Абстрактные типы данных. Стек, очередь, дек, куча.

Стек (stack)

A. *Стек с защитой от ошибок*

Реализуйте структуру данных "стек". Напишите программу, содержащую описание стека и моделирующую работу стека, реализовав все указанные здесь методы. Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку.

Описание команд:

`push n` Добавить в стек число n. Программа должна вывести `ok`.

`pop` Удалить из стека последний элемент. Программа должна вывести его значение. Если стек пуст, то программа должна вместо числового значения вывести строку `error`.

`back` Программа должна вывести значение последнего элемента, не удаляя его из стека. Если стек пуст, то программа должна вместо числового значения вывести строку `error`.

`size` Программа должна вывести количество элементов в стеке.

`clear` Программа должна очистить стек и вывести `ok`.

`exit` Программа должна вывести `bye` и завершить работу.

Перед исполнением операций `back` и `pop` программа должна проверять, содержится ли в стеке хотя бы один элемент. Время работы каждой операции `O(1)`.

Input	Output
size push 1 size push 2 size push 3 size exit	0 ok 1 ok 2 ok 3 bye

B. *Ближайший бóльший справа*

Дана последовательность $A_0, A_1, \ldots, A_{n-1}$ из n целых чисел. Для каждого числа A_i найдите наименьшее j такое, что $j > i$ и $A_j > A_i$.

В первой строке входного файла записано натуральное число n ($n < 5 \cdot 10^4$). В следующей строке через пробел записаны целые числа $A_0, A_1, \ldots, A_{n-1}$.

Требуется вывести n чисел в одной строке через пробел. Для каждого i ($0 \leqslant i \leqslant n-1$) вывести соответствующее j. Если правее A_i нет чисел, бóльших его, в качестве ответа выведите число -1.

Время $O(n)$, дополнительная память $O(n)$.

Input	Output
5 1 3 2 4 5	1 3 3 4 -1

Определение: Правильной скобочной последовательностью называется последовательность символов `( ) [ ] { }`, которая может быть получена конечным числом применения следующих правил:

- Пустая последовательность — правильная.
- Если A, B — правильные последовательности, то их конкатенация A B — правильная последовательность.
- Если A — правильная последовательность, то $(A), [A], \{A\}$ — правильные последовательности.

C. *Правильная скобочная последовательность*

С клавиатуры вводится строка — скобочное выражение, содержащее три вида скобок: `( ) [ ] { }`.

Программа дожна определить, является ли данная скобочная последовательность правильной.

В единственной строке входного файла записана скобочная последовательность, содержащая не более 100000 скобок.

Если данная скобочная последовательность правильная, то программа должна вывести строку `YES`, иначе строку `NO`.

Input	Output
`()[]`	`YES`
`][`	`NO`

D. *Удаление скобок*

Дана строка, составленная из круглых скобок. Определите, какое наименьшее количество символов необходимо удалить из этой строки, чтобы оставшиеся символы образовывали правильную скобочную последовательность.

Во входном файле записана строка из круглых скобок. Длина строки не превосходит 100000 символов.

Выведите единственное целое число — ответ на поставленную задачу.

Input	Output
())(()	2
))(((	5

E. *Постфиксная запись*

В постфиксной записи (или обратной польской записи) операция записывается после двух операндов.

Термин получил название благодаря польскому логику Яну Лукасевичу, придумавшему обычную, т.е. префиксную запись формул (операция предшествует записи операндов).

Обратная польская запись (RPN — Reverse Polish Notation) появилась примерно в середине 50-х — начале 60-х годов благодаря работам сразу нескольких учёных (Burks, Warren, Wright, позже — Bauer, Dijkstra).

Ниже приведены примеры выражений в обычной и постфиксной формах:

Обычная	Постфиксная
A + B	A B +
(B + C) * D	B C + D *
A + (B + C) * D	A B C + D * +

Такой способ записи позволяет избавиться от скобок и вычислять значение алгебраического выражения при помощи стека, не занимаясь его синтаксическим разбором.

Выражение обрабатывается слева направо. Числа кладутся в стек, а операции выполняются с верхними двумя элементами стека, вместо которых в стек помещается результат этой операции.

Нужно обратить внимание на некоммутативные операции (вычитание и деление).

