Контрольная работа-3. Массивы и сортировка. 8Д.

В каждой из предложенных ниже задач, нужно реализовать те функции, которые описаны в условии. Функции должны называться, иметь точно такие же аргументы и возвращать те же значения, как написано в условии. В противном случае, система проверки не зачтет ваше решение.
Прототипы функций, которые нужно реализовать, писать **НЕ** надо. Система сама вставит их.
Также **НЕ** надо писать `main()`.
Если для реализации нужных функций вам понадобятся вспомогательные, используйте их. Для них заголовки писать можно и нужно.
К каждому решению система автоматически подключит библиотеки `stdio.h` и `iostream`. Если вам понадобятся другие библиотеки, подключайте их самостоятельно. Не стоит подключать `TXLib.h`, система вас не поймет.
Запрещается использовать стандартные функции сортировки, реализованные не вами.

A. *Три товарища.* Мария Ивановна проводила контрольную работу по информатике. После проверки работ она обнаружила, что каждый из учеников ''совещался'' со своими соседями (в классе компьютеры расположены по кругу, нулевой ученик соседствует с последним). Поэтому она решила изменить оценки следующим образом: каждому ученику поставить максимум из его оценки и оценки его соседей, деленный на 3. Оценка должна быть натуральным числом от 1 до 10^9. Поэтому, если максимум не делится на 3 нацело, то оценка округляется вниз до ближайшего целого числа. А если оценка выходит меньше 1, то в журнал будет проставлена единица.
Напишите, функцию
`void MariaReMark(int marks[], int size),`
которая поможет Марии Ивановне изменить массив изначальных оценок по описанному выше алгоритму.
Также необходимо реализовать функцию
`void SetMarks(int const array[], int size),`
которая печатает массив `array` на одной строчке через пробел.

B. *Три товарища-2.* После переписывания контрольной, Мария Ивановна поняла, что школьники все еще продолжали ''совещаться'' с соседями. На этот раз она решила изменять оценки следующим образом: проставить каждому школьнику среднее арифметическое его оценки и оценки его соседей, уменьшенное на некоторое число. То есть,

$$new\_mark_i = \frac{mark_{i-1} + mark_i + mark_{i+1}}{3} - penalty.$$

Параметр *penalty* общий для всего класса. Правила округления оценок остаются из предыдущей задачи. Напишите, функцию
`void MariaReMark2(int marks[], int size, int penalty),`
которая поможет Марии Ивановне изменить массив изначальных оценок по описанному выше алгоритму.
Также необходимо реализовать функцию
`void SetMarks(int const array[], int size),`
которая печатает массив `array` на одной строчке через пробел.
Замечание. Обратите внимание, что $3 \cdot 10^9 > 2^{31} - 1$.

C. *Сортировка по последней цифре.* Реализуйте функцию
`void LastDigitSort(int array[], int size),`
которая отсортирует массив `array` по последней цифре. Например, при таком сравнении $9 > 85$, $12 > 1000000$.
Сортировка не должна поменять местами элементы, у которых одинаковая последняя цифра. То есть если мы сортируем массив $\{12, 101, 2\}$, должно получиться $\{101, 12, 2\}$. Ответ $\{101, 2, 12\}$ будет неправильным, поскольку «равные» элементы 12 и 2 поменяли порядок относительно друг друга.
Все элементы массива — натуральные числа.
Также необходимо реализовать функцию
`void PrintArray(int const array[], int size),`
которая печатает массив `array` на одной строчке через пробел.
Замечание. Последнюю цифру числа можно определить при помощи операции взятия остатка — $n\%10$.

D. *Юный геометр.* Ване подарили набор «Юный геометр», который состоит из палочек различной длины. Ваня решил составлять из этих палочек треугольники. Назовем набор *бракованным*, если в нем есть палочка, которая не может быть использована при построении хотя бы одного треугольника.
Помогите Ване определить является ли его набор бракованным.
Для начала он считает, что нужно отсортировать массив. Поэтому нужна функция
`void MySort(int array[], int size).`
После понадобится функция
`bool NoDefects(int const array[], int size),`
которая вернет `false`, если отсортированный массив `array` является бракованным, и вернет `true`, если брака не обнаружено.
Замечание. Напомним, что из трех отрезков можно составить треугольник тогда и только тогда, когда длина каждого отрезка меньше суммы двух оставшихся.

