# Задача A. Сумма минимумов отрезков

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 5 секунд |
| Ограничение по памяти: | 256 мегабайт |

В этой задаче вам надо написать не целую программу, а одну единственную функцию
`unsigned SegMinSum(const unsigned *a, const unsigned *b, int n);`
Функция принимает 2 массива $a$ и $b$ (эти массивы могут пересекаться в памяти) и должна посчитать

$$\sum_{i=0}^{n-1} min(a[i], b[i])$$

То есть берётся минимум по первым элементам $a$ и $b$, это складывается с минимумом по вторым элементам $a$ и $b$, это складывается с минимумом по третим элементам $a$ и $b$ и так далее, всего обрабатывается $n$ элементов.

Например `SegMinSum({1, 2, 3}, {3, 2, 1}, 3)` $=$ 4, потому что берётся $min(1,3) + min(2,2) + min(3,1) = 4$.

Функция суммарно вызовется не более $200\,000$ раз, $n$ не превышает $200\,000$.

## Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>
```

Никакие прагмы писать не надо, они работать всё равно не будут.

# Задача B. Прибавление арифметической прогрессии

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 5 секунд |
| Ограничение по памяти: | 256 мегабайт |

В этой задаче вам надо написать не целую программу, а две функции:

1. `unsigned GetMin(const unsigned *a, int n);`

   Эта функция считает минимум в массиве $a$ длиной $n$, то есть считает минимум от элементов $a[0], a[1], a[2], \ldots, a[n-1]$. Например `GetMin({2, 1, 3}, 3)` $= 1$.

2. `void AddProg(unsigned *a, int n, unsigned k);`

   Эта функция прибавляет к массиву $a$ длиной $n$ арифметическую прогрессию с коэффициентом $k$. То есть к $a[0]$ прибавляется $k$, к $a[1]$ прибавляется $k \cdot 2$, к $a[2]$ прибавляется $k \cdot 3$, и так далее, к $a[n-1]$ прибавляется $k \cdot n$. Функция должна модифицировать сам массив $a$. Прибавления делаются по модулю $2^{32}$. Например $AddProg(\{1, 2, 3\}, 3, 1)$ сделает массив равным $\{2, 3, 4\}$

Функции суммарно вызовутся не более 200 000 раз, $n$ не превышает 200 000.

## Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>
```

Никакие прагмы писать не надо, они работать всё равно не будут.

# Задача C. Сумма на отрезке

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 8 секунд |
| Ограничение по памяти: | 256 мегабайт |

В этой задаче вам надо написать не целую программу, а две функции:

1. `void Prepair(int n, const unsigned short *s);`

   Эта функция подготовится отвечать на запросы сумма на отрезке в массиве $s$ длинной $n$. Массив состоит из чисел типа `unsigned short`, сумму надо выводить по модулю $2^{16}$. Массив $s$ надо копировать.

2. `unsigned short GetSum(int l, int r);`

   Эта функция ищет сумма на отрезке массива $s$ с позиции $l$ по позицию $r$. Сумму надо выводить по модулю $2^{16}$.

В начале будет вызвана функция *Prepair*, а потом будут вызываться какие-то из этих функций. При этом функция *Prepair* может быть вызвана несколько раз. Например возможна такая последовательность вызовов функций:

- `Prepair(3, {1, 2, 3})`
- `GetSum(1, 3)` — надо вернуть 6.
- `GetSum(2, 2)` — надо вернуть 2.
- `Prepair(4, {1, 0, 1, 1})`
- `GetSum(1, 4)` — надо вернуть 3.

Суммарно функции вызываются не более $10^7$ раз, при этом в одном вызове функции *Prepair* $n$ не превышает $10^7$, суммарно все $n$ по функциям *Prepair* не превышают $10^{10}$. По функциям *GetSum* сумма всех $r - l$ не превышает $10^{14}$.

## Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>
```

Никакие прагмы писать не надо, они работать всё равно не будут.

# Задача D. Кол-во на отрезке, больших k

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 2.8 секунд |
| Ограничение по памяти: | 4 мегабайта |

Дан массив размера $n$, есть $m$ запросов посчитать кол-во чисел на отрезке с $l_i$ по $r_i$, больших $k_i$. Обратите внимание на ограничение по памяти. Чтобы не словить ML, вы должны завести один глобальный массив типа `int` размера 300 000, на дополнительные массивы памяти не хватит. Так как `iostream` требует большое количество памяти, то вы должны всё считывать и выводить с помощью `printf` и `scanf`.

## Формат входных данных

В первой строке даны числа $n$ и $m$ ($1 \leqslant n, m \leqslant 3 \cdot 10^5$).

Во второй строке $n$ чисел — сам массив (числа от 0 до $10^9$).

В следующих $m$ строках даны запросы, в строке вводятся $l_i$, $r_i$, $k$.

## Формат выходных данных

Выведите $m$ строк — ответы на запросы.

## Пример

| стандартный ввод | стандартный вывод |
|---|---|
| 5 3<br>1 2 3 4 5<br>1 5 3<br>2 4 4<br>1 3 0 | 2<br>0<br>3 |

## Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>}
#pragma GCC target("avx,avx2")
```

Только этих прагм не хватит, попробуйте поразвлекаться с другими прагмами и всё таки загнать эту задачу.

