Задача A. Двоичный поиск

В данной задаче можно реализовать свой двоичный поиск, либо воспользоваться функцией `lower_bound` в случае C++, либо функцией `bisect_left` в случае Python.

Задача B. Быстрый поиск в массиве

Пусть x — позиция первого элемента в массиве, такого что $a_x \geqslant l$, а y — позиция первого элемента в массиве, такого что $a_y > r$. Тогда ответ на запрос равен $y - x$.

Данные позиции можно найти при помощи функций `lower_bound` и `upper_bound` в случае C++, либо функций `bisect_left` и `bisect_right` в случае Python.

Задача C. Очень Легкая Задача

Если $n = 1$, то минимальное время равно $\min(x, y)$. В противном случае мы можем сделать первую копию за время $\min(x, y)$, а остальные $(n - 1)$ копий можно будет сделать, параллельно используя два принтера.

Таким образом, теперь нужно решить задачу для $(n-1)$ копий. Научимся проверять, можно ли успеть сделать необходимое количество копий за время t. Первый ксерокс успеет сделать $\left\lfloor \frac{t}{x} \right\rfloor$ копий, а второй — $\left\lfloor \frac{t}{y} \right\rfloor$ копий. Таким образом, для проверки необходимо сравнить сумму этих двух величин с числом копий $n - 1$.

Очевидно, что данная функция монотонна по t, поэтому ответ можно искать при помощи двоичного поиска. В качестве левой границы можно выбрать число 0, так как за нулевое время невозможно сделать ни одной копии. В качестве правой границы можно выбрать число $\min(x, y) \cdot (n - 1)$, так как за это время, пользуясь лишь одним ксероксом, можно сделать все копии.

Задача D. Дипломы

Сделаем бинпоиск по ответу (по стороне длины доски) — понятно, что если подходит сторона длиной a, то и большие стороны тоже подходят. Чтобы проверить, что длина a подходит, достаточно жадно ставить дипломы. Получится, что $\left\lfloor \frac{a}{w} \right\rfloor \cdot \left\lfloor \frac{a}{h} \right\rfloor$ (максимальное количество дипломов, которые мы в приницпе можем уместить в квадрат со стороной a) должно быть не меньше n.

Задача E. Коровы в стойла

Будем использовать алгоритм двоичного поиска по ответу. За каждое предполагаемое расстояние между коровами m будем пробовать расставить коров таким образом, чтобы расстояние между ними было больше либо равно, чем m. Если мы смогли расставить всех коров, нужно увеличить расстояние m, если не смогли расставить всех, то уменьшаем его.

Для проверки достаточно поставить первую корову в стойло с минимальной координатой. Вторую корову необходимо поставить в самое левое стойло, находящееся на расстоянии, не меньшем m, от первой коровы. И так далее. Таким образом, при фиксированном m проверка выполняется за $\mathcal{O}(n)$.

Задача F. Приближенный бинарный поиск

Воспользуемся функцией `lower_bound` в случае C++ и функцией `bisect_left` в случае Python. Данные функции найдут минимальный элемент в массиве, больше либо равный, чем искомое число. После этого посмотрим на найденный элемент и предыдущий элемент, сравним их и выберем в качестве ответа тот, который ближе к искомому числу.

Задача G. Подпоследовательность и подмассив

Для решения задачи на частичный балл достаточно научиться проверять, является ли один массив подпоследовательностью другого массива.

Пусть мы хотим проверить, что массив a_2 является подпоследовательностью массива a_1. Для начала найдем в массиве a_1 элемент с минимальным индексом, равный первому элементу массива a_2. Обозначим индекс найденного элемента как p_1. Далее найдем в массиве a_1 элемент с минимальным индексом, большим p_1, который равен второму элементу массива a_2. Обозначим индекс найденного элемента как p_2. Продолжим по аналогии для остальных элементов. Если в какой-то

момент времени не удается найти необходимый элемент в массиве a_1, то массив a_2 не является подпоследовательностью a_1. В противном случае подпоследовательность будет найдена.

Теперь для ответа на каждый запрос в явном виде выполним данный алгоритм. Сложность такого решения: $\mathcal{O}(Q \cdot N + \sum m_i)$. Конечно, при больших N и Q данное решение будет работать слишком долго.

Теперь рассмотрим решение данной задачи на полный балл. Очевидно, что нужно научиться выполнять операцию «найти в массиве a элемент с минимальным индексом, большим, чем p, равный x» быстрее, чем за линейный проход. Для начала запомним для каждого элемента x список таких индексов $i_1, i_2, \ldots, i_k$, что $a[i_j] = x$. Это можно сделать во время считывания массива a.

Теперь научимся выполнять требуемую операцию быстрее. Посмотрим в список индексов вхождения числа x в массив a. В нем требуется найти минимальный индекс, больший, чем p. Это можно выполнить при помощи бинарного поиска за $\mathcal{O}(\log N)$. В языке C++ это можно сделать при помощи встроенной функции `upper_bound`.

Таким образом, решение выглядит так: для каждого запроса пройдемся по элементам массива b_i, и каждый раз будем искать первое вхождение текущего элемента в массив a, индекс которого больше, чем p. Изначально $p = l_i - 1$. Если в какой-то момент времени p станет больше, чем r_i, то найти требуемую подпоследовательность невозможно.

Сложность: $\mathcal{O}(N + (\sum m_i) \log N)$.

Задача H. Медиана объединений

Сделаем двоичный поиск по ответу. Пусть зафиксировано значение медианы m. Необходимо найти, сколько есть элементов, не больших m, в обеих последовательностях, и в зависимости от их количества сдвинуть границы.

Для того, чтобы найти количество элементов, не больших m, в отсортированной последовательности, достаточно найти первый элемент, строго больший m, и взять в качестве ответа его индекс. Для этого можно воспользоваться функциями `upper_bound` в случае C++ и `bisect_right` в случае Python.

Таким образом, мы сможем найти искомое количество элементов за время $\mathcal{O}(\log n_i + \log n_j)$. Итоговое время работы алгоритма равно $\mathcal{O}(n^2 \log C \log n)$, где $n = n_1 + n_2 + \ldots$

Задача I. Провода

Данная задача была разобрана на лекции.

Задача J. Среди Них

Найдем количество раундов игры при помощи двоичного поиска по ответу. Пусть двоичный поиск зафиксировал некоторое число k, необходимо определить, было ли в игре хотя бы k раундов.

Если $n \leqslant k$, то количество раундов в игре не больше, чем k, так как всего есть n предателей, а на каждом ходу один предатель удаляется из игры.

В противном случае нужно понять, выживет ли хотя бы один член экипажа спустя k раундов. После первого раунда будут удалены n членов экипажа. После второго раунда будут удалены $(n-1)$ членов экипажа. После k-го раунда игры будут удалены $(n-k+1)$ членов экипажа. Сумма данных чисел образует арифметическую прогрессию и может быть вычислена за $\mathcal{O}(1)$. Далее необходимо сравнить данную сумму с числом m.

Наконец, в конце осталось определить, кто выиграет в данной игре. Для этого, зная количество ходов, нужно понять, сколько членов экипажа останется в конце. Если это число отлично от нуля, то члены экипажа победят. В противном случае победят предатели.