E. *Великий волшебник.* У Вани есть набор карточек, на которых написаны числа. Он научился показывать следующий фокус — перемешивать карты, затем выкладывать поочередно по одной карте сверху и снизу колоды, пока карты не кончатся. В результате чего последовательность чисел, написанных на выложенных картах, оказывается отсортированной по возрастанию.
Реализуйте функцию
`void StrangeSort(int array[], int size),`
которая сортирует элементы массива в таком странном порядке. Минимальный элемент должен стоять на нулевом месте, второй минимум на последнем, третий минимум на первом, и так далее.
Также необходимо реализовать функцию
`void PrintArray(int const array[], int size),`
которая печатает массив `array` на одной строчке через пробел.

Решение задачи D. Заметим, что если отрезок, участвующий в составлении тругольника является либо максимальной его стороной, либо минимальной, либо средней.
Также отметим, что из трех отрезков можно составить треугольник тогда и только тогда, когда максимальный из этих отрезков меньше суммы двух других.
Обозначим за $\{a_i\}$ отсортированный массив из длин.
Сначала найдем все отрезки, которые могут являться максимальными сторонами треугольника. Пусть некоторый отрезок a_j может быть максимальной стороной некоторого треугольника. Две другие стороны которого a_k и a_l, причем $k < l < j$. Значит, $a_k + a_l > a_j$. Но тогда должно выполняться и неравенство $a_{j-2} + a_{j-1} > a_j$, поскольку $k \geqslant j-2, l \geqslant j-1$.
Таким образом, чтобы найти все отрезки, которые могут быть максимальными в некоторых треугольниках, нам достаточно пройти по массиву и проверить, выполняется ли неравенство $a_i < a_{i-1} + a_{i-2}$ для каждого i. Для того чтобы сохранить результат этих проверок нужно ввести дополнительный массив
`bool CanBeMax[size];`
Соответсвенно, если $a_i < a_{i-1} + a_{i-2}$ мы сделаем `CanBeMax[i] = true`. В противном случае, определим `CanBeMax[i] = false`.
Аналогично со случаем максимальной стороны можно рассмотреть и случай средней. Если немного поразмыслить, можно понять, что эти два случая можно рассматривать одновременно. Таким образом, можно заполнить массив
`bool CanBeMaxOrMed[size]`, который имеет понятный смысл.
Случай про минимальную сторону разбирается сложнее. Некий отрезок a_j является минимальной стороной треугольника тогда и только тогда, когда найдутся два отрезка a_k и a_l, такие что $l > k > j$ и $a_j + a_k > a_l$. Или другими словами, $a_j > a_l - a_k$. Можно полагать, что $k = l - 1$, поскольку $a_l - a_{l-1} < a_l - a_k$. Таким образом, чтобы заполнить массив `bool CanBeMin[size]`, нам для каждого i нужно уметь определять минимальную из разниц вида $a_l - a_{l-1}$ для всех $l > j$. Этот минимум, конечно, можно находить каждый раз в отдельной функции, но это потребует много операций. Поэтому можно создать еще один дополнительный массив `int MinDiff[size - 1]`. `MinDiff[i]` будет равен минимальному элементу из множества
$\{a_{i+1} - a_i, a_{i+2} - a_{i+1}, \ldots, a_{size} - a_{size-1}\}$. заполнить его можно следующим образом:

```
int MinDiff[size - 1];
int curMin = a[size - 1] - a[size -2];
for (int i = size - 2; i > -1; i -= 1)
    {
    curMin = min(a[i + 1] - a[i], curMin);
    MinDiff[i] = curMin;
    }
```

Далее массив `bool CanBeMin[size]` можно заполнить следующим образом:

```
CanBeMin[i] = a[i] > MinDiff[i + 1];
```

После того, как мы заполнили массивы `bool CanBeMin[size]` и `bool CanBeMaxOrMed[size]` решение очевидно.