

Поиск

Горденко Мария Константиновна

Постановка задачи поиска

Пусть задано множество данных, которое описывается как *массив*, состоящий из N элементов.

Требуется найти:

- Элемент, равный заданному ключу/индекс элемента, количество элементов, равных заданному ключу и т.п.
- Максимальный элемент/индекс максимального элемента/количество максимальных элементов и т.п.
- Количество элементов, значения которых лежат в заданном интервале.

Уточнения: учесть ситуацию, когда заданного ключа нет в массиве

Линейный поиск (linear search)

- Данный алгоритм является простейшим алгоритмом поиска
- Не накладывает никаких ограничений на функцию
- Имеет простейшую реализацию
- Поиск осуществляется простым сравнением очередного рассматриваемого значения множества данных с заданным значением, и если значения совпадают (с той или иной точностью), то поиск считается завершённым.

Линейный поиск (linear search)

- Асимптотическая сложность - $O(n)$
- Худший случай – искомого элемента не существует
- Лучший случай – первый элемент является искомым

```
1. def linear_search(A, L, R, Key):  
2.     for X in range(L, R + 1):  
3.         if A[X] == Key:  
4.             return X  
5.     return -1
```

Бинарный поиск (binary search)

- Бинарный поиск производится в упорядоченном массиве
- Алгоритм может быть определен в рекурсивной и нерекурсивной формах
- При бинарном поиске искомый ключ сравнивается с ключом среднего элемента в массиве. Если они равны, то поиск успешен. В противном случае поиск осуществляется аналогично в левой или правой частях массива. В зависимости от постановки задачи, мы можем остановить процесс, когда мы получим первый или же последний индекс вхождения элемента. Последнее условие — это левосторонний/правосторонний двоичный поиск.

Бинарный поиск (binary search)

Для простоты дальнейших определений будем считать, что $a[-1] = -\infty$ и что $a[n] = +\infty$ (массив нумеруется с 0).

Правосторонний бинарный поиск (англ. *rightside binary search*) — бинарный поиск, с помощью которого мы ищем $\max_{i \in [-1, n-1]} \{i \mid a[i] \leq x\}$,

где a — массив, а x — искомый ключ

Левосторонний бинарный поиск (англ. *leftside binary search*) — бинарный поиск, с помощью которого мы ищем $\min_{i \in [0, n]} \{i \mid a[i] \geq x\}$, где a

— массив, а x — искомый ключ

Используя эти два вида двоичного поиска, мы можем найти отрезок позиций $[l, r]$ таких, что $\forall i \in [l, r] : a[i] = x$ и $\forall i \notin [l, r] : a[i] \neq x$

Бинарный поиск (binary search)

Задан отсортированный массив $[1, 2, 2, 2, 2, 3, 5, 8, 9, 11]$, $x = 2$

Правосторонний поиск двойки выдаст в результате 4, в то время как левосторонний выдаст 1 (нумерация с нуля).

Отсюда следует, что количество подряд идущих двоек равно длине отрезка $[1; 4]$, то есть 4.

Если искомого элемента в массиве нет, то правосторонний поиск выдаст максимальный элемент, меньший искомого, а левосторонний наоборот, минимальный элемент, больший искомого.

Напоминание

В некоторых языках программирования присвоение $m = (l + r) / 2$ приводит к переполнению. Вместо этого рекомендуется использовать $m = l + (r - l) / 2;$ или эквивалентные выражения.

Бинарный поиск (binary search)

- Асимптотическая сложность - $O(\log n)$
- Худший случай – искомый элемент первый/последний
- Лучший случай – средний элемент является искомым

```
def bin_search(a, key):  
    l = -1  
    r = len(a)  
    while l < r - 1:  
        m = (l + r) // 2  
        if a[m] < key:  
            l = m  
        else:  
            r = m  
    return r  
# l, r – левая и правая границы  
# Правая граница - длина массива  
# Пока левая граница строго меньше правой  
# m – середина области поиска. Используем целочисленное деление  
# Сдвигаем левую границу  
# Сузим область поиска, сдвигая правую границу  
# Возвращаем индекс элемента, равного ключу, или индекс элемента, который  
# следует за последним меньшим ключом
```

