Задача 1A. Набридли довгі умови задач?

Автор: **Фейса Богдан**
Підготували: **Фейса Богдан**
Розбір: **Фейса Богдан**

Блок 1: $a_1 = a_2 = \cdots = a_n$;

Якщо $a_1 = a_2 = \cdots = a_n$, для кожної перестановки a $f(a)$ дорівнює 0. Таким чином, відповідь завжди 0 в цьому блоці.

Блок 2: $n \leqslant 9$;

Для $n \leqslant 9$ ми можемо перебрати всі перестановки a і вручну обчислити $f(a)$ з часовою складністю $O(n)$. Загальна часова складність $O(n! \cdot n)$.

Блок 3: $n \leqslant 500$;

Для $n \leqslant 500$ ми можемо побачити, що для кожної фіксованої трійки i, j, k $(i < j < k)$ оптимально, щоб ці елементи в a були впорядковані за зростанням $(a_i \leqslant a_j \leqslant a_k)$ або за спаданням $(a_i \geqslant a_j \geqslant a_k)$.

Доказ:

Припустимо, що ми хочемо мінімізувати

$$(x - y)^2 + (y - z)^2$$

$$(x - y)^2 + (y - z)^2 = x^2 + y^2 - 2xy + y^2 + z^2 - 2yz$$

Ми маємо максимізувати $2xy + 2yz$.

Для будь-якого заданого x, y що $x + y = A$, максимальне значення xy дорівнює $\frac{A^2}{4}$

Доказ: Для будь-якого дійсного q:

$$\frac{A^2}{4} \geqslant (\frac{A}{2} - q)(\frac{A}{2} + q) = \frac{A^2}{4} - q^2$$

Іншими словами, $x - y$ повинно бути мінімальним.

Отже, щоб максимізувати $2xy + 2yz$ умова $(x \leqslant y \leqslant z)$ повинна бути виконана.

У масиві є не більше десяти різних a_i;

Для цього блоку ми повинні поєднати ідеї попередніх блоків. Загальна часова складність для тесту, в якому є k різних значень в a, становить $O(k! \cdot k)$.

Блок 4: $n \leqslant 10,000$;

Ми повинні взяти факт з блоку 3 і перефразувати його:

Не повинно бути локальних максимумів або мінімумів.

Ми повинні почати з другого елемента і виконати такі дії для кожного j $(2 \leqslant j \leqslant n)$:

- якщо $a_{j-1} > a_j$ ми міняємо місцями елементи на позиціях j і $j - 1$.

Ми повторюємо цей алгоритм, поки не буде локальних максимумів і мінімумів. Можна довести, що найгірша часова складність цього алгоритму $O(n^2)$.

Блок 5: без додаткових обмежень.

Алгоритм з блоку 4 призводить до впорядкованого масиву, тому використовуючи ефективний алгоритм сортування, ми можемо досягти часової складності $O(n \cdot \log(n))$.

Задача 1B. Дивні запити

Автор: **Тимкович Олександр**
Підготували: **Тимкович Олександр, Ціцей Павло**
Розбір: **Тимкович Олександр**

Блок 1: $p = [1, 2, \ldots, n]$.

У цьому блоці можна помітити, що $p_i^k = p_i$, отже відповіддю на кожен запит є $\max\limits_{1 \leqslant i \leqslant n} (p_i + i)$.

Блок 2: $q \leqslant 10$ і $1 \leqslant k \leqslant 100$.

У цьому блоці задумано знайти p_i^k за $O(k)$ для конкретного i. Загальна складність буде $O(q \cdot n \cdot k)$.

Блок 3: $1 \leqslant k \leqslant 100$.

У цьому блоці ми можемо попередньо обчислити p_i^k для кожного $1 \leqslant i \leqslant n$ за $O(n \cdot k)$. Потім попередньо обчислити відповідь для кожного k і відповідати на запити за $O(1)$.

Блок 4: для кожної пари $1 \leqslant i, j \leqslant n$ існує k, що $p_i^k = j$.

У цьому блоці задумано розглядати перестановку як граф. Перестановка утворює єдиний цикл довжиною n. Отже, ми можемо замінити k на $k \bmod n$ і зробити те саме, що і в блоці 3.

