Задача 1. Летоисчисление

Для начала просто прибавим к году первого события A число n. Мы получим ответ, не учитывающий отсутствие нулевого года. Чтобы учесть наличие нулевого года, необходимо в некоторых случаях прибавить к ответу 1 или вычесть из ответа 1.

```
a = int(input())
n = int(input())
b = a + n
if a < 0 and b >= 0:
    b += 1
elif a > 0 and b <= 0:
    b -= 1
print(b)
```

Задача 2. Прожектора

Если ресурс первого прожектора равен a, а ресурс остальных прожекторов неограничен, то прожектора смогут гореть $4a$ секунд.

Если ресурс второго прожектора равен b, а ресурсы остальных прожекторов неограничены, то время горения будет $2b + 1$.

Наконец, если ограничено только время горения третьего прожектора c, то ответ будет $4c + 2$.

Необходимо вывести наименьшее из этих величин.

```
a = int(input())
b = int(input())
c = int(input())
print(min(4 * a, 2 * b + 1, 4 * c + 2))
```

Задача 3. Ремонт забора

В задаче нужно последовательность из N единиц и нулей “накрыть” отрезками длины не более L так, чтобы были “накрыты” все единицы.

Это можно сделать простым жадным алгоритмом: найдём первую единицу и будем считать, что очередной отрезок длины L начинается с этой единицы, то есть можно пропустить следующие $L - 1$ элемент массива: если единица была встречена на позиции i, то можно продолжить просмотр последовательности начиная со значения $i + L$. В эффективном решении (на полный балл) задача решается за один проход по массиву, то есть с использованием одного цикла. Если в решении есть вложенные циклы, то оно может не уложиться по времени в ограничение 1 секунда и наберёт неполный балл.

Вот пример правильного и эффективного решения:

```
L = int(input())
N = int(input())
A = []
for i in range(N):
    A.append(int(input()))
ans = 0
i = 0
while i < len(A):
    if A[i] == 1:
        ans += 1
        i += L
    else:
        i += 1
print(ans)
```

Задачу можно решать и без использования массива, то есть необязательно сохранять в памяти все считанные числа. Достаточно запоминать номер элемента последовательности, являющийся началом последнего отрезка длины L (`last_start`). Тогда следующий отрезок начинается, если была считана единица и номер текущего считанного числа больше или равен, чем `last_start`$+L$. Пример такого решения:

```
L = int(input())
N = int(input())
ans = 0
last_start = -10 ** 6
for i in range(N):
    fence_elem = int(input())
    if fence_elem == 1 and i >= last_start + L:
        ans += 1
        last_start = i
print(ans)
```

Задача 4. Американские горки

Для начала можно попробовать перебрать все возможные ответы, то есть все возможные начальные и конечные точки выбранного участка, при помощи двух вложенных циклов. Затем найдём максимальное значение высоты на этом участке, для чего понадобится ещё один цикл (в приведённом ниже решении вместо третьего цикла используется функция max для среза списка, то есть для рассматриваемого фрагмента). Если значение максимума на участке строго больше значений крайних элементов, то этот участок удовлетворяет условию задачи. Осталось только выбрать самый короткий участок из таких (то есть участок, для которого минимальная разница для индексов конца и начала участка). Такое решение имеет алгоритмическую сложность $O(N^3)$, то есть время работы такой программы растёт пропорционально N^3 и набирает 40 баллов.

Пример решения на языке Python (40 баллов)

```
n = int(input())
a = [int(input()) for i in range(n)]
ans1 = 0
ans2 = 10 ** 9
for i in range(0, n):
   for j in range(i + 2, n):
       m = max(a[i:j + 1])
       if a[i] < m > a[j] and j - i <= ans2 - ans1:
           ans2 = j
           ans1 = i
if ans2 != 10 ** 9:
   print(ans1 + 1, ans2 + 1)
else:
   print(0)
```

Сложность этого решения можно уменьшить до $O(N^2)$, если максимум на отрезке считать быстрее. Для этого можно заметить, что при увеличении правой границы j на 1 к рассматриваемому отрезку добавляется один элемент, поэтому можно хранить текущее значение максимума на отрезке и при увеличении j новый добавляемый к отрезку элемент сравнивать с максимумом, обновляя максимум при необходимости. Но такое решение тоже не получает максимальный балл.