В единственной строке записано корректное выражение в постфиксной записи, содержащее однозначные числа и операции $+, -, *$, разделённые пробелами.

Необходимо вывести значение записанного выражения.

Input	Output
8 9 + 1 7 - *	-102

F. *Шарики*

В одной компьютерной игре игрок выставляет в линию шарики разных цветов. Когда образуется непрерывная цепочка из трёх и более шариков одного цвета, она удаляется из линии. Все шарики при этом сдвигаются друг к другу, и ситуация может повториться.

Напишите программу, которая по данной ситуации определяет, сколько шариков будет сейчас "уничтожено". Непрерывных цепочек из трех и более одноцветных шаров в начальный момент может быть не более одной.

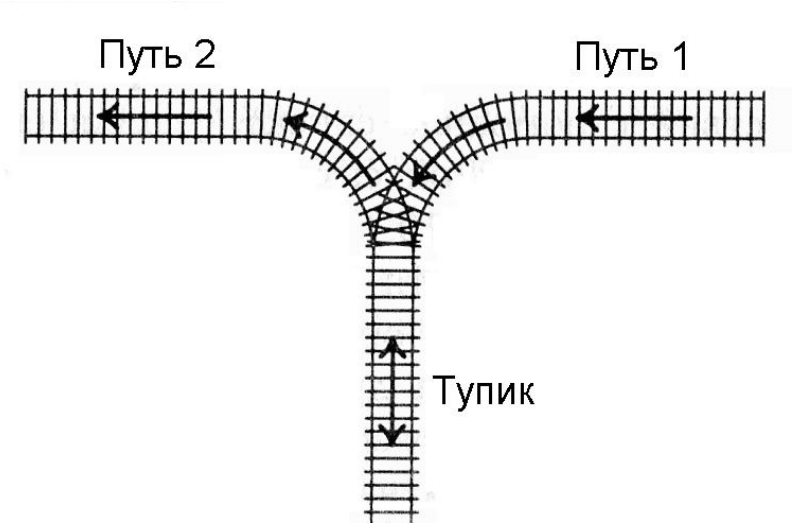
Сначала вводится число N ($1 \leqslant N \leqslant 3 \cdot 10^5$) — количество шариков в цепочке, а в следующей строке цвета шариков (числа от 0 до 9, каждому цвету соответствует своё целое число).

Требуется вывести количество шариков, которое будет "уничтожено".

Input	Output
6 5 1 3 3 3 2	3

G. *Сортировка вагонов*

К тупику со стороны пути 1 (см. рисунок) подъехал поезд. Разрешается отцепить от поезда один или сразу несколько первых вагонов и завезти их в тупик (при желании, можно даже завезти в тупик сразу весь поезд). После этого часть из этих вагонов вывезти в сторону пути 2. Затем можно завезти в тупик еще несколько вагонов и снова часть оказавшихся вагонов вывезти в сторону пути 2. И так далее (так, что каждый вагон может лишь один раз заехать с пути 1 в тупик, а затем один раз выехать из тупика на путь 2). Заезжать в тупик с пути 2 или выезжать из тупика на путь 1 запрещается. Нельзя с пути 1 попасть на путь 2, не заезжая в тупик.



Известно, в каком порядке изначально идут вагоны поезда. Требуется с помощью указанных операций сделать так, чтобы вагоны поезда шли по порядку (сначала первый, потом второй и т.д., считая от головы поезда, едущего по пути 2 в сторону от тупика). Напишите программу, определяющую, можно ли это сделать.

В первой строке вводится число N — количество вагонов в поезде ($1 \leqslant N \leqslant 10^5$). Во второй строке перечислены номера вагонов в порядке от головы поезда, едущего по пути 1 в сторону тупика. Вагоны пронумерованы натуральными числами от 1 до N, каждое из которых встречается ровно один раз.

Если можно сделать так, чтобы вагоны шли в порядке от 1 до N, считая от головы поезда, когда поезд поедет по пути 2 из тупика, выведите слово `YES`, иначе выведите `NO`.

Input	Output
3 3 2 1	YES
4 4 1 3 2	YES
3 2 3 1	NO

Комментарий к первому примеру: надо весь поезд завезти в тупик, а затем целиком вывезти его на 2-й путь.