Блок 5: $n \leqslant 20$.

Тепер ми можемо обробляти кожен цикл незалежно. У кожному циклі ми можемо замінити k на $k \bmod \ell$, де ℓ - це довжина поточного циклу. Потім, ідея та ж, що і в блоці 2. Загальна складність складає $O(q \cdot n^2)$.

Блок 6: $q \leqslant 10^4$.

Для кожного циклу в перестановці ми можемо попередньо обчислити максимум для кожного k за $O(n^2)$. Потім відповідати на кожен запит за $O(n)$.

Блок 7: немає додаткових обмежень.

Спочатку робимо те ж попереднє обчислення, що і в блоці 6. Також ми можемо групувати цикли з однаковою довжиною, беручи максимальне значення для кожного $k \bmod \ell$.

Яка верхня межа кількості різних довжин циклів у перестановці довжини n? Довжина перестановки дорівнює сумі довжин циклів. Розглянемо суму різних довжин циклів. Припустимо, що ці довжини є $1, 2, \ldots, x$. Тоді сума дорівнює $1 + 2 + \cdots + x = \frac{x(x+1)}{2}$. Для $x \approx \sqrt{n}$ сума вже більша за n. Це доводить, що існує не більше ніж приблизно $\sqrt{n}$ різних довжин циклів.

Отже, ми можемо відповідати на кожен запит за $O(\sqrt{n})$. Загальна складність складає $O(n^2 + q\sqrt{n})$.

Задача 1C. І знову ж таки, запити

Автор: **Тимкович Олександр**
Підготували: **Тимкович Олександр, Фейса Богдан**
Розбір: **Тимкович Олександр**

Блок 1: $q = 1$, $l = 1$ та $r = n$.

У цьому блоці ми маємо знайти відповідь для всього масиву. Жадібна ідея працює! Ми можемо спробувати взяти якомога більше елементів у певний сегмент.

Блок 2: $k = 1$.

Обмеження в цьому блоці означають, що в кожному сегменті є рівно один різний елемент. Ми можемо відповідати на запити, використовуючи дерево відрізків на заданому масиві. Для кожної вершини будемо зберігати відповідь для сегмента, перший та останній елементи у сегменті. Для злиття двох вершин можна порівняти останній елемент лівого сегмента та перший елемент правого сегмента.

Блок 3: $l = 1$ для всіх запитів.

Ми можемо використати ідею з блоку 1 і зберегти відповідь для кожного префікса. Потім відповідати на запити за $O(1)$.

Блок 4: $a_i \leqslant 10^5$, $1 \leqslant n \leqslant 10^5$.

Обчислимо для кожного індексу $1 \leqslant i \leqslant n$ найбільше j $(i \leqslant j \leqslant n)$ таке, що між $a_i, a_{i+1}, \ldots, a_j$ є не більше k різних елементів. Назвемо це j як x_i. Це можна зробити за $O(n \log n \log A)$, використовуючи дерево Фенвіка з бінарним пошуком.

Тепер відповідь на запит - це кількість переходів типу $i \to x_i$, починаючи з l і закінчуючи в r. Тут можна використовувати техніку бінарних стрибків. $up_{i,k}$ - це позиція, в якій ми опинимося після 2^k стрибків, починаючи з i. Це можна попередньо обчислити за $O(n \log n)$. Також легко відповідати на запити, використовуючи це попереднє обчислення, за $O(\log n)$ для кожного запиту.

Загальна складність - $O(n \log n \log A + q \log n)$.

Блок 5: $a_i \leqslant 10^5$, $1 \leqslant n \leqslant 10^5$.

У цьому блоці задумано ту ж саму ідею, що і в блоці 4, але попереднє обчислення x_i можна зробити за $O(n \log A)$, використовуючи два вказівники або дерево відрізків.

Блок 6: $a_i \leqslant 10^5$, $1 \leqslant n \leqslant 10^5$.

Ідея та ж, що і в блоці 5. Можна стиснути цілі числа масиву, щоб отримати попереднє обчислення за $O(n \log n)$.

Задача 1D. Проста задача?

