Класс Heap.

В этом листочке нужно реализовать и использовать класс `Heap`, представляющий из себя бинарную кучу.

A. *Предки, потомки, братья и сестры*
Реализуйте функции `left(i)`, `right(i)`, `parent(i)`, `sibling(i)`, которые по индексу элемента находят индекс его левого потомка, правого потомка, предка, брата соответственно. Функции должны быть членами класса `Heap`.

B. *Просеивание вверх*
Реализуйте функцию `sift_up(self, i)`, которая просеивает элемент с заданным индексом вверх.

C. *Добавление элемента*
Реализуйте метод `add(self, element)`, который добавит в кучу указанный элемент.

D. *Создание пустой кучи*
Реализуйте метод `__init__(self)`, который пока создает только пустую кучу.
Реализуйте метод `__len__(self)`, который будет возвращать количество элементов в куче.

E. *Печать*
Добавьте возможность распечатывать кучу (реализуйте метод `__str__`). Куча должна распечатываться по ярусам — на первой строчке только корень, на второй его левый и правый потомки, и так далее. Поймите, какие нужно делать отступы в каждой из строчек, чтобы куча смотрелась красиво (предок был над своими потомками).

F. *Просеивание вниз*
Реализуйте функцию `sift_down(self, i)`, которая просеивает элемент с заданным индексом вниз.

G. *Корень зла*
Реализуйте функции `get_root(self)`, которая возвращает элемент в вершине кучи, и `pop_root(self)`, которая возвращает элемент в вершине кучи и удаляет его из кучи.

H. *Замена элемента*
Реализуйте функцию `replace(self, new_elem)`, которая удаляет корень кучи и добавляет вместо него элемент `new_elem`. Очевидным образом, функцию можно реализовать как композицию `pop_root()` и `add(new_elem)`. Однако при таком подходе придется по одному разу вызвать `sift_down()` и `sift_up()`. Цель этой функции — немного сэкономить операции и обойтись лишь одним `sift_down()`.

I. *Создание кучи*
Поменяйте функцию `__init__` так, чтобы она могла принимать на вход массив значений. При этом возможность создать пустую кучу должна остаться (сделайте `__init__(self, values = None)`).
То есть должны работать оба варианта
```
my_heap = Heap()
my_heap = Heap([3, 4, 1, -7])
```
Создание кучи из массива должно работать за $O(n)$, а не за $O(n \log n)$.

J. *Больше/меньше — все равно*
Добавьте в класс `Heap` поле `cmp`, которое будет отвечать за сравнение элементов. Сделайте `__init__(self, values = None, cmp = None)`.

K. *Пирамидальная сортировка*

Напишите сортировку кучей — `HeapSort(array, cmp = None)`. Это должна быть отдельная функция (использующая кучу внутри), а не метод класса `Heap`. Если параметр `cmp` не указан (по умолчанию равен `None`), функция `HeapSort` должна сортировать по неубыванию.

Функция не должна изменять входной массив, а возвращать результат.

Input	Output
`x = [2, 3, 1]` `print(HeapSort(x))`	`[1, 2, 3]`
`x = [2, -3, -5]` `print(HeapSort(x, cmp = lambda a, b: abs(a) < abs(b)))`	`[2, -3, -5]`

L. *Сложение куч*

Реализуйте метод `__add__(self, another)`, который из двух куч создаст одну. Метод должен работать за $O(M + N)$, где M и N — размеры объединяемых куч.

M. *Удаление по индексу*

Реализуйте функцию, удаляющую из кучи элемент с известным индексом.

N. *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`

O. *Максимум в окне*

Дан массив чисел и два индекса l, r (начало окна и конец окна — левый включается, правый нет). Изначально $l = 0$, $r = 1$ — окно состоит из первого элемента. На вход программе подается последовательность букв L и R, каждая буква обозначает сдвиг соответствующей границы окна. После каждого сдвига требуется вывести максимум в текущем окне. Гарантируется, что левая граница никогда не будет правее правой.

В первой строке входного потока задано число N ($1 \leqslant N \leqslant 10^5$) — размер массива. Во второй строке N целых чисел от -10^9 до 10^9 — сам массив. В третьей строке указано число M ($0 \leqslant M \leqslant 2N - 2$) — количество перемещений. В четвёртой строке — M символов `L` или `R`, разделенных пробелами. `L` означает, что нужно сдвинуть l вправо, `R` — что нужно сдвинуть r вправо.

Input	Output

`10`
`1 4 2 3 5 8 6 7 9 10`
`12`
`R R L R R R L L L R L L`
	`4 4 4 4 5 8 8 8 8 8 8 6`

P. *k-я порядковая статистика в окне*

Дан массив чисел и два индекса l, r (начало окна и конец окна — левый включается, правый нет). Изначально $l = 0$, $r = 1$ — окно состоит из первого элемента. На вход программе подается последовательность букв L и R, каждая буква означает сдвиг соответствующей границы окна. После каждого сдвига требуется вывести k-й по величине элемент в текущем окне. Гарантируется, что левая граница никогда не будет правее правой. Если в окне меньше k элементов — выведите `-1`.

В первой строке входного потока задано число N ($1 \leqslant N \leqslant 10^5$) — размер массива. Во второй строке N целых чисел от -10^9 до 10^9 — сам массив. В третьей строке указано число M ($0 \leqslant M \leqslant 2N - 2$) — количество перемещений. В четвёртой строке — M символов `L` или `R`, разделенных пробелами. `L` означает, что нужно сдвинуть l вправо, `R` — что нужно сдвинуть r вправо.

Выведите ровно m строк, в каждой — ровно по одному целому числу. После выполнения каждой из операций нужно вывести k-е в порядке возрастания число среди всех чисел от l до r включительно, либо -1, если всего чисел от l до r меньше, чем k.

Input	Output
7 4 2 4 2 1 3 6 5 7 R R L L	4 2 2 -1

Q. *Таможня*

Недавно Министерство Налогов и Сборов выделило отделению определённую сумму денег на установку новых аппаратов на таможне для автоматического досмотра грузов. Естественно, средства были выделены с таким расчётом, чтобы грузы теперь находились на таможне ровно столько времени, сколько требуется непосредственно на их досмотр.
В руках шефа каким-то образом оказались сведения о надвигающейся ревизии — список из N грузов, которые будут контролироваться Министерством. Для каждого груза известны время его прибытия и время, требуемое аппарату для обработки этого груза. Шеф дал таможенникам задание по этим данным определить, какое минимальное количество аппаратов необходимо заказать на заводе, чтобы все грузы Министерства начинали досматриваться сразу после прибытия. Необходимо учесть, что конструкция тех аппаратов, которые было решено установить, не позволяет обрабатывать два груза одновременно на одном аппарате. Напишите программу, которая поможет сотрудникам таможни справиться с этой задачей.

В первой строке входного файла задано число N ($0 \leqslant N \leqslant 50000$). На следующих N строках находится по 2 целых положительных числа T_i и L_i — время прибытия соответствующего груза и время, требуемое для его обработки ($1 \leqslant T_i \leqslant 10^6, 1 \leqslant L_i \leqslant 10^6$).

В выходной файл выведите одно число —- наименьшее количество аппаратов, которое нужно установить, чтобы не вызвать подозрений у Министерства.

Input	Output
3 3 2 4 2 5 2	2
5 13 4 15 1 11 5 12 3 10 3	3