Комментарий ко второму примеру: сначала надо в тупик завезти два вагона, один из которых оставить в тупике, а второй — вывезти на 2-й путь, после чего завезти в тупик еще два вагона и вывезти 3 вагона, стоящие в тупике, на 2-й путь.

H. *Баржа*

На барже располагается K грузовых отсеков. В каждый отсек можно поместить некоторое количество бочек с одним из 10000 видов топлива. Причём извлечь бочку из отсека можно лишь в случае, если все бочки, помещённые в этот отсек после неё, уже были извлечены. Таким образом в каждый момент времени в каждом непустом отсеке имеется ровно одна бочка, которую можно извлечь не трогая остальных. Будем называть такие бочки крайними.

В начале баржа пуста. Затем она последовательно проплывает через N доков, причём в каждом доке на баржу либо погружается бочка с некоторым видом топлива в некоторый отсек, либо выгружается крайняя бочка из некоторого отсека. Однако, если указанный отсек пуст, либо если выгруженная бочка содержит не тот вид топлива, который ожидалось, следует зафиксировать ошибку. Если на баржу оказывается погружено более P бочек или если после прохождения всех доков она не стала пуста, следует также зафиксировать ошибку. От вас требуется либо указать максимальное количество бочек, которые одновременно пребывали на барже либо зафиксировать ошибку.

В первой строке три целых числа N, K и P ($1 \leqslant N, K, P \leqslant 100000$). Далее следует N строк с описанием действия, выполняемого в очередном доке. Если в нём происходит погрузка, то строка имеет вид `+ A B`, где A — номер отсека, в который помещается бочка, а B — номер вида топлива в ней. Если же док занимается разгрузкой, то строка имеет вид `- A B`, где A — номер отсека, из которого извлекается бочка, а B — номер ожидаемого вида топлива.

Вывести либо одно число, равное искомому максимуму в случае безошибочного прохождения баржой маршрута, либо вывести слово `Error` в противном случае.

Input	Output
6 1 2 + 1 1 + 1 2 - 1 2 - 1 1 + 1 3 - 1 3	2

I* *Контейнеры*

На складе хранятся контейнеры с товарами N различных видов. Все контейнеры составлены в N стопок. В каждой стопке могут находиться товары любых видов. Стопка может быть первоначально пустой.

Автопогрузчик может взять верхний контейнер из любой стопки и поставить его сверху в любую стопку. Необходимо расставить все контейнеры с товаром первого вида в первую стопку, второго — во вторую и т.д.

В первой строке входных данных записано одно натуральное число N, не превосходящее 500.

В следующих N строках описаны стопки контейнеров: сначала записано число k_i — количество контейнеров в i-й стопке, а затем k_i чисел — виды товара в контейнерах в данной стопке, снизу вверх. В каждой стопке вначале не более 500 контейнеров (в процессе переноса контейнеров это ограничение может быть нарушено).

Программа должна вывести описание действий автопогрузчика: для каждого действия напечатать два числа – из какой стопки брать контейнер и в какую стопку класть. Обратите внимание, что минимизировать количество операций автопогрузчика не требуется. Если задача не имеет решения, необходимо вывести одно число 0. Если контейнеры изначально правильно размещены по стопкам, то выводить ничего не нужно.

Input	Output
3 4 1 2 3 2 0 0	1 2 1 3 1 2

Комментарий к примеру: в первой стопке лежат четыре контейнера — снизу контейнер с товаром первого вида, над ним — с товаром второго вида, над ним третьего, и сверху еще один контейнер с товаром второго вида. Вторая и третья стопки — пусты.

J. *Гемоглобин*

Каждый день к Грегори Хаусу приходит много больных, и у каждого измеряется уровень гемоглобина в крови. Данные по всем пациентам заносятся в базу данных. Но волчанка попадается один раз на миллион, а работать с остальными неинтересно. Чтобы Хаус не выгонял больных, Кадди иногда запрашивает статистику по k последним больным: ей хочется знать сумму их уровня гемоглобина. Также Хаус — мизантроп: он смотрит уровень гемоглобина больного, который поступил к нему позже всех, и, видя, что это точно не волчанка, выписывает его из больницы и удаляет информацию о нем из базы.

