A\*. Бинпоиск. Шаблоны

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 256.0 Мб |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Пришлите заголовочный файл с определениями трех шаблонных функций для работы с отсортированными последовательностями. Каждая функция принимает три аргумента: begin*begin* - указатель на начало последовательности, end*end* - указатель на конец последовательности (элемент следующий за последним), value*value* - запрос.

* *BinarySearch*
  + функция, осуществляющая проверку наличия элемента value*value* в отсортированной последовательности [begin,end)[*begin*,*end*). Возвращает true*true*, если и только если на полуинтервале [begin,end)[*begin*,*end*) имеется значение value*value*.
* *LowerBound*
  + функция, осуществляющая поиск первого элемента не меньшего (≥≥) value*value* на полуинтервале [begin,end)[*begin*,*end*). Возвращает указатель на найденный элемент, либо end*end*, если такого нет.
* *UpperBound*
  + функция, осуществляющая поиск первого элемента строго большего (>>) value*value* на полуинтервале [begin,end)[*begin*,*end*). Возвращает указатель на найденный элемент, либо end*end*, если такого нет.

**Важно:** используйте только сравнение на меньше (<<) и избегайте лишних копирований элементов. То есть считайте, что элементы последовательности не копируемы и могут быть сравнены только с помощью операции <<.

**Пример:**

*int array[5]{1, 2, 3, 3, 5};*

*BinarySearch(array + 3, array + 4, 2); // false*

*LowerBound(array + 1, array + 5, 3); // указатель на 2й элемент*

*UpperBound(array, array + 2, 4); // указатель на 2й элемент (на [0, 2) такого элемента нет)*

Пришлите на проверку заголовочный файл, содержащий ТОЛЬКО определения шаблонов. Он будет включен в cpp файл с тестами автоматически (main писать не нужно!).

Формат ввода

**Данная информация нужна только для понимания примера. Обработку пользовательского ввода писать не нужно.** **Вместо указателей в примере вводятся и выводятся индексы элементов.**

На первой строчке подается число элементов в последовательности, а далее сами элементы.

В следующих строках следует число запросов и сами запросы в формате (begin, end, value).

Формат вывода

**Данная информация нужна только для понимания примера. Ваши функции ничего выводить не должны.**

Выводятся 3 числа - результаты вызова *BinarySearch*, *LowerBound* и *UpperBound* соответственно.

Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5  1 2 3 4 5  5  0 3 4  0 3 3  1 5 6  1 2 2  0 5 5 | NO 3 3  YES 2 3  NO 4 4  YES 0 1  YES 4 5 |

# B\*. Слияние. Шаблоны

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 66.9 Мб |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Реализуйте шаблонную функцию Merge, которая принимает две отсортированные последовательности и сливает их одну большую отсортированную.

Функция должна принимать 5 аргументов: first\_begin, first\_end - указатели на начало и конец первой последовательности; second\_begin, second\_end - начало и конец второй последовательности; out - указатель на начало итоговой последовательности (куда записывать ответ).

Последовательности first, second и out могут хранить значения разных типов!

**Пример:**

int16\_t first[2]{1, 4};

int32\_t second[3]{2, 3, 5};

int64\_t out[5];

Merge(first, first + 2, second, second + 3, out); // out = {1, 2, 3, 4, 5}

Ваш код будет проверяться на скрытых от вас юнит-тестах, проверяющих корректность шаблонной реализации. Требование к реализации: из операций сравнения можно использовать только операцию "<".

Пришлите на проверку заголовочный файл, содержащий ТОЛЬКО определение шаблона. Он будет включен в cpp файл с тестами автоматически (main писать не нужно!).

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 3  1 4 6  4  2 3 5 7 | 1 2 3 4 5 6 7 |

# C\*. Сортировка. Шаблоны

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 256.0 Мб |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

В данной задаче вам необходимо реализовать шаблонную функцию Sort, сортирующую переданную последовательность. Функция должна принимать два аргумента: begin - указатель на начало последовательности, end - указатель на конец последовательности (элемент следующий за последним).

**Пример:**

int array[5]{3, 5, 1, 4, 2};

Sort(array + 1, array + 4); // array = {3, 1, 4, 5, 2}

Ваш код будет проверяться на скрытых от вас юнит-тестах, проверяющих корректность шаблонной реализации. Требование к реализации: из операций сравнения можно использовать только операцию "<".

Пришлите на проверку заголовочный файл, содержащий ТОЛЬКО определение шаблона. Он будет включен в cpp файл с тестами автоматически (main писать не нужно!).

## Формат ввода

На вход подается массив размера n≤105*n*≤105.

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5  5 2 3 1 4 | 1 2 3 4 5 |

# D\*. k-я порядковая статистика. Шаблоны

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 256.0 Мб |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Реализуйте шаблонную функцию KthElement, которая упорядочивает элементы последовательности так, чтобы на k*k*-м месте (в 0 индексации) стоял k*k*-й по величине элемент последовательности.

Функция должна принимать 3 аргумента: begin - указатель на начало последовательности; kth - указатель на k*k*-ю позицию; end - указатель на конец последовательности.

Среднее время работы алгоритма должно быть линейным от длины полуинтервала \*[begin, end)\*.

**Пример:**

int array[5]{2, 3, 5, 4, 1}

KthElement(array, array + 2, array + 5); // array = {1, 2, 3, 5, 4}

Ваш код будет проверяться на скрытых от вас юнит-тестах, проверяющих корректность шаблонной реализации. Требование к реализации: из операций сравнения можно использовать только операцию "<".

Пришлите на проверку заголовочный файл, содержащий ТОЛЬКО определение шаблона. Он будет включен в cpp файл с тестами автоматически (main писать не нужно!).

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 2 2 100 1000 | 1000 |

# E\*. Приоритетная очередь. Шаблоны

|  |  |
| --- | --- |
| Ограничение времени | 0.5 секунд |
| Ограничение памяти | 64.8 Мб |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

В данной задаче вам необходимо реализовать шаблонные функции для работы с бинарной пирамидой (с максимумом на вершине):

PushHeap - восстанавливает пирамиду после добавления нового элемента (Insert);

PopHeap - восстанавливает пирамиду после извлечения максимального элемента из пирамиды.

Функция PushHeap должна принимать два аргумента: begin - указатель на начало последовательности, end - указатель на конец последовательности (элемент следующий за последним). Поведение, ожидающееся от функции: вставляет в пирамиду элемент, находящийся на позиции end-1, восстанавливает пирамиду на диапазоне [begin, end). Гарантируется, что перед выполнением функции PushHeap диапазон [begin, end - 1) является корректной пирамидой.

Функция PopHeap должