A. Пирамидальная сортировка

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Язык | Ограничение времени | Ограничение памяти | Ввод | Вывод |
| Все языки | 5 секунд | 256Mb | стандартный ввод или input.txt | стандартный вывод или output.txt |
| C# (MS .Net 5.0)+ASP | 1.5 секунд | 256Mb |
| Oracle Java 8 | 1.5 секунд | 256Mb |
| OpenJDK Java 11 | 1.5 секунд | 256Mb |
| GNU c++17 7.3 | 0.7 секунд | 64Mb |

**В данной задаче необходимо реализовать сортировку кучей. При этом кучу необходимо реализовать самостоятельно, использовать имеющиеся в языке реализации нельзя. Сначала рекомендуется решить задачи про просеивание вниз и вверх.**

Тимофей решил организовать соревнование по спортивному программированию, чтобы найти талантливых стажёров. Задачи подобраны, участники зарегистрированы, тесты написаны. Осталось придумать, как в конце соревнования будет определяться победитель.

Каждый участник имеет уникальный логин. Когда соревнование закончится, к нему будут привязаны два показателя: количество решённых задач *Pi* и размер штрафа *Fi*. Штраф начисляется за неудачные попытки и время, затраченное на задачу.

Тимофей решил сортировать таблицу результатов следующим образом: при сравнении двух участников выше будет идти тот, у которого решено больше задач. При равенстве числа решённых задач первым идёт участник с меньшим штрафом. Если же и штрафы совпадают, то первым будет тот, у которого логин идёт раньше в алфавитном (лексикографическом) порядке.

Тимофей заказал толстовки для победителей и накануне поехал за ними в магазин. В своё отсутствие он поручил вам реализовать алгоритм сортировки кучей (*англ.* Heapsort) для таблицы результатов.

Формат ввода

В первой строке задано число участников *n*, *1 ≤ n ≤ 100 000*.  
В каждой из следующих *n* строк задана информация про одного из участников.  
*i*-й участник описывается тремя параметрами:

* уникальным логином (строкой из маленьких латинских букв длиной не более *20*)
* числом решённых задач *Pi*
* штрафом *Fi*

*Fi* и *Pi* — целые числа, лежащие в диапазоне от *0* до *109*.

Формат вывода

Для отсортированного списка участников выведите по порядку их логины по одному в строке.

Пример 1

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5  alla 4 100  gena 6 1000  gosha 2 90  rita 2 90  timofey 4 80 | gena  timofey  alla  gosha  rita |

Пример 2

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5  alla 0 0  gena 0 0  gosha 0 0  rita 0 0  timofey 0 0 | alla  gena  gosha  rita  timofey |

# B. Удали узел

|  |  |
| --- | --- |
| Ограничение времени | 3 секунды |
| Ограничение памяти | 128Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Дано бинарное дерево поиска, в котором хранятся ключи. Ключи — уникальные целые числа. Найдите вершину с заданным ключом и удалите её из дерева так, чтобы дерево осталось корректным бинарным деревом поиска. Если ключа в дереве нет, то изменять дерево не надо.

На вход вашей функции подаётся корень дерева и ключ, который надо удалить. Функция должна вернуть корень изменённого дерева. Сложность удаления узла должна составлять O(h), где h –— высота дерева.

Создавать новые вершины (вдруг очень захочется) нельзя.

## Формат ввода

Ключи дерева – натуральные числа, не превышающие 109. В итоговом решении не надо определять свою структуру/свой класс, описывающий вершину дерева.

Мы рекомендуем воспользоваться заготовками кода для данной задачи, расположенными по [ссылке](https://disk.yandex.ru/d/fPDFAzBxY_GxZw).

## Формат вывода

По умолчанию выбран компилятор Make. Решение нужно отправлять в виде файла с расширением, которое соответствует вашему языку программирования. Если вы пишете на Java, имя файла должно быть Solution.java, для C# – Solution.cs. Для остальных языков назовите файл my\_solution.ext, заменив ext на необходимое расширение. Ниже приведены сигнатуры функций, которые надо реализовать.

**Python**

*# do not declare Node in your submit-file*   
**class** Node:   
    **def** \_\_init\_\_(self, left=None, right=None, value=0):   
        self.value = value   
        self.right = right   
        self.left = left   
   
**def** remove(root: Node, key: **int**) -> Node

**C++**

*// do not declare Node in your submit-file*   
**struct** Node {   
    Node∗ left;   
    Node∗ right;   
    **int** value;   
};   
**#include** "solution.h" *// Attention!*   
Node∗ remove(Node∗ root, **int** key);

**Java**

*// do not declare Node in your submit-file*   
**public** **class** Node {   
    **private** Node left;   
    **private** Node right;   
    **private** **int** value;   
    **public** **int** getValue();   
    **public** Node getRight();   
    **public** Node getLeft();   
    **public** **void** setValue(**int** value);   
    **public** **void** setRight(Node right);   
    **public** **void** setLeft(Node left);   
}   
**public** **class** Solution {   
        **public** **static** Node remove(Node root, **int** key);   
}

**Go**

*// do not declare Node in your submit-file*   
type Node **struct** {   
        value    **int**   
        left   ∗Node   
        right  ∗Node   
}   
   
package main   
func remove(node ∗Node, key **int**) ∗Node