Автоматизацию процесса Хаус поручил Чейзу. Но Чейз почему-то не справился с этой задачей и попросил вас ему помочь.

В первой строке входного файла задано число n ($1 \leqslant n \leqslant 100000$) — число обращений к базе данных. Запросы к базе выглядят следующим образом: `+x` ($1 \leqslant x \leqslant 10^9$) — добавить пациента с уровнем гемоглобина x в базу, знак "`-`" — удалить последнего пациента из базы, `?k` ($1 \leqslant k \leqslant 10^5$) — вывести суммарный гемоглобин последних k пациентов. Гарантируется, что k не превосходит число элементов в базе. Также гарантируется, что запросов на удаление к пустой базе не поступает. Перед началом работы база данных пуста.

Для каждого запроса "`-`" вывести уровень гемоглобина в крови пациента, а для каждого запроса `?k` — суммарный гемоглобин у последних k оставшихся пациентов. Ответы выводите в порядке поступления запросов.

Input	Output
7 +1 +2 +3 ?2 - - ?1	5 3 2 1

K. *Гистограмма*

Гистограмма — многоугольник, сформированный из последовательности прямоугольников, выровненных на общей базовой линии. Прямоугольники имеют равную ширину, но могут иметь различные высоты. Например, фигура слева показывает гистограмму, которая состоит из прямоугольников с высотами $2, 1, 4, 5, 1, 3, 3$. Все прямоугольники на этом рисунке имеют ширину, равную 1.

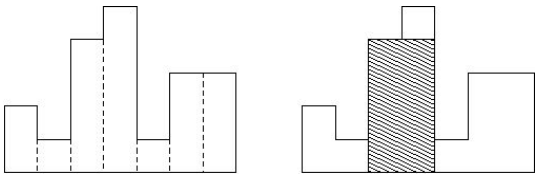


Рис. 1: Иллюстрация к тесту из условия

Вычислите максимальную площадь прямоугольника в гистограмме, который находится на общей базовой линии. На рисунке справа заштрихованная фигура является самым большим выровненным прямоугольником на изображенной гистограмме.

В первой строке входного файла записано число N ($0 \leqslant N \leqslant 10^6$) — количество прямоугольников гистограммы. Затем следует N целых чисел $h1, \ldots, h_n$, где $0 \leqslant h_i \leqslant 10^9$. Эти числа обозначают высоты прямоугольников гистограммы слева направо. Ширина каждого прямоугольника равна 1.

Выведите площадь самого большого прямоугольника в гистограмме. Помните, что этот прямоугольник должен быть на общей базовой линии.

Input	Output
7 2 1 4 5 1 3 3	8

Очередь (queue)

L° *Реализация очереди*

Реализуйте структуру данных "очередь". Напишите программу, содержащую описание очереди и моделирующую работу очереди, реализовав все указанные здесь методы. Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Описание команд:

`push n` Добавить в очередь число `n` (значение `n` задается после команды). Программа должна вывести `ok`.

`pop` Удалить из очереди первый элемент. Программа должна вывести его значение.

`front` Программа должна вывести значение первого элемента, не удаляя его из очереди.

`size` Программа должна вывести количество элементов в очереди.

`clear` Программа должна очистить очередь и вывести `ok`.

`exit` Программа должна вывести `bye` и завершить работу.

Гарантируется, что набор входных команд удовлетворяет следующим требованиям: максимальное количество элементов в очереди в любой момент не превосходит 100, все команды pop и front корректны, то есть при их исполнении в очереди содержится хотя бы один элемент.
Время работы каждой операции `O(1)`, т.е. не должно зависеть от размера очереди.
Размер массива, в котором хранятся элементы очереди должен быть фиксирован при создании очереди и не меняться во время выполнения программы.
Замечание: для представления очереди удобно хранить такую следующие параметры: массив, в котором хранятся элементы очереди, индекс элемента-головы (`head`), текущее количество элементов в очереди (`size`) и максимальное количество элементов в очереди (`length` — фактически это размер массива).

Input	Output
size push 1 size push 2 size push 3 size exit	0 ok 1 ok 2 ok 3 bye

M° *Реализация очереди неограниченного размера с защитой от ошибок*

Реализуйте структуру данных "очередь". Напишите программу, содержащую описание очереди и моделирующую работу очереди, реализовав все указанные здесь методы. Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Описание команд:

`push n` Добавить в очередь число `n`. Программа должна вывести `ok`.

`pop` Удалить из очереди первый элемент. Программа должна вывести его значение.

`front` Программа должна вывести значение первого элемента, не удаляя его из очереди.

`size` Программа должна вывести количество элементов в очереди.

`clear` Программа должна очистить очередь и вывести `ok`.

`exit` Программа должна вывести `bye` и завершить работу.

Размер очереди в момент создания должен быть равен 100. При добавлении элемента в случае, если очередь полна, необходимо увеличивать размер массива для хранения элементов очереди. Увеличение должно быть умножением размера массива на некоторую подобранную константу. Перед исполнением операций `front` и `pop` программа должна проверять, содержится ли в очереди хотя бы один элемент. Если во входных данных встречается операция `front` или `pop`, и при этом очередь пуста, то программа должна вместо числового значения вывести строку `error`. Время работы каждой операции (в среднем) `O(1)`, т.е. не должно зависеть от размера очереди.

Input	Output
push 1 front exit	ok 1 bye
size push 1 size push 2 size push 3 size exit	0 ok 1 ok 2 ok 3 bye

N. *Игра в пьяницу*

В игре в пьяницу карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт — проигрывает.

Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту ("шестерка берет туза").

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

Напишите программу, которая моделирует игру в пьяницу и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9 (и только её).

Программа получает на вход две строки: первая строка содержит 5 чисел, разделенных пробелами — номера карт первого игрока, вторая – аналогично 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

Программа должна определить, кто выигрывает при данной раздаче, и вывести слово `first` или `second`, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 10^6 ходов игра не заканчивается, программа должна вывести слово `tie`.

Input	Output
1 3 5 7 9 2 4 6 8 0	second 5

Дек (deque)

O. *Реализация дека*

Реализуйте структуру данных "дек". Напишите программу, содержащую описание дека и моделирующую работу дека, реализовав все указанные здесь методы. Программа считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения каждой команды программа должна вывести одну строчку. Возможные команды для программы:

`push_front` Добавить (положить) в начало дека новый элемент. Программа должна вывести `ok`.

`push_back` Добавить (положить) в конец дека новый элемент. Программа должна вывести `ok`.

`pop_front` Извлечь из дека первый элемент. Программа должна вывести его значение.

`pop_back` Извлечь из дека последний элемент. Программа должна вывести его значение.

`front` Узнать значение первого элемента (не удаляя его). Программа должна вывести его значение.

`back` Узнать значение последнего элемента (не удаляя его). Программа должна вывести его значение.

`size` Вывести количество элементов в деке.

`clear` Очистить дек (удалить из него все элементы) и вывести `ok`.

exit Программа должна вывести `bye` и завершить работу.

Гарантируется, что количество элементов в деке в любой момент не превосходит 10^5. Все операции `pop_front`, `pop_back`, `front`, `back` всегда корректны. Время на выполнение каждой операции должно быть $O(1)$, то есть не зависеть от количества элементов в деке.

Программе на вход подаются команды управления деком, по одной на строке.

Требуется вывести протокол работы дека, по одному сообщению на строке.

Input	Output
size push_back 1 size push_back 2 size push_front 3 size exit	0 ok 1 ok 2 ok 3 bye

P. *Минимум в окошке*

Дан массив целых чисел размера N и натуральное число k — размер «окошка», в котором видно ровно k соседних элементов массива. «Окошко» двигается от крайнего левого положения (совпадают левый край массива и левый край "окошка") до крайнего правого положения (совпадают правый край массива и правый край окошка) — всего $N - k + 1$ позиций.

Требуется написать программу, выводящую значение минимума для всех последовательных положений “окошка”. Время работы линейно зависит от N и не зависит от k.

В первой строке входных данных содержатся два числа N и K ($1 \leqslant N \leqslant 150000, 1 \leqslant K \leqslant 10000, K \leqslant N$) — длины последовательности и “окна”, соответственно. На следующей строке находятся N чисел — сама последовательность.