Бинарный поиск. Пример

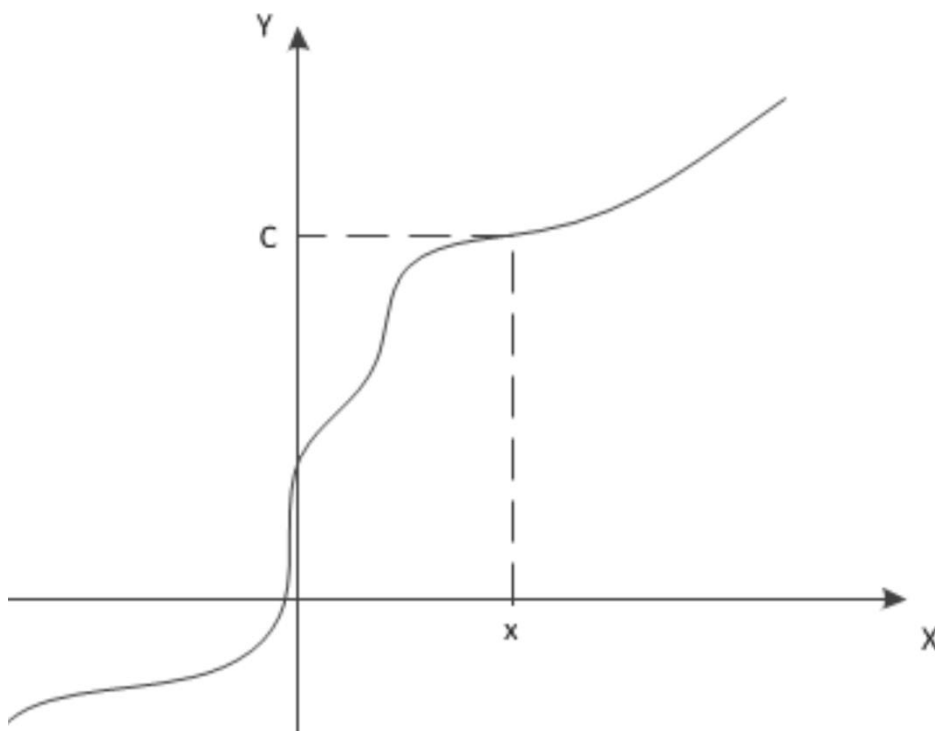
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Вещественный бинарный поиск (Bisection method)

- Вещественный двоичный поиск (англ. Bisection method)— алгоритм поиска аргумента для заданного значения монотонной вещественной функции.

Постановка задачи поиска

Пусть нам задана монотонная функция f и какое-то значение C этой функции. Необходимо найти значение аргумента x этой функции, такое, что $f(x) = C$.



Вещественный бинарный поиск (Bisection method)

- Примерное количество итераций алгоритма $O\left(\log \frac{right-left}{eps}\right)$

```
def find_left_board(C):  
    x = -1  
    while f(x) > C:  
        x = x * 2  
    return x
```

```
1. def find_right_board(C):  
2.     x = 1  
3.     while f(x) < C:  
4.         x = x * 2  
5.     return x
```

```
1. def bin_search(C, eps=1e-6):  
2.     left = find_left_board(C)  
3.     right = find_right_board(C)  
4.     while right - left > eps:  
5.         mid = (left + right) / 2  
6.         if f(mid) < C:  
7.             left = mid  
8.         else:  
9.             right = mid  
10.    return (left + right) / 2
```

Замечания

- Необходимо отметить, то функция должна быть строго монотонна, если мы ищем конкретный корень и он единственный. Нестрого монотонна, если нам необходимо найти самый левый (правый) аргумент. Если же функция не монотонна, то данный алгоритм не найдет искомый аргумент, либо найдет аргумент, но он не будет единственным.

Троичный поиск (ternary search)

- Троичный поиск (ternary search, тернарный поиск) — метод поиска минимума или максимума функции на отрезке (функции), которая либо сначала строго возрастает, затем строго убывает, либо наоборот.

Постановка задачи поиска

- Пусть дана функция $f(x)$, унимодальная на некотором отрезке $[l; r]$.
- Под унимодальностью понимается один из двух вариантов. Первый: функция сначала строго возрастает, потом достигает максимума (в одной точке или целом отрезке), потом строго убывает. Второй вариант, симметричный: функция сначала убывает, достигает минимума, возрастает.
- Требуется найти максимум функции $f(x)$ на отрезке $[l; r]$.