Автор: **Ціцей Павло**
Підготував: **Ціцей Павло**
Розбір: **Ціцей Павло**

Визначення:

Позначимо $[a, b, c]$ як кількість підрядків непарної довжини з $s_l = a$, $s_m = b$, $s_r = c$. Таким чином, відповідь - це $[0, 0, 0]+[1, 1, 1]$. Також позначимо $[all]$ як усі підрядки непарної довжини.

Блок 1: В рядку немає нулів:

Тут усі підрядки непарної довжини є хорошими підрядками. Таким чином, ми можемо пройтися по рядку та легко підрахувати кількість таких підрядків. Розв'язок працює за $O(n)$.

Блок 2: $n \leqslant 100$:

Ми проходимо усі можливі трійки l, r, k та перевіряємо, чи $k = \frac{l+r}{2}$, його довжина непарна, $l < r$ та $s_l = s_r = s_k$. Розв'язок працює за $O(n^3)$.

Блок 3: $n \leqslant 10^3$:

Розширимо розв'язок для другого блоку. Якщо ми фіксуємо лише l, r та $k = \frac{l+r}{2}$. Тоді ми просто перевіряємо, чи підрядок має непарну довжину, $l < r$ та $s_l = s_r = s_k$. Розв'язок працює за $O(n^2)$.

Блок 4: Рядок містить не більше 10^3 одиниць:

Відповідь - це $[0, 0, 0] + [1, 1, 1] = [all] - [1, 0, 0] - [0, 1, 0] - [0, 0, 1] - [1, 0, 1] - [1, 1, 0] - [0, 1, 1]$. Ми можемо легко знайти $[all]$ (перший блок). Давайте фіксуємо лише одну "1" та знайдемо кількість підрядків з цим. Це буде $[1, 0, 0] + [0, 1, 0] + [0, 0, 1] + 2 \cdot [1, 0, 1] + 2 \cdot [1, 1, 0] + 2 \cdot [0, 1, 1] + 3 \cdot [1, 1, 1]$. Ми можемо знайти $[1, 0, 1]$, $[1, 1, 0]$, $[0, 1, 1]$, $[1, 1, 1]$ за $O(n^2)$ просто фіксуючи дві "1". Таким чином, ми можемо звести рівняння до $[1, 0, 0] + [0, 1, 0] + [0, 0, 1] + [1, 0, 1] + [1, 1, 0] + [0, 1, 1]$ та відняти від $[all]$ та отримати відповідь. Загальна складність $O(n^2)$.

Блок 5: Без додаткових обмежень:

Ми можемо фіксувати деякі два елементи та знайти кількість підрядків з цим в $O(n)$. Але ми не можемо вказати третій елемент. Таким чином, якщо ми записуємо деякі можливі варіанти, ми можемо отримати такі змінні:

$[1, 0, 1] + [1, 1, 1]$ (фіксуємо лівий та правий елемент на "1")
$[0, 1, 1] + [1, 1, 1]$ (фіксуємо середній та правий елемент на "1")
$[1, 1, 0] + [1, 1, 1]$ (фіксуємо лівий та середній елемент на "1")
$[0, 0, 0] + [1, 0, 0]$ (фіксуємо середній та правий елемент на "0")
$[0, 0, 0] + [0, 1, 0]$ (фіксуємо лівий та правий елемент на "0")
$[0, 0, 0] + [0, 0, 1]$ (фіксуємо лівий та середній елемент на "0").

Легко бачити, що сума всіх це $[all] + 2 * ([0, 0, 0] + [1, 1, 1])$. Таким чином, ми просто віднімаємо $[all]$, ділимо на 2 та отримуємо відповідь. Розв'язок працює за $O(n)$.

Задача 2A. Трішки пригод нікому не завадить

Автор: **Тимкович Олександр**
Підготував: **Тимкович Олександр**
Розбір: **Тимкович Олександр**

Формально, умова говорить, що вечірка погана, коли на відрізку існує елемент, який є підмаскою будь-якого іншого елемента на відрізку. Щоб перевірити, чи є число x підмаскою y, можна перевірити, чи x OR $y = y$, де OR - це побітова операція.

Блок 1: $n \leqslant 40$.

У цьому блоці ми можемо спробувати перевірити кожен відрізок. На кожному відрізку ми намагаємось знайти погану особу. Складність становить $O(n^4)$.

Блок 2: $n \leqslant 100$.

Зверніть увагу, що погана особа завжди має мінімальне значення, оскільки вона повинна бути підмаскою всіх інших елементів. Тому ми можемо відстежувати можливу погану особу. Складність становить $O(n^3)$.

Блок 3: $a_i \in \{0, 1\}$.

Зверніть увагу, що тут на кожному відрізку існує погана особа.

Блок 4: без додаткових обмежень.

Ми знаємо, що погана особа, можливо, є мінімумом на відрізку. Також ми знаємо, що вона повинна бути підмаскою всіх інших елементів на відрізку. Спостереження полягає в тому, що мінімум повинен дорівнювати побітовому AND всіх елементів на відрізку.

Тому ми можемо відстежувати мінімум і побітовий AND для кожного відрізка. Складність становить $O(n^2)$.

Задача 2B. Трішки обману ніколи не зашкодить

Автор: **Ціцей Павло**
Підготував: **Ціцей Павло**
Розбір: **Ціцей Павло**

Блок 1: $k = 0$:

Ми не можемо виконати жодну операцію, тому нам потрібно знайти максимальну суму підмасиву. Ми можемо знайти його, використовуючи префіксну суму. Задача зводиться до пошуку i, j таких, що $j < i$ та $a_i - a_j$ є максимальним. Якщо ми фіксуємо i, то j - це елемент, для якого a_j є мінімальним на префіксі від 0 до i. Це можна зробити за допомогою структури "set"в STL або звичайної черги. Рішення працює за $O(n \cdot log(n))$.

Блок 2: $k = 1$:

Тут ми можемо виконати лише одну операцію. Тому ми знаходимо найлівіший максимальний підмасив і виконуємо операцію на найправіший елемент. Це найкращий елемент, оскільки він розглядає всі елементи, які перетинаються з цим максимальним підмасивом, тому якщо всі підмасиви перетинаються, то відповідь буде зменшена на 1. Рішення працює за $O(n \cdot log(n))$, оскільки використовує два рази алгоритм з першого блоку.

Блок 3: $p_i \geqslant 0$:

Тут максимальна сума підмасиву - це весь масив, тому ми просто виконуємо всі операції на додатніх цілих числах, поки вони існують. Якщо їх більше, то ми бачимо, що максимальний підмасив - це деякий елемент, оскільки $p_i \leqslant 0$. Тому нам просто потрібно мінімізувати максимальний елемент, що ми можемо зробити, зменшуючи кожен елемент на 1. Таким чином, відповідь буде $\frac{n}{k}$, де k - залишена кількість операцій. Рішення працює за $O(n)$.

Блок 4: $n \leqslant 1\,000$:

Тут ми будемо припускати, що ми не знаємо, як знайти максимальну суму підмасиву за $O(n \cdot log(n))$, але за $O(n^2)$ просто перебираючи всі підмасиви. Ми можемо використовувати бінарний пошук для відповіді. Якщо ми можемо отримати максимальну суму підмасиву менше або рівну x, то очевидно, що ми можемо отримати її менше або рівну y, коли $x < y$. Також, якщо ми не можемо отримати максимальну суму підмасиву менше або рівну x, то ми не зможемо отримати її менше або рівну y для $y < x$. Тому ця функція є монотонною, і нам потрібно знайти таке x, що ми можемо зробити максимальну суму підмасиву менше або рівну x, але ми не можемо зробити це для $x - 1$. Давайте фіксуємо деяке m та намагаємося зробити всі суми підмасивів менше або рівні m. Для цього ми перебираємо i від лівого до правого та знаходимо максимальну суму підмасиву для його елемента, і якщо він більший за m, то ми зменшуємо елемент i до такого значення, щоб цей відрізок мав суму менше або рівну m. Ми рахуємо кількість операцій, які ми повинні виконати, і якщо вона більша за k, то очевидно, що відповідь більша, інакше менша. Цей алгоритм працює з тим самим доведенням, як і в другому блоці. Рішення працює за $O(n^2 \cdot log(n))$.