Выходные данные должны содержать $N - K + 1$ строк — минимумы для каждого положения окна.

Указание: рассмотрите дек, в клотором хранятся пары `<значение, индекс>`. Инвариант: с правой стороны дека всегда находится текущий ответ, а внутри — только числа, которые *могут* стать ответом. Далее, перебирая элементы массива слева направо, требуется обновить состояние дека. В частности — рассмотреть случаи, когда текущий элемент меньше элемента с правого конца дека, меньше/больше элемента с правого конца дека.

Input	Output
7 3 1 3 2 4 5 3 1	1 2 2 3 1

Q. *Реклама*

Фирма NNN решила транслировать свой рекламный ролик в супермаркете XXX. Однако денег, запланированных на рекламную кампанию, хватило лишь на две трансляции ролика в течение одного рабочего дня. И при этом обязательно транслировать ровно два рекламных ролика в день.

Фирма собрала информацию о количестве покупателей в каждый момент некоторого дня. Менеджер по рекламе предположил, что и на следующий день покупатели будут приходить и уходить ровно в те же моменты времени.

Помогите ему определить моменты времени, когда нужно включить трансляцию рекламных роликов, чтобы как можно большее количество покупателей прослушало ролик.

Ролик длится ровно одну единицу времени. Для каждого момента времени известно количество покупателей, находящихся в магазине в этот момент. Между концом первой рекламы и началом следующей должна пройти как минимум $K - 1$ единица времени.

Первая строка содержит два числа N (моментов времени) и K (время совершения покупки одним покупателем). Гарантируется, что $N > K$ ($N, K \leqslant 200000$). Во второй строке содержится N чисел a_i — количество покупателей в момент времени i ($0 \leqslant a_i \leqslant 2 \cdot 10^9$).

Выведите единственное число — количество просмотревших рекламу покупателей.

Input	Output
5 2 3 5 1 4 2	9

Куча (heap)

Максимальная (минимальная) куча — бинарное дерево, каждый узел которого больше (меньше), чем любой из его потомков. Во всех задачах рассматриваются максимальные кучи. В задачах `Q` и `R` входные данные устроены следующим образом.

В первой строке задан размер кучи $N \in [1; 10^5]$.

Во второй строке вводится сама куча — N различных целых чисел, каждое из диапазона $[-10^9; 10^9]$. Гарантируется, что эти числа составляют корректную максимальную кучу.

В третьей строке вводится число M — количество запросов, $M \in [0; 10^5]$. В следующих M строках вводятся сами запросы — по одному в строке. Как устроен запрос, указано в условии каждой задачи.

Для простоты в первых трёх задачах будем иметь дело с кучей, где все элементы различны (однако на практике обеспечить выполнение этого условия почти нереально: при добавлении или изменении элемента невозможно за разумное время проверить его уникальность средствами самой кучи).

R. *Увеличение приоритета заданного элемента*

Запрос задаётся двумя целыми числами i и x. Требуется увеличить значение i-го элемента кучи на x и выполнить `Sift_Up` для восстановления кучи.

Гарантируется, что $i \in [1; N], x \geqslant 0$, новое значение $A[i] + x$ не превышает 10^9 и отличается от текущих значений всех остальных элементов кучи.

В качестве ответа на запрос требуется вывести одно число: сообщить, на каком месте массива оказался изменённый элемент после выполнения `Sift_Up`. Вывести в отдельной строке одно число — соответствующий индекс.

Кроме того, после выполнения всех запросов требуется вывести кучу в её конечном состоянии.

Input	Output
6 12 6 8 3 4 7 2 5 11 3 6	1 3 15 12 14 3 6 7

S. *Уменьшение приоритета заданного элемента*

Запрос задаётся двумя целыми числами i и x. Требуется уменьшить значение i-го элемента кучи на x и выполнить `Sift_Down` для восстановления кучи.

Гарантируется, что $i \in [1; N], x \geqslant 0$, новое значение $A[i] - x$ не превышает 10^9 и отличается от текущих значений всех остальных элементов кучи.

В качестве ответа на запрос требуется вывести одно число: сообщить, на каком месте массива оказался изменённый элемент после выполнения `Sift_Down`. Вывести в отдельной строке одно число — соответствующий индекс.