Алгоритм

Пусть функция $f(x)$ на отрезке $[l, r]$ имеет минимум, и мы хотим найти точку x_{min} , в которой он достигается.

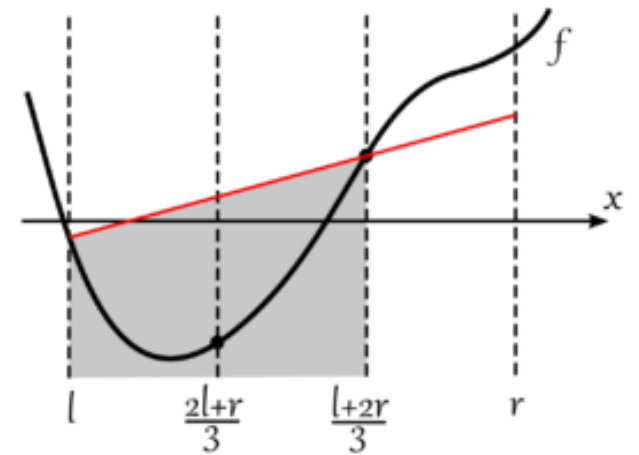
Посчитаем значения функции в точках $a = l + \frac{(r-l)}{3}$ и $b = l + \frac{2(r-l)}{3}$.

Так как в точке x_{min} минимум, то на отрезке $[l, x_{min}]$ функция убывает, а на $[x_{min}, r]$ — возрастает, то есть

$\forall x', x'' \in [l, r]$:

$$l < x' < x'' < x_{min} \implies f(l) > f(x') > f(x'') > f(x_{min})$$

$$x_{min} < x' < x'' < r \implies f(x_{min}) < f(x') < f(x'') < f(r)$$



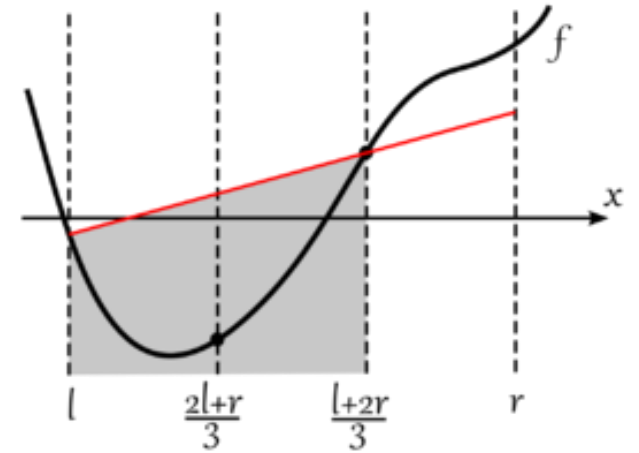
Алгоритм

Значит если $f(a) < f(b)$, то $x_{min} \in [l, b]$, аналогично из $f(a) > f(b)$ следует $x_{min} \in [a, r]$.

Тогда нам нужно изменить границы поиска и искать дальше, пока не будет достигнута необходимая точность, то есть $r - l < \varepsilon$.

Из рассуждений и рисунка может возникнуть идея взять, например, отрезок $[l, a]$ вместо отрезка $[l, b]$. Но этого делать нельзя, потому что мы не умеем различать случаи, когда $f(a) < f(b)$ и a слева или справа от минимума.

Можно заметить, что если мы всегда будем брать отрезок $[l, b]$ при $f(a) < f(b)$ или $[a, r]$ при $f(a) > f(b)$, то минимум функции всегда будет в нашем отрезке. Если $f(a) = f(b)$, то можно взять любой отрезок.



Алгоритм

```
def ternary_search_min_rec(f, left, right, eps):  
    if right - left < eps:  
        return (left + right) / 2  
    a = (left * 2 + right) / 3  
    b = (left + right * 2) / 3  
    if f(a) < f(b):  
        return ternary_search_min_rec(f, left, b, eps)  
    else:  
        return ternary_search_min_rec(f, a, right, eps)
```

```
def ternary_search_min_iter(f, left, right, eps):  
    while right - left > eps:  
        a = (left * 2 + right) / 3  
        b = (left + right * 2) / 3  
        if f(a) < f(b):  
            right = b  
        else:  
            left = a  
    return (left + right) / 2
```

Время работы

Так как на каждой итерации мы считаем два значения функции и уменьшаем область поиска в полтора раза, пока $r - l > \varepsilon$, то время работы алгоритма составит $2 \log_{\frac{3}{2}} \left(\frac{r - l}{\varepsilon} \right)$

Интерполяционный поиск (interpolating search)

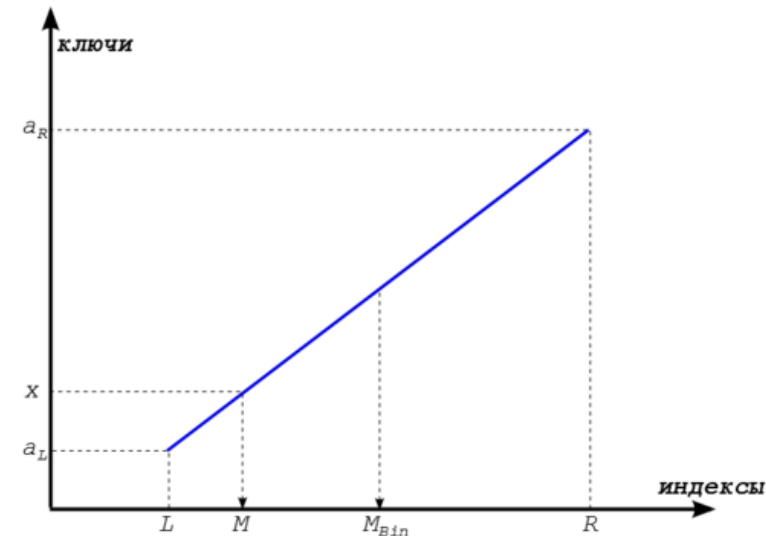
- В основе интерполяционного поиска лежит операция интерполирование.
- Интерполирование – нахождение промежуточных значений величины по имеющемуся дискретному набору известных значений. Интерполяционный поиск работает только с упорядоченными массивами; он похож на бинарный, в том смысле, что на каждом шаге вычисляется некоторая область поиска, которая, по мере выполнения алгоритма, сужается.
- В отличие от двоичного, интерполяционный поиск не делит последовательность на две равные части, а вычисляет приблизительное расположение ключа (искомого элемента), ориентируясь на расстояние между искомым и текущим значением элемента.

Алгоритм

Пусть a — отсортированный массив из n чисел, x — значение, которое нужно найти. Поиск происходит подобно двоичному поиску, но вместо деления области поиска на две примерно равные части, интерполирующий поиск производит оценку новой области поиска по расстоянию между ключом и текущим значением элемента. Если известно, что x лежит между a_l и a_r , то следующая проверка выполняется примерно на расстоянии $\frac{x-a_l}{a_r-a_l} \cdot (r-l)$ от l .

Формула для разделительного элемента m получается из следующего уравнения: $\frac{x-a_l}{m-l} = \frac{a_r-a_l}{r-l}$ — откуда следует, что $m = l + \frac{x-a_l}{a_r-a_l} \cdot (r-l)$. На рисунке внизу показано, из каких соображений берется такая оценка.

Интерполяционный поиск основывается на том, что наш массив представляет из себя что-то наподобии арифметической прогрессии.



Алгоритм

```
1. int interpolationSearch(a : int[], key : int)
2.   left = 0
3.   right = a.length - 1

4.   while a[left] < key and key < a[right]
5.     mid = left + (key - a[left]) * (right -
left) / (a[right] - a[left])
6.     if a[mid] < key
7.       left = mid + 1
8.     else if a[mid] > key
9.       right = mid - 1
10.    else
11.      return mid

12.  if a[left] == key
13.    return left
14.  else if a[right] == key
15.    return right
16.  else
17.    return -1
```

Сложность?

Binary vs interpolating

Поиск по ключу 5 в массиве:

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Бинарный поиск

1) left = 0, right = 19, mid = 10

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

2) left = 11, right = 19, mid = 15

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

3) left = 11, right = 14, mid = 12

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

4) left = 13, right = 14

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Интерполирующий поиск

1) left = 0, right = 19, mid = $0 + (5-1) * (8-1) * 19 = 10$

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

2) left = 11, right = 19, mid = $11 + (5-4) * (8-4) * 8 = 13$

1	1	1	2	2	3	3	3	4	4	4	4	5	5	6	6	7	7	8	8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

А. Поиск

	Все языки	Python 3.7.3
Ограничение времени	5 секунд	5 секунд
Ограничение памяти	64.0 Мб	256.0 Мб
Ввод	стандартный ввод или input.txt	
Вывод	стандартный вывод или output.txt	

Реализуйте алгоритм бинарного поиска.



Формат ввода

В первой строке входных данных содержатся натуральные числа N и K ($1 \leq N, K \leq 1000000$). Во второй строке задаются N элементов первого массива, отсортированного по возрастанию, а в третьей строке – K элементов второго массива. Элементы обоих массивов - целые числа, каждое из которых по модулю не превосходит 10^9

Формат вывода

Требуется для каждого из K чисел вывести в отдельную строку "YES", если это число встречается в первом массиве, и "NO" в противном случае.

Пример 1

Ввод 	Вывод 
3 3	NO
2 3 5	YES
1 2 3	YES

В. Быстрый поиск в массиве

Ограничение времени	3 секунды
Ограничение памяти	64Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Дан массив из N целых чисел. Все числа от -10^9 до 10^9 .
Нужно уметь отвечать на запросы вида “Сколько чисел имеют значения от L до R ?”.

Формат ввода

Число N ($1 \leq N \leq 10^5$). Далее N целых чисел.
Затем число запросов K ($1 \leq K \leq 10^5$).

Далее K пар чисел L, R ($-10^9 \leq L \leq R \leq 10^9$) — собственно запросы.

Формат вывода

Выведите K чисел — ответы на запросы.

Пример

Ввод	Вывод
5	5 2 2 0
10 1 10 3 4	
4	
1 10	
2 9	
3 4	
2 2	

С. Корень кубического уравнения

Ограничение времени	1 секунда
Ограничение памяти	64Mb
Ввод	cubroot.in
Вывод	cubroot.out

Дано кубическое уравнение $ax^3+bx^2+cx+d=0$ ($a\neq 0$). Известно, что у этого уравнения есть ровно один корень. Требуется его найти.

Формат ввода

Во входном файле через пробел записаны четыре целых числа:
 $-1000 \leq a, b, c, d \leq 1000$.

Формат вывода

Выведите единственный корень уравнения с точностью не менее 5 знаков после десятичной точки.

Пример 1

Ввод	Вывод
------	-------

1 -3 3 -1	1.0000036491
-----------	--------------

D. N в степени N

Ограничение времени	1 секунда
Ограничение памяти	731Mb
Ввод	стандартный ввод или input.txt
Вывод	стандартный вывод или output.txt

Вам даны два целых числа N и M ($1 \leq N \leq 10^9$, $2 \leq M \leq 10^9$). Вычислите остаток от деления N^N на M .

Формат ввода

Входной файл содержит через пробел числа N и M .



Формат вывода

Выведите одно число — ответ к задаче.

Пример 1

Ввод 	Вывод 
3 1001	27

Пример 2

Ввод 	Вывод 
10 123456	64000

Е. Дипломы

Ограничение времени	2 секунды
Ограничение памяти	256Mb
Ввод	стандартный ввод или diploma.in
Вывод	стандартный вывод или diploma.out

Когда Петя учился в школе, он часто участвовал в олимпиадах по информатике, математике и физике. Так как он был достаточно способным мальчиком и усердно учился, то на многих из этих олимпиад он получал дипломы. К окончанию школы у него скопилось n дипломов, причем, как оказалось, все они имели одинаковые размеры: w – в ширину и h – в высоту.