Блок 5: $n \leqslant 10^5$:

Якщо ми використаємо рішення з блоку 4 та використаємо алгоритм для знаходження максимальної суми підмасиву з першого блоку, ми отримаємо рішення, яке працює за $O(n \cdot log^2(n))$.

Блок 6: без додаткових обмежень:

Тут нам потрібно оптимізувати рішення. Давайте оптимізувати знаходження максимального підмасиву. Як ми сказали, нам потрібно знайти мінімальний елемент на префіксі $[1, i]$ для всіх i. Це можна зробити за лінійний час, використовуючи той самий алгоритм, що й префіксна сума, але зберігати мінімум а не суму. Цей алгоритм працюватиме за $O(n)$, тому рішення працює за $O(n \cdot log(n))$.

Задача 2C. Трішки знань з криптографії нікому не завадить

Автор: **Фейса Богдан**
Підготували: **Фейса Богдан**
Розбір: **Фейса Богдан**

- Спостереження 1.

  Загальна кількість пірамідальних перестановок довжини n дорівнює 2^{n-1}.

  Доказ:

  Припустимо, що ви починаєте з послідовності з n нулів. На кожному кроці i ви будете додавати число i в якусь позицію. Завжди існує лише 2 можливих позиції для числа i, коли всі менші числа вже розташовані десь у послідовності.

  Доказ:

  Якщо після розміщення i в послідовності обидва сусідні елементи все ще нулі, позиція i стає локальним мінімумом. (І за визначенням це вже не пірамідальна перестановка, оскільки є щонайменше 2 локальні максимуми. (перефразувавши визначення, має бути не більше одного максимуму.))

  Лише останній елемент n має лише одне можливе розміщення.

  Таким чином, є $\underbrace{2 \cdot 2 \cdot \cdots \cdot 2 \cdot 2}_{\text{n-1 разів}} \cdot 1 = 2^{n-1}$ можливих перестановок.

- Спостереження 2:

  Кожна з 2^{n-1} перестановок може бути представлена за допомогою бітової маски довжиною $n-1$.

  Доказ:

  Зі спостереження 1 ми знаємо, що для кожного елемента, що менший за n, існує рівно 2 можливі позиції. Після створення перестановки ми робимо наступне:

  Встановлюємо всі біти в бітовій масці на 1.

  – для кожного числа j, яке розміщене в позиції, меншій за розміщення числа n, ми встановлюємо j-ий біт спереду рівним 0.

  Як приклад:

  $[1, 3, 5, 4, 2] \to [0, 1, 0, 1]$;

  $[1, 5, 4, 3, 2] \to [1, 0, 0, 0]$;

- Спостереження 3:

  для кожної перестановки її бітова маска також дорівнює кількості пірамідальних перестановок, що строго менші за неї.

Блок 1: $n \leqslant 10$:

У цьому блоці ми можемо перевірити кожну перестановку і визначити, чи є вона пірамідальною. Після цього ми можемо вручну знайти кількість тих, що менші або дорівнюють p. Часова складність $O(n! \cdot n)$

Блок 2: $n \leqslant 30$:

Ми можемо використовувати бінарний пошук, щоб знайти лексикографічно найбільшу перестановку, меншу або рівну даній. Ми будемо використовувати спостереження $1, 2, 3$ для цього.

Блок 3: $n \leqslant 60$:

Ми будемо використовувати ідею з блоку 2, але з трохи більшими обмеженнями та виводом за модулем.

Блок 4: $n \leqslant 500$:

Ми вручну побудуємо перестановку, що менша або дорівнює p, після цього перетворимо її в бінарний рядок і, нарешті, у бажану відповідь. Перестановка будується наступним чином:

Для деякого префікса довжини k найбільша пірамідальна перестановка лексикографічно менша або рівна p буде дорівнювати префіксу p. Ми будемо намагатися побудувати перестановку для кожного k ($0 \leqslant k \leqslant n$). Якщо перші k елементів однакові, $k+1$-й елемент повинен бути меншим за p_{k+1}. Ми можемо зберігати числа, які не були використані, у множині. Після цього, починаючи з $k+2$-го елемента, ми візьмемо найбільше, що не було використано, і використаємо його. Якщо створена перестановка не більша за p, ми збережемо її на потім. Існує кілька умов, коли побудова перестановки з першими k елементами, рівними використаним у p, неможлива:

- є щонайменше один індекс $j \leqslant k$, що $p_j < p_{j-1}$.
- $k+1 < n$ і коли ми на позиції $k+1$, немає числа, меншого за p_{k+1}.
- побудована перестановка для фіксованого k лексикографічно більша за p.