Кроме того, после выполнения всех запросов требуется вывести кучу в её конечном состоянии.

Input	Output
6 12 6 8 3 4 7 2 2 5 1 2	5 1 10 4 8 3 1 7

T. *Извлечение максимального элемента.*

Дана куча размера $N > 1$. Требуется $N - 1$ раз выполнить извлечение максимального элемента (`Extract_Max`). В процессе выполнения процедуры `Extract_Max` последний элемент кучи помещается в её корень, а затем просеивается вниз вызовом `Sift_Down`. После каждого выполнения процедуры `Extract_Max` нужно будет вывести индекс конечного положения этого элемента после просеивания, а также значение извлечённого максимального элемента.

Формат входных данных

В первой строке задан размер кучи $N \in [2; 10^5]$. Во второй строке вводится сама куча — N различных целых чисел, каждое из диапазона $[-10^9; 10^9]$. Гарантируется, что эти числа составляют корректную максимальную кучу.

Формат выходных данных

Требуется вывести $N - 1$ строку, в каждой — два числа. Первое — индекс конечного положения элемента после его просеивания; второе — значение извлечённого элемента.

Input	Output
6 12 6 8 3 4 7	3 12 3 8 2 7 1 6 1 4

В следующих задачах в куче могут встречаться равные элементы. Это создаёт некоторый произвол при выполнении процедур `Sift_Up` и `Sift_Down`. Для однозначности ответа его придётся ограничить. Поэтому введём два правила:

- Процедуры просеивания не должны перемещать элемент дальше, чем это действительно необходимо. Например, если `A[i]=A[2*i]`, то вызов `Sift_Up(2*i)` не должен менять местами эти два элемента (хотя их обмен и не испортит кучу, он бесполезен).
- Если при просеивании вниз можно перемещать рассматриваемый элемент как влево вниз, так и вправо вниз (это бывает, когда он меньше двух равных дочерних), то следует выбирать направление влево.

Второе правило довольно произвольно и введено лишь для обеспечения однозначности ответа. Первому правилу, напротив, должна удовлетворять любая разумная реализация кучи.

U. *Приоритетная очередь (добавление элемента, получение максимального)*

Требуется реализовать с помощью кучи приоритетную очередь, поддерживающую две операции: добавить элемент и извлечь максимальный элемент.

Формат входных данных

В первой строке вводятся два числа — максимальный размер приоритетной очереди N и количество запросов M ($1 \leqslant M, N \leqslant 10^5$). Далее идут M строк, в каждой строке — по одному запросу. Первое число в запросе задаёт его тип, остальные числа (если есть) — параметры запроса.

Тип 1 — извлечь максимальный (без параметров)

Тип 2 — добавить данный элемент в очередь. Запрос имеет один параметр — число из диапазона $[-10^9; 10^9]$

Формат выходных данных

В ответ на запрос типа 1 следует вывести:

- Если извлекать было нечего (очередь пуста), то -1.
- Иначе, как и в предыдущей задаче — два числа: первое — индекс конечного положения элемента после его просеивания (если же удалён был последний элемент и просеивать осталось нечего, вывести 0); второе — значение извлечённого элемента.

В ответ на запрос типа 2 следует вывести:

- Если добавить нельзя (нет места, поскольку в очереди уже N элементов), то вывести -1. (При этом куча не должна измениться).
- Иначе — индекс добавленного элемента.

Кроме того, после выполнения всех запросов требуется вывести кучу в её конечном состоянии.

Input	Output
4 7 1 2 9 2 4 2 9 2 9 2 7 1	-1 1 2 3 2 -1 2 9 9 4 9

V. *Приоритетная очередь (удаление)*

Условие этой задачи отличается от условия предыдущей лишь наличием ещё одного типа запроса — запроса на удаление заданного (не обязательно максимального) элемента. Это будет запрос типа 3 с единственным параметром, задающим индекс элемента, который требуется удалить из кучи. В ответ на запрос типа 3 следует вывести:

- -1, если элемента с таким индексом нет и удаление невозможно. При этом куча не должна измениться.
- Иначе —- значение удаленного элемента.

Гарантируется, что параметр является неотрицательным целым не больше 10^9.

