмулируя процесс посадки. То есть в отличие от предыдущих подзадач можно создать одномерных массив размером m.

Итоговая асимптотика решения $O(k \cdot \log k) + O(k \cdot m) = O(k \cdot (\log k + m))$.

Подзадача 4

Решение основывается на решении из подзадачи 3, но теперь нужно обрабатывать каждого пассажирова быстрее чем $O(m)$. Для этого можно воспользоваться, например, деревом отрезков и при обработке очередного пассажира вычислять количество пассажиров на отрезке в ряду за $O(\log m)$. В таком случае итоговая асимптотика будет $O(k \cdot \log k) + O(k \cdot \log m) = O(k \cdot (\log k + \log m))$.

Подзадача 5

Заметим, что задача сводится к задаче о подсчёте количества инверсий. В таком случае нам не нужно хранить данные о текущем обрабатываемом ряде в одномерном массиве размером m, а можно обойтись массивом размером, равном количеству пассажиров в текущем ряду. Асимптотика решения $O(k \cdot \log k)$.

Задача D. Максимальное покрытие

Для решения задачи на 30 баллов достаточно перебрать начало первого отрезка, после чего найти минимальное значение в каждом отрезке, взять максимум по посчитанным минимумам и выбрать найлучший вариант из всех способов начать первый отрезок.

Далее обсудим решение третьей подзадачи. В ней n делится на k. Это значит, что достаточно начало первого отрезка можно перебирать от 0 до $k - 1$, и, поскольку всего отрезков будет n/k, то вычислить значение искомой функции не составит труда, заранее предподсчитав минимум в каждом из отрезков длины k. Асимптотика такого решения $\mathcal{O}(n \log n)$ из-за предподсчета.

Для решения на полный балл заметим используется следующая идея. Давайте переберем позицию p короткого отрезка, если такой есть (то есть если n не делится на k). Теперь давайте заметим, что если переместить отрезок с позиции p на позицию $p + k$, то большинство отрезков останутся без изменений, а значит такой перенес можно обработать за $O(\log n)$, поддерживая текущие значения минимумов внутри каждого отрезка. Тогда если $p \ mod \ k$ фиксированно, то можно обработать все такие позиции p за $\mathcal{O}(n/k \log n)$. Тогда суммарно алгоритм работает за $\mathcal{O}(n \log n)$.

Задача E. Постройка домов

Подзадача 1

Так как количество различных чисел в масииве равно двум и $t = 1$, задача сводится к решению линейного диофантового уравнения: $brick_0 \cdot cnt_0 + brick_1 \cdot cnt_1 = q$, где $brick_0$ и $brick_1$ - уникальное количество кирпичей на складах, а cnt_i - количество нужных кирпичей $brick_i$. Посчитаем $brick_0$ и $brick_1$ с помощью словаря и переберем все возможные варианты. Асимптотика решения $O(n^2)$

Подзадача 2

Для решения подзадачи 2 достаточно рекурсивно перебрать все возможные суммы кирпечей со складов и для каждой суммы также рекурсивно перебрать сумму кирпечей для заказов, проверяя равны ли суммы. Если суммы оказались равны максимизируем ответ. Асимптотика решения $O(2^{n+t})$

```
n = int(input())
brick = list(map(int, input().split()))
t = int(input())
query = list(map(int, input().split()))
ans = 0

def f_query(t, sum_query, k, sum_brick, cnt):
    if (k == t):
        global ans
        if (sum_query == sum_brick):
            ans = max(ans, cnt)
        return
    f_query(t, sum_query + query[k], k + 1, sum_brick, cnt + 1)
    f_query(t, sum_query, k + 1, sum_brick, cnt)

def f_brick(n, sum_brick, k):
    if (k == n):
        global t
        if (sum_brick > 0):
            f_query(t, 0, 0, sum_brick, 0)
        return
    f_brick(n, sum_brick + brick[k], k + 1)
    f_brick(n, sum_brick, k + 1)

f_brick(n, 0, 0)
print(ans)
```