Сейчас Петя учится в одном из лучших российских университетов и живет в общежитии со своими одногруппниками. Он решил украсить свою комнату, повесив на одну из стен свои дипломы за школьные олимпиады. Так как к бетонной стене прикрепить дипломы достаточно трудно, то он решил купить специальную доску из пробкового дерева, чтобы прикрепить ее к стене, а к ней – дипломы. Для того чтобы эта конструкция выглядела более красиво, Петя хочет, чтобы доска была квадратной и занимала как можно меньше места на стене. Каждый диплом должен быть размещен строго в прямоугольнике размером w на h . Прямоугольники, соответствующие различным дипломам, не должны иметь общих внутренних точек.

Требуется написать программу, которая вычислит минимальный размер стороны доски, которая потребуется Пете для размещения всех своих дипломов.

Формат ввода

Входной файл содержит три целых числа: w, h, n ($1 \leq w, h, n \leq 10^9$).

Формат вывода

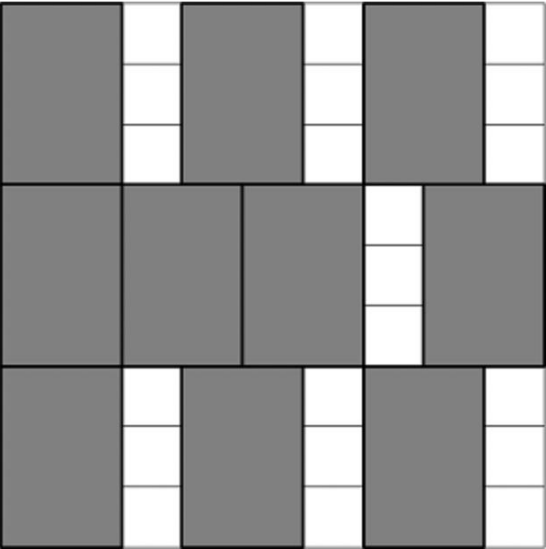
В выходной файл необходимо вывести ответ на поставленную задачу.

Пример

Ввод	Вывод
2 3 10	9

Примечания

Иллюстрация к примеру



F. Коровы -- в стойла!

Ограничение времени	1 секунда
Ограничение памяти	256Mb
Ввод	стандартный ввод или cows.in
Вывод	стандартный вывод или cows.out

На прямой расположены стойла, в которые необходимо расставить коров так, чтобы минимальное расстояние между коровами было как можно больше.

Формат ввода

В первой строке вводятся числа N ($2 < N \leq 10^4$) — количество стойл и K ($1 < K < N$) — количество коров. Во второй строке задаются N натуральных чисел в порядке возрастания — координаты стойл (координаты не превосходят 10^9).

Формат вывода

Выведите одно число — наибольшее возможное допустимое расстояние.

Пример

Ввод	Вывод
<div> <div>5</div> <div>3</div> <div>1</div> <div>2</div> <div>3</div> <div>100</div> <div>1000</div> </div>	<div>99</div>

С утра шел дождь, и ничего не предвещало беды. Но к обеду выглянуло солнце, и в лагерь заглянула СЭС. Пройдя по всем домикам и корпусам, СЭС вынесла следующий вердикт: бельевые веревки в жилых домиках не удовлетворяют нормам СЭС. Как выяснилось, в каждом домике должно быть ровно по одной бельевой веревке, и все веревки должны иметь одинаковую длину. В лагере имеется N бельевых веревок и K домиков. Чтобы лагерь не закрыли, требуется так нарезать данные веревки, чтобы среди получившихся веревочек было K одинаковой длины. Размер штрафа обратно пропорционален длине бельевых веревок, которые будут развешены в домиках. Поэтому начальство лагеря стремиться максимизировать длину этих веревочек.



Формат ввода

В первой строке заданы два числа — N ($1 \leq N \leq 10001$) и K ($1 \leq K \leq 10001$). Далее в каждой из последующих N строк записано по одному числу — длине очередной бельевой веревки. Длина веревки задана в сантиметрах. Все длины лежат в интервале от 1 сантиметра до 100 километров включительно.

Формат вывода

В выходной файл следует вывести одно целое число — максимальную длину веревочек, удовлетворяющую условию, в сантиметрах. В случае, если лагерь закроют, выведите 0 .

Пример

Ввод 	Вывод 
4 11 802 743 457 539	200

1539. Kth Missing Positive Number

Easy

Topics

Companies

Hint

Given an array `arr` of positive integers sorted in a **strictly increasing order**, and an integer `k`.