Після побудови кожної можливої перестановки ми виберемо лексикографічно найбільшу. Ми використовуємо спостереження 2 і перетворимо її на бітову маску. Після цього, використовуючи спостереження 3, ми обчислимо відповідь і виведемо її за бажаним модулем.

Загальна часова складність: $O(n^2 \cdot \log(n))$ або $O(n^2)$ залежно від реалізації.

Загальна використана пам'ять: $O(n^2)$ для зберігання оптимальних перестановок для кожного k.

Блок 5: $n \leqslant 10^4$:

Ми будемо використовувати ідею з попереднього блоку. Якщо ми візьмемо рішення $O(n^2)$ з попереднього блоку, нам лише потрібно оптимізувати пам'ять. Для оптимізації пам'яті ми не будемо зберігати кожну перестановку, лише найбільшу. Коли створюється нова можлива перестановка, ми перевіряємо, чи більша вона за поточний максимум, якщо так, ми видаляємо попередню і зберігаємо нову.

Блок 6: $n \leqslant 10^5$:

Остаточне спостереження для оптимізації полягає в тому, що можлива перестановка, створена для фіксованого k, **завжди** буде більшою, ніж та, що створена для $k-1$ (якщо обидві перестановки можна створити).

Отже, завдання полягає в тому, щоб знайти максимальне можливе k, для якого можна створити пірамідальну перестановку.

Це можна досягти, відстежуючи перший індекс m p, де:

- якщо $i <$ позиція числа n і $p_i < p_{i-1}$.
- якщо $i \geqslant$ позиція числа n і $p_i > p_{i-1}$.

якщо немає індексу m, який задовольняє наведені умови, p є пірамідальним, і ми повинні безпосередньо перейти до спостережень 2 і 3, щоб отримати відповідь.

Якщо бажаний m знайдено, є два можливі випадки:

- $m <$ позиція числа n. Нам потрібно знайти найбільший $j \leqslant m$, що $p_j > p_{j-1} + 1$.
- $m \geqslant$ позиція числа n. Нам потрібно знайти найбільший $j \leqslant$ (позиція числа n в p), що $p_j > p_{j-1} + 1$.

індекс j, який ми знайшли, буде дорівнювати $k+1$, тому ми можемо створити рішення для фіксованого k за час $O(n \cdot \log(n))$. Базова реалізація полягає у зберіганні кожного використаного значення у префіксі в множині.

Після цього ми беремо побудовану перестановку і, використовуючи спостереження $2, 3$, знаходимо відповідь.

Часова складність $O(n \cdot \log(n))$.

Блок 7: без додаткових обмежень:

Останній блок вимагає подальшої оптимізації, позбавившись структури *set* та попередньо обчисливши ступені числа 2, необхідні для визначення відповіді.

Часова складність $O(n)$.

Задача 2D. Трішки формальних умов нікому не завадять

Автор: **Ціцей Павло**
Підготував: **Ціцей Павло**
Розбір: **Ціцей Павло**

Блок 1: $n \leqslant 100$:

Нехай елемент a_i максимальний на деякому підмасиві. Тоді ми можемо переглянути всі підмасиви, де $l \leqslant i \leqslant r$, і підрахувати кількість підмасивів, які відповідають умові, та a_i там є максимальним. Рішення працює за $O(n^3)$.

Блок 2: $n \leqslant 10^3$:

Фіксуємо лівий край підмасиву та ітеруємося вправо для правого краю. Також, фіксуємо індекс максимального елемента на відрізку паралельно. Використовуючи префіксну суму, ми можемо перевірити, чи відрізок є добрим за константний час, тому рішення працює за $O(n^2)$.

