1. **Рюкзак на отрезке**
   Идем в оффлайн правой границей, поддерживаем для каждого веса самое правое l, при котором этот вес все еще можно набрать при данной правой границе. `max_left[right, weight]`.

2. **Мосты**
   У нас есть x брёвен длины a и y брёвен длины b.
   Сделаем бинпоиск по ответу. Пытаемся построить мост из k рядов размера m.
   Решение `#1`: $\langle k, len \rangle$`[x,y]` – максимальная пара "количество рядов, длина последнего", от того, сколько рёбер мы уже использовали. Два перехода из состояния.
   Решение `#2`: `x[y,k]` – сколько максимум брёвен 1-го типа осталось, если осталось y брёвен второго типа, и мы уже построили k рядов. Переход: перебрать сколько берём брёвен 1-го типа, 2-го добрать жадно. Всего $\frac{m}{a}$ переходов из каждого состояния. Заметим, что и в состоянии, и в переходе можно было 1-й тип поменять со 2-м. Итого состояний $\min(x, y) \cdot k$ и переходов $\frac{m}{\max(a,b)}$. При этом $m \le \frac{ax+by}{k}$, получаем не более $\frac{x+y}{k}$ переходов и время внутри бинпоиска $\mathcal{O}(N^2)$.

3. **Четыре множества**
   (а) $n \le 50$. Динамика `ans[i,x1,x2,x3]` — можно ли первые i чисел разбить на 4 множества так, чтобы `xor` 1-го множества был равен $x1$ и т.д.
   При этом $x4 = X \oplus x1 \oplus x2 \oplus x3$, где X — `xor` всех чисел от первого до i-го.
   (б) $n \le 500$. Битовое сжатие.

4. **Казино**
   (а) $\mathcal{O}(n^4)$ времени и $\mathcal{O}(n^3)$ памяти. Динамика `f[l,r]` — можно ли уничтожить все символы с l по r, а также динамика `f[l,r,i,k]` — можно ли уничтожить некоторые символы на отрезке с l по r таким образом, чтобы остались первые k букв правила i.
   (б) $\mathcal{O}(n^4/w)$ времени и $\mathcal{O}(n^3/w)$ памяти. Битовое сжатие.

5. **LCIS**
   Будем добавлять элементы первой последовательности по одному, а для каждой позиции второй последовательности будем поддерживать длину максимальной `LCIS`, заканчивающейся в этом элементе.

6. **Пути в графе**
   (а) ровно k: возвести матрицу смежности в степень.
   (б) не более k: или добавить фиктивную вершину, или взять сумму геометрической прогрессии матриц.

7. **k-я лексикографически скобочная последовательность**
   $\mathcal{O}(n^2 m)$ — стандартная динамика, для ответа перебираем, какую скобку ставить. Заметим, что мы можем поставить либо одну закрывающую (вершину стека), либо открывающую, но для каждой открывающей число способов одинаково, так что делим с остатком, получаем $\mathcal{O}(n^2)$.

8. **Диаграммы Юнга**
   Будем расставлять числа, начиная с самых больших. Заметим, что если мы поставили несколько максимальных чисел, то оставшаяся картинка — тоже диаграмма Юнга, где расставлены числа от 1 до x, это и будет состоянием динамики. Переход: перебираем, в какой строке будет максимальное число. Число состояний не превосходит число разбиений на слагаемые.

9. **Чёрно-белый прямоугольник**
   Будем перебирать все левые верхние углы. Пусть текущий ответ равен ans, будем увеличивать ans, пока подходит квадрат со стороной $ans + 1$.

10. **Посёлки**
    Нужно массив длины n посплитить на k отрезков.
    Сделаем предподсчёт за $\mathcal{O}(n^2)$: $c[l, r]$ – стоимость отрезка $[l, r]$.
    Динамика $f[n, k]$ – стоимость разбить первые n элементов на k отрезков. $p[n, k]$ – граница последнего из k отрезков. Нужно доказать, что $p[n-1, k] \le p[n, k] \le p[n, k-1]$. И при подсчёта очередного $p[n, k]$, пользуясь уже посчитанными $p[n-1, k]$, $p[n, k-1]$ перебирать только в диапазоне. Это $\mathcal{O}(n^2)$.

11. **Найти самый большой gap**
    Решите её сами, это прикольно `:)`

12. **Лексикографически минимальный путь**
    (а) $\mathcal{O}(VE)$: делаем слоистый bfs, в очереди лежат вершины, до которых расстояние ровно k и от начала одинаковый лексикографически минимальный путь.
    (б) $\mathcal{O}(E \log V)$: для каждой вершины будем искать минимальный лексикографический путь до конца из этой вершины. Для вычисления требуется выбрать все переходы по минимальной букве, а далее уметь сравнивать строки. Заметим, что у каждой уже обработанной вершины есть однозначная следующая (куда надо идти), этот граф является деревом. Для сравнения строк нужно в этом дереве считать хеш на пути вверх, в дерево добавляются листья. Это всё можно делать двоичными подъемами с хешами.