Return the k^{th} **positive** integer that is **missing** from this array.

Example 1:

Input: `arr = [2,3,4,7,11]`, `k = 5`

Output: 9

Explanation: The missing positive integers are `[1,5,6,8,9,10,12,13,...]`. The 5th missing positive integer is 9.

Example 2:

Input: `arr = [1,2,3,4]`, `k = 2`

Output: 6

Explanation: The missing positive integers are `[5,6,7,...]`. The 2nd missing positive integer is 6.

Constraints:

- `1 <= arr.length <= 1000`
- `1 <= arr[i] <= 1000`
- `1 <= k <= 1000`
- `arr[i] < arr[j]` for `1 <= i < j <= arr.length`

540. Single Element in a Sorted Array

Medium

🔖 Topics

🏢 Companies

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return *the single element that appears only once*.

Your solution must run in $O(\log n)$ time and $O(1)$ space.

Example 1:

Input: `nums = [1,1,2,3,3,4,4,8,8]`

Output: `2`

Example 2:

Input: `nums = [3,3,7,7,10,11,11]`

Output: `10`

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $0 \leq \text{nums}[i] \leq 10^5$

33. Search in Rotated Sorted Array

Medium

Topics

Companies

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` ($1 \leq k < \text{nums.length}$) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if it is not in* `nums`.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: `nums = [4,5,6,7,0,1,2]`, `target = 0`

Output: `4`

Example 2:

Input: `nums = [4,5,6,7,0,1,2]`, `target = 3`

Output: `-1`

Example 3:

Input: `nums = [1]`, `target = 0`

Output: `-1`

153. Find Minimum in Rotated Sorted Array

Medium

Topics

Companies

Hint

Suppose an array of length n sorted in ascending order is **rotated** between 1 and n times. For example, the array `nums = [0,1,2,4,5,6,7]` might become:

- `[4,5,6,7,0,1,2]` if it was rotated 4 times.
- `[0,1,2,4,5,6,7]` if it was rotated 7 times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in $O(\log n)$ time.

Example 1:

Input: `nums = [3,4,5,1,2]`

Output: `1`

Explanation: The original array was `[1,2,3,4,5]` rotated 3 times.

Example 2:

Input: `nums = [4,5,6,7,0,1,2]`

Output: `0`

Explanation: The original array was `[0,1,2,4,5,6,7]` and it was rotated 4 times.

Example 3:

Input: `nums = [11,13,15,17]`

Output: `11`

Explanation: The original array was `[11,13,15,17]` and it was rotated 4 times.

154. Find Minimum in Rotated Sorted Array II

Hard

Topics

Companies

Suppose an array of length n sorted in ascending order is **rotated** between 1 and n times. For example, the array `nums = [0,1,4,4,5,6,7]` might become:

- `[4,5,6,7,0,1,4]` if it was rotated 4 times.
- `[0,1,4,4,5,6,7]` if it was rotated 7 times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` that may contain **duplicates**, return *the minimum element of this array*.

You must decrease the overall operation steps as much as possible.

Example 1:

Input: `nums = [1,3,5]`

Output: `1`

Example 2:

Input: `nums = [2,2,2,0,1]`

Output: `0`

162. Find Peak Element

Medium

Topics

Companies

A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in $O(\log n)$ time.

Example 1:

Input: `nums = [1,2,3,1]`

Output: 2

Explanation: 3 is a peak element and your function should return the index number 2.

Example 2:

Input: `nums = [1,2,1,3,5,6,4]`

Output: 5

Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

367. Valid Perfect Square

Easy

📁 Topics

🏢 Companies

Given a positive integer `num`, return `true` if `num` is a perfect square or `false` otherwise.

A **perfect square** is an integer that is the square of an integer. In other words, it is the product of some integer with itself.

You must not use any built-in library function, such as `sqrt`.

Example 1:

Input: `num = 16`

Output: `true`

Explanation: We return `true` because $4 * 4 = 16$ and 4 is an integer.

Example 2:

Input: `num = 14`

Output: `false`

Explanation: We return `false` because $3.742 * 3.742 = 14$ and 3.742 is not an integer.

Constraints:

- $1 \leq \text{num} \leq 2^{31} - 1$