Input	Output
4 10 1 2 9 2 4 2 9 2 9 2 7 1 3 4 2 1 3 3	-1 1 2 3 2 -1 2 9 -1 4 9 9 4 1

W. *Построение кучи просеиванием вверх*

Дан массив. Требуется преобразовать его в кучу с помощью процедуры просеивания вверх.

Формат входных данных:

В первой строке вводится длина массива N. В следующей строке идут элементы массива — N целых чисел, каждое из которых не превышает по модулю $10^9 (0 \leqslant N \leqslant 10^5)$.

Формат выходных данных:

N целых чисел — элементы кучи по порядку.

Input	Output
6 1 2 3 4 5 6	6 4 5 1 3 2

X. *Построение кучи просеиванием вниз*

Дан массив. Требуется преобразовать его в кучу с помощью процедуры просеивания вниз. Ввод-вывод устроен так же, как в предыдущей задаче.

Input	Output
6 1 2 3 4 5 6	6 5 3 4 2 1

Y. *Пирамидальная сортировка (heapsort) с выводом промежуточных результатов*

Требуется отсортировать по неубыванию с помощью изученного алгоритма целочисленный массив размера N, выводя также некоторые промежуточные результаты работы. А именно, должны быть выведены:

- первоначальная куча, построенная по условию задачи ZY, т.е. просеиванием вниз
- куча после удаления каждого элемента (то есть после каждой итерации внешнего цикла)
- отсортированный массив

Формат входных данных:

В первой строке вводится длина массива N. В следующей строке идут элементы массива — N целых чисел, каждое из которых не превышает по модулю $10^9 (1 \leqslant N \leqslant 500)$.

Формат выходных данных:

В первой строке должна быть выведена куча, построенная при помощи просеивания вниз, а в каждой из следующих $N-1$ строк должно быть выведено состояние кучи после удаления очередного элемента. Таким образом, в i-й строке должно быть выведено $N+1-i$ чисел). В последней ($N+1$-й) строке нужно вывести отсортированный массив (N чисел).

Input	Output
6 1 2 3 4 5 6	6 5 3 4 2 1 5 4 3 1 2 4 2 3 1 3 2 1 2 1 1 1 2 3 4 5 6

Z. *Платный калькулятор*

Фирма OISAC выпустила новую версию калькулятора. Этот калькулятор берет с пользователя деньги за совершаемые арифметические операции. Стоимость каждой операции в долларах равна 5% от числа, которое является результатом операции.

На этом калькуляторе требуется вычислить сумму N натуральных чисел (числа известны). Нетрудно заметить, что от того, в каком порядке мы будем складывать эти числа, иногда зависит, в какую сумму денег нам обойдется вычисление суммы чисел (тем самым, оказывается нарушен классический принцип «от перестановки мест слагаемых сумма не меняется»).

Например, пусть нам нужно сложить числа $10, 11, 1213$. Тогда если мы сначала сложим 10 и 11 (это обойдется нам в \$1.05), потом результат — с 12 (\$1.65), и затем — с 13 (\$2.3), то всего мы заплатим \$5, если же сначала отдельно сложить 10 и 11 (\$1.05), потом — 12 и 13 (\$1.25) и, наконец, сложить между собой два полученных числа (\$2.3), то в итоге мы заплатим лишь \$4.6.

Напишите программу, которая будет определять, за какую минимальную сумму денег можно найти сумму данных N чисел.

Input	Output
4 10 11 12 13	4.60
2 1 1	0.10

ZA. *Игры со строками*

Во время игры в Scrabble Дуня любит играть со своими буквами, перебирая их и меняя в задумчивости местами. Буквы стоят на специальной подставке, для каждой буквы предусмотрено отдельная позиция. Все позиции пронумерованы от 0 до $N - 1$, где N — количество букв.

Вам дана строка и целое неотрицательное число K, имеющее следующий смысл: после того, как Дуня поменяла порядок некоторых букв из первоначальной строки, известно, что каждая буква изменила свою позицию не более, чем на K.

Требуется по входным данным вывести лексикографически минимальную строку, которая могла получиться при указанных ограничениях.

Input	Output
ABRACADABRA 2	AABARACBDAR