Разбор задач

Задача 1. Крепость

Сложим количество охранников на всех сторонах, их bn. При этом охранников на башнях мы посчитали два раза, поэтому нужно вычесть an. Получится выражение $bn - an$, которое можно записать в виде `b * n - a * n`, но для того, чтобы минимизировать число операций, ответ должен быть таким:

```
(b - a) * n
```

Задача 2. Счастливые билеты

Ближайшие меньшие и ближайшие большие счастливые билеты:
826178 826187
866992 867399
281920 282039
199991 200002
304700 305008

Задача 3. Сортировка

Кратчайшее решение состоит из 5 строк:
231564
265134
156234
154326
123456

Задача 4. Час расплаты

Первые три числа можно разложить на суммы из 3 слагаемых. Последние два числа раскладываются на суммы из 4 слагаемых. Один из вариантов решения:
21 4 4
21 13 4
21 21 4
22 22 13 22
50 22 13 22

Задача 5. Путешествие поездом

Легко видеть, что номера мест в плацкартном купе можно найти с помощью следующих формул:

1. $(номер\_вагона - 1) * 4 + 1$;
2. $(номер\_вагона - 1) * 4 + 2$;
3. $(номер\_вагона - 1) * 4 + 3$;
4. $(номер\_вагона - 1) * 4 + 4$.

Схожим образом можно построить формулы и для вычисления номеров боковых мест. Для этого достаточно заметить, что номера боковых мест идут по убыванию, начиная с номера равного количеству мест в вагоне. Итоговые формулы для боковых мест:

5. $11 * 6 - (номер\_вагона - 1) * 2 - 1$;
6. $11 * 6 - (номер\_вагона - 1) * 2$.

Также возможно альтернативное решение — так как количество возможных тестов всего 11, то можно было в программе сохранить ответы для всех возможных тестов.

```
carriage = int(input())
print((carriage - 1) * 4 + 1)
print((carriage - 1) * 4 + 2)
print((carriage - 1) * 4 + 3)
print((carriage - 1) * 4 + 4)
print(11 * 6 - (carriage - 1) * 2 - 1)
print(11 * 6 - (carriage - 1) * 2)
```

Задача 6. Пирожки

Каждые три дня Петя будет тратить $a + b + c$ рублей вне зависимости от того, какой именно пирожок он купил первым. Поэтому можно сначала посчитать количество полных групп длительностью три дня, разделив нацело n на 3 (в Python $n//3$, в C++, C#, Java $n/3$, в Pascal n **div** 3) и вычислить сумму, потраченную за это время.

Остаток от деления целого числа (n) на 3 может принимать значения 0, 1 или 2 — это количество дней, не учтённых при вычислении суммы выше.

Таким образом, если остаток от деления составляет 0, то сумма уже известна.

Если остаток от деления равен 1, то к уже вычисленной сумме нужно добавить цену пирожка, который Петя купил в первый день.

Если же остаток от деления равен 2, то к уже вычисленной сумме нужно добавить цену пирожка, который Петя купил в первый день, и цену пирожка, который Петя купил во второй день.

Приведём одно из возможных решений (на Python):

```
a = int(input())
b = int(input())
c = int(input())
n = int(input())
start = int(input())

ans = n // 3 * (a + b + c)
if start == 2:
    a, b, c = b, c, a
elif start == 3:
    a, b, c = c, a, b

if n % 3 >= 1:
    ans += a
if n % 3 >= 2:
    ans += b
print(ans)
```

Решения, не использующие формулу для подсчёта суммы, потраченной за количество дней, кратное 3, а пошагово симулирующие процесс, не укладываются в ограничения по времени и набирают не более 72 баллов. Пример такого решения (на Python):

```
a = int(input())
b = int(input())
c = int(input())
n = int(input())
start = int(input())

if start == 2:
```

```
    a, b, c = b, c, a
elif start == 3:
    a, b, c = c, a, b

prices = [a, b, c]
ans = 0
for i in range(n):
    ans = ans + prices[i%3]
print(ans)
```

.

Задача 7. Игра

В данной задаче, вместо самого числа нам требуется его позиция среди всех чисел, ещё не стёртых с доски (вначале позиция числа равна самому числу). Алгоритм решения задачи следующий: пока на доске записано больше одного числа, в зависимости от чётности текущей позиции числа, стираем либо все чётные, либо все нечётные числа, и пересчитываем позицию числа, а также количество оставшихся чисел. Очевидно, что данный алгоритм корректен, ведь по сути он является симуляцией игрового процесса.

Также несложно заметить, что за один ход мы уменьшаем количество чисел в два раза, а значит асимптотическая сложность программы будет равна $O(\log N)$.

```
n = int(input())
position = int(input())
while n > 1:
    if position % 2 == 1:
        print(2)
        n = (n + 1) // 2
        position = (position + 1) // 2
    else:
        print(1)
        n = n // 2
        position = position // 2
```

Чтобы набрать баллы за тесты $3 - 10$, можно написать полностью симуляционное решение поместив числа, записанные на доске, в массив (список) и явно удаляя их из массива при каждой операции стирания. Такое решение будет иметь сложность $O(N)$.

```
n = int(input())
position = int(input())
numbers_on_board = list(range(1, n + 1))
while len(lst) > 1:
    i = numbers_on_board.index(position)
    if i % 2 == 0:
        print(2)
        del numbers_on_board[1::2]
    else:
        print(1)
        del numbers_on_board[::2]
```

Для прохождения 11 теста достаточно на каждом ходу стирать числа на чётных позициях.

Для прохождения тестов $12 - 14$ достаточно заметить, что чётность стираемых на каждом ходу чисел должна отличаться от чётности оставшегося количества чисел. Решение для данной группы тестов могло натолкнуть участников на идею полного решения.