Блок 3: $a_1 < a_2 < \cdots < a_n$:

Розширимо рішення для першого блоку. Якщо a_i є максимальним, то i є правим краєм підмасиву, де a_i є максимальним, оскільки $a_{i+1} > a_i$. Тоді у нас вже є алгоритм $O(n^2)$. Нам потрібно знайти кількість (l, r), де $a_l + \cdots + a_k = a_k + \cdots + a_r$ є правдою. Ми знаємо, що $k = r$, тому $a_l + \cdots + a_r = a_r \implies a_l + \cdots + a_{r-1} = 0$. Таким чином, завданням це знайти кількість підмасивів з сумою всіх елементів, рівною 0, і $r < n$, що можна легко вирішити за $O(n)$, використовуючи хеш-таблицю. Також, нам потрібно додати n до відповіді, оскільки $a_l = a_r$ завжди є хорошим підмасивом. Таким чином, рішення працює за $O(n)$.

Блок 4: існує i $(1 \leqslant i \leqslant n)$, таке, що $a_1 < \cdots < a_i$ та $a_i > \cdots > a_n$:

Розширимо рішення для третього блоку. Таким чином, ми знаємо, як вирішити завдання для $a_1 < \cdots < a_i$ $(l, r \leqslant i)$ і за симетрією для $a_i > \cdots > a_n$ $(l, r \geqslant i)$. Тому ми можемо фіксувати a_i та шукати $l < i$ та $r > i$. Ми можемо ітеруватися від $i - 1$ до 1 та фіксувати l, запам'ятовуючи суму від l до i $(a_l + \cdots + a_i)$. Потім ітеруємося від $i + 1$ до n, щоб фіксувати r. Потім ми можемо легко знайти кількість l, таких що $a_i + \cdots + a_r = a_i + \cdots + a_l$. Використовуючи хеш-таблицю, рішення працює за $O(n)$.

Блок 5: $n \leqslant 10^5$:

Розширимо рішення для першого блоку. Ми можемо знайти перше більше число зліва та справа, використовуючи стек за $O(n)$. Для елемента i ми маємо l_i та r_i, де l_i - перший більший елемент зліва, а r_i - перший більший елемент справа. Якщо $i - l_i \leqslant r_i - i$, то ми можемо ітеруватися від i до l_i та фіксувати l (симетрично для r), і ми матимемо суму $a_l + \cdots + a_i$. Тому нам потрібно знайти кількість r таких, що $i \leqslant r < r_i$ та $a_i + \cdots + a_r = a_l + \cdots + a_i$. Нехай b буде префіксною сумою a. Тоді $a_i + \cdots + a_r = a_l + \cdots + a_i$ те саме, що й $b_r - b_{i-1} = b_i - b_{l-1}$. Нехай h буде хеш-таблицею масивів. Тоді ми можемо запам'ятати i в h_{b_i}. Використовуючи це, коли ми фіксуємо l, ми знаємо, чому дорівнює b_r, і використовуючи бінарний пошук знаходимо кількість таких індексів r в діапазоні $[i, r_i)$. Насправді, це рішення працює за $O(n \cdot log^2(n))$, оскільки бінарний пошук в хеш-таблиці працює за $log(n)$, а перебір працює принаймні за $n \cdot log(n)$.

Доведення:

Ми використовуватимемо математичну індукцію за кількістю елементів. Для $n = 1$ або $n = 0$ відповідь робоча за принаймні $n \cdot log(n)$. Розглянемо $n > 1$. Нехай k буде індексом максимального

елемента в масиві. За індукцією, відповідь у підмасиві $[1, k-1]$ є принаймні $(k-1) \cdot log(k-1)$, а для $[k+1, n]$ є принаймні $(n-k-1) \cdot log(n-k-1)$. Також, $min(k-1, n-k) \leqslant \frac{n}{2}$, тому рішення працює за $(k-1) \cdot log(k-1) + (n-k-1) \cdot log(n-k-1) + \frac{n}{2}$. Це сума однієї зростаючої та однієї спадної функції, тому максимальне значення досягається, коли $k = \frac{n}{2}$, і ми отримуємо $(k-1) \cdot log(k-1) + (n-k-1) \cdot log(n-k-1) + \frac{n}{2} \leqslant \frac{n}{2} \cdot log(\frac{n}{2}) + \frac{n}{2} \cdot log(\frac{n}{2}) + \frac{n}{2} = n \cdot (log(\frac{n}{2}) + \frac{1}{2}) = n \cdot (log(\frac{n}{2}) + log(\sqrt{2})) \leqslant n \cdot log(n)$.