# Задача E. Прибавление отрезка к отрезку

| | |
|---|---|
| Имя входного файла: | стандартный ввод |
| Имя выходного файла: | стандартный вывод |
| Ограничение по времени: | 3.7 секунд |
| Ограничение по памяти: | 256 мегабайт |

Дан массив $s$ размера $n$. Есть $m$ запросов поэлементно прибавить отрезок с $a_i$ длинной $k_i$ к отрезку массива с $b_i$ длинной $k_i$. То есть $s_b$ увеличивается на $s_a$, $s_{b+1}$ увеличивается на $s_{a+1}$, $s_{b+2}$ увеличивается на $s_{a+2}$ и так далее, $s_{b+k-1}$ увеличивается на $s_{a+k-1}$. Отрезки одного запроса не пересекаются. Все прибавления надо делать по модулю $2^{32}$.

## Формат входных данных

В первой строке вводится числа $n$ и $m$ ($1 \leqslant n, m \leqslant 200\,000$).

Во второй строке вводится массив (числа от 0 до $2^{32} - 1$.

В следующих $m$ строках по 3 числа, в $i$-й строке 3 числа $a_i$, $b_i$, $k_i$. Отрезки одного запроса не пересекаются.

## Формат выходных данных

В единственной строке выведите полученный массив.

## Пример

| стандартный ввод | стандартный вывод |
|---|---|
| 5 3<br>1 1 1 1 1<br>1 3 2<br>4 1 2<br>2 5 1 | 3 2 2 2 3 |

## Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>}
#pragma GCC target("avx,avx2")
```

Только этих прагм не хватит, попробуйте поразвлекаться с другими прагмами и всё таки загнать эту задачу.

# Задача F. Сумма на отрезке с изменениями

| | |
|---|---|
| Имя входного файла: | `стандартный ввод` |
| Имя выходного файла: | `стандартный вывод` |
| Ограничение по времени: | 15 секунд |
| Ограничение по памяти: | 256 мегабайт |

Эта задача почти как задача «Сумма на отрезке», однако добавляется функция изменения. В этой задаче вам надо написать уже 3 функции:

1. `void Prepair(int n, const unsigned short *s);`

   Эта функция подготовится отвечать на запросы сумма на отрезке в массиве $s$ длинной $n$. Массив состоит из чисел типа `unsigned short`, сумму надо выводить по модулю $2^{16}$. Массив $s$ надо копировать.

2. `unsigned short GetSum(int l, int r);`

   Эта функция ищет сумма на отрезке массива $s$ с позиции $l$ по позицию $r$. Сумму надо выводить по модулю $2^{16}$.

3. `void Change(int a, unsigned short x);`

   Эта функция должна изменить число на позиции $a$ на значение $x$.

В начале будет вызвана функция $Prepair$, а потом будут вызываться какие-то из этих функций. При этом функция $Prepair$ может быть вызвана несколько раз. Например возможна такая последовательность вызовов функций:

- `Prepair(3, {1, 2, 3})`
- `GetSum(1, 3)` — надо вернуть 6.
- `Change(2, 1)` — теперь массив имеет вид $\{1, 1, 3\}$.
- `GetSum(2, 2)` — надо вернуть 1.
- `Prepair(4, {1, 0, 1, 1})`
- `GetSum(1, 4)` — надо вернуть 3.

Суммарно функции вызываются не более $10^7$ раз, при этом в одном вызове функции $Prepair$ $n$ не превышает $10^7$, суммарно все $n$ по функциям $Prepair$ не превышают $10^{10}$. По функциям $GetSum$ сумма всех $r - l$ не превышает $10^{14}$.

## Замечание

Про то как работает AVX и как его примерно использовать написано здесь: https://www.codeproject.com/Articles/874396/Crunching-Numbers-with-AVX-and-AVX

Все инструкции AVX написанны здесь:

https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX,AVX2

Не забудьте в начале программы написать

```
#include <immintrin.h>
```

Никакие прагмы писать не надо, они работать всё равно не будут.