Подзадача 3

Для решения подзадачи 3 необходимо улучшить рекурсивный перебор, сохраняя значения в массиве dp_brick и dp_query. Массив dp_brick будем заполнять по следующему правилу: $dp\_brick[i] = 1$, если сумму i можно собрать используя какие-то кирпечи со складов

```
n = int(input())
brick = list(map(int, input().split()))
t = int(input())
query = list(map(int, input().split()))
W = 10000 + 1
dp_brick = [0] * W
dp_query = [0] * W
ans = 0
def f_tree(n, sum_brick, k):
    if (k == n):
        dp_brick[sum_brick] = 1
        return
    f_tree(n, sum_brick + brick[k], k + 1)
    f_tree(n, sum_brick, k + 1)

def f_query(t, sum_query, cnt, k):
    if (k == t):
```

```
        dp_query[sum_query] = max(dp_query[sum_query], cnt)
        if (dp_brick[sum_query]):
            global ans
            ans = max(ans, dp_query[sum_query])
        return
    f_query(t, sum_query + query[k], cnt + 1, k + 1)
    f_query(t, sum_query, cnt, k + 1)

f_tree(n, 0, 0)
f_query(t, 0, 0, 0)
print(ans)
```

Подзадача 4

Данная подзадача подразумевала за собой использование алгоритма «укладки рюкзака». $dp\_brick[i][j] = 1$, если сумму j можно собрать используя какие-то первые i кирпечей со складов из нашего массива $brick$. Соответственно пересчет нашей динамики будет: $dp\_brick[i][j] = dp\_brick[i-1][j]$ or $dp\_brick[i-1][j-brick[i]]$. Так как $t = 1$, в $dp\_brick[n][query_0]$ будет лежать наш ответ, где $query_0$ - это наш едиственный заказ. Асимптотика решения $O(n \cdot W)$.

Пересчет динамики:

```
for i in range(1, n + 1):
    for w in range(W):
        if (w - brick[i] >= 0):
            dp_brick[i][w] = dp_brick[i - 1][w - brick[i]] or dp_brick[i - 1][w]
        else:
            dp_brick[i][w] = dp_brick[i - 1][w]
```

Подзадача 5

Для решения этой подзадачи необходимо воспользоваться алгоритмом «укладки рюкзака со стоимостью». Соответственно $dp\_query[i][j]$ - максимальное количество заказов, которые мы можем выбрать и выполнить, чтобы набрать сумму j, используя какие-то первые i заказы. Единственное изменение алгоритма будет в пересчете динамики, так как нам необходимо максимизировать количество заказов, каждый раз жадно берем еще один заказ, если можем:

$$dp\_query[i][j] = max(dp\_query[i-1][j], dp\_query[i-1][j-query[i]] + 1)$$

Асимптотика решения $O(t \cdot W)$.

Пересчет динамики:

```
for i in range(1, t + 1):
    for w in range(W):
        if (w - query[i] >= 0):
            dp_query[i][w] = max(dp_query[i - 1][w], dp_query[i - 1][w - query[i]] + 1)
        else:
            dp_query[i][w] = dp_query[i - 1][w]
```

Подзадача 6

Для решения задачи на полный балл необходимо объединить решения подзадач 4 и 5. Для вычисления ответа пройтись по всем возможным суммам, если сумму i можно собрать используя заказы древесин и эту же сумму можно набрать, используя длины деревьев, максимизируем ответ. Асимптотика решения $O(W \cdot (n + t))$.

```
n = int(input())
brick = list(map(int, input().split()))
t = int(input())
query = list(map(int, input().split()))
brick = [0] + brick
```

```
query = [0] + query
W = 10001

dp_brick = [[0 for row in range(0, W)] for col in range(0, n + 1)]
dp_brick[0][0] = 1

dp_query = [[-W for row in range(0, W)] for col in range(0, t + 1)]
dp_query[0][0] = 0

for i in range(1, n + 1):
    for w in range(W):
        if (w - brick[i] >= 0):
            dp_brick[i][w] = dp_brick[i - 1][w - brick[i]] or dp_brick[i - 1][w]
        else:
            dp_brick[i][w] = dp_brick[i - 1][w]


for i in range(1, t + 1):
    for w in range(W):
        if (w - query[i] >= 0):
            dp_query[i][w] = max(dp_query[i - 1][w], dp_query[i - 1][w - query[i]] + 1)
        else:
            dp_query[i][w] = dp_query[i - 1][w]
ans = 0
for i in range(W):
    if (dp_brick[n][i] and dp_query[t][i]):
        ans = max(ans,dp_query[t][i])
print(ans)
```