Блок 6: масив складається з випадкових цілих чисел $(-10^9 \leqslant a_i \leqslant 10^9)$:

Тут ми можемо використати метод "розділяй і володарю щоб вирішити цей блок за $n \cdot log(n)$. Спочатку, нехай k буде індексом максимального елемента в масиві. Потім ми можемо ітерувати через праву сторону та запам'ятати в хеш-таблиці кількість кожної префіксної суми. Після цього ми можемо ітерувати через ліву сторону та використовуючи формулу $b_r = b_i + b_{i-1} - b_{l-1}$ можемо підсумувати всі відповіді для цього максимуму за $O(n)$. Потім нам лише потрібно знайти відповіді для підмасивів $[1, k]$ та $[k, n]$, що ми можемо зробити рекурсивно. Рішення працює за $O(n \cdot log(n))$ на випадкових тестах.

Доведення:

Коли тести є випадковими, ймовірність того, що максимум знаходиться на сегменті $[l, r]$ дорівнює $\frac{r-l+1}{n}$. Тому, якщо ми призначимо $l = \frac{n}{4}$ та $r = \frac{3 \cdot n}{4}$, то ймовірність дорівнює $\frac{1}{2}$. Таким чином, ймовірність того, що максимум знаходиться близько до центру, дорівнює $\frac{1}{2}$, що означає, що в середньому 1 з 2 разів розмір сегмента зменшується в 2 рази, тому рішення зменшується експоненційно, і, отже, має логарифмічні стани, тому в середньому працює за $O(n \cdot log(n))$.

Блок 7: немає додаткових обмежень.:

Тут є 2 рішення, обидва працюють за $O(n \cdot log(n))$:

Перший:

Давайте використовувати «розділяй і володарюй». Маємо відрізок від l до r. Нехай $m = \lfloor \frac{l+r}{2} \rfloor$ і ми підраховуємо кількість підвідрізків, які задовольняють твердження та перетинають точку m. Припустимо, що максимум знаходиться десь праворуч від m. Перебираючи праву частину підвідрізку, ми можемо однозначно визначити, яке значення має бути максимальним. Якщо ми зараз знаходимося в індексі r, тоді ми знаємо, що максимум знаходиться десь на підвідрізку $[m, r]$, тому це може бути максимальне значення лише там. Нехай k — індекс максимального елемента. Тепер нам потрібно знайти таке число l, що $b_{l-1} = b_k - b_r + b_{k-1}$. Крім того, максимум на відрізку $[l, i]$ повинен бути меншим за a_k. Це можна зробити за допомогою двох вказівників. Ми можемо запам'ятати всі індекси на відрізку $[l, i]$ у хеш-таблиці, і хоча максимум на відрізку $[l, i]$ менше ніж a_k, ми можемо запам'ятати b_l, а потім зменшити його. Крім того, нам потрібно отримати відповіді з рекурсивних викликів ліворуч і праворуч.

Другий:

Ми можемо розширити рішення для 6 блоку. Воно працює за $O(n \cdot log^2(n))$ через бінарний пошук, тому ми позбудемося цього. Отже, нам потрібно знайти число r у діапазоні $[i, r_i)$ із фіксованим значенням. Іншими словами, нам дано щонайбільше $n \cdot log(n)$ запитів типу знайти кількість чисел x на відрізку $[l, r]$. Ми можемо використовувати «розділяй і володарюй» по запитах. Отже, для фіксованих l_i, r_i ми маємо $m = \lfloor \frac{l_i+r_i}{2} \rfloor$, де кожен запит проходить через m. Ми можемо розділити запит $x, [l, r]$ на запити $x, [l, m]$ і $x, [m+1, r]$. Ми виправили m, щоб ми могли просто повторювати та запам'ятовувати відповіді хеш-таблиці паралельно, щоб отримати рішення в $O(n \cdot log(n))$.