Для того, чтобы придумать правильное и эффективное решение заметим, что ответ (высоты наилучшего участка) всегда имеет вид, как в примере (15, 20, 20, 10) — все значения кроме двух крайних равны, а два крайних меньше их. Иначе можно сократить участок, оставив только два крайних значения, меньших наибольшего значения на участке. Поэтому можно изучать только последовательности равных идущих подряд элементов массива, среди всех таких последовательностей

нужно выбрать последовательность наименьшей длины, причём перед и после этой последовательности должны быть меньшие значения. При этом алгоритм должен иметь сложность $O(N)$, иначе программа также не сможет уложиться в ограничение по времени в 1 секунду.

Решение можно реализовать и без использования массива для хранения всех данных чисел. Будем считывать числа по одному, при этом будем использовать следущие переменные:
`last_height` — значение последнего считанного элемента.
`last_len` — количество последних считанных элементов, равных `last_height`.
`prev_height` — значение элемента, который был перед последовательностью последних элементов равных `last_height`.

Пусть значение нового считанного элемента равно `h`. Если `h = last_height`, то новый элемент продолжает последовательность элементов равных `last_height`, поэтому увеличим значение `last_len` на 1. Если же новый элемент неравен `last_height`, то этот элемент начинает новуй последовательность равных элементов, поэтому запишем его значение в переменную `last_height`, переменной `last_len` присвоим значение 1, а старое значение `last_height` переместится в переменную `prev_height`. Но сначала проверим, является ли последовательность элементов равных `last_height`, удовлетворяющей условию задачи, что означает выполнение неравенств `prev_height < last_height > h`. Если эти неравенства верны, а также если длина последовательности `last_len` наименьшая (для этого будем также хранить наименьшую известную длину последовательности, удовлетворяющую условию задачи в переменной `best_len`), то запомним ответ и обновим значение переменной `best_len`.

Пример решения на языке Python (100 баллов)

```
n = int(input())
ans1 = 0
ans2 = 0
last_height = 10 ** 9
prev_height = 10 ** 9
last_len = 0
best_len = 10 ** 9
for i in range(1, n + 1):
   h = int(input())
   if h == last_height:
       last_len += 1
   else:
       if prev_height < last_height > h and last_len < best_len:
           best_len = last_len
           ans1 = i - last_len - 1
           ans2 = i
       prev_height = last_height
       last_height = h
       last_len = 1
if ans1 == 0:
   print(0)
else:
   print(ans1, ans2)
```

Задача 5. Числа

Решение, в котором перебираются все числа от A до B и для каждого из них подсчитывается сумма цифр, не пройдет тесты при больших A и B ввиду наличия ограничения по времени, но может набрать частичные баллы.

Для полного решения задачи нужно заметить, что в каждом десятке (числа, отличающиеся только последней цифрой) ровно 5 чисел имеют нечётную сумму цифр. Поэтому можно сначала циклом пройти от числа A до ближайшего числа, заканчивающегося цифрой 0, для каждого из этих

чисел посчитать сумму цифр и увеличивать ответ на 1, если сумма цифр числа нечётная. Затем посчитать количество полных десятков до числа B, добавив к ответу это количество, умноженное на 5. Наконец, перебрать числа, начиная с последнего числа последнего десятка, увеличенного на 1, до B, и также найти сколько среди них имеет нечётную сумму цифр.

Пример правильного решения на языке Python версии 3:

```
def check(n): # Проверка четности суммы цифр числа n
    s = 0
    while n > 0:
        s += n % 10
        n //= 10
    return s % 2

A = int(input())
B = int(input())

ans = 0
while A <= B and A % 10 != 0:
    ans += check(A)
    A += 1

if A < B:
    ans += (B - A) // 10 * 5
    A += (B - A) // 10 * 10

while A <= B:
    ans += check(A)
    A += 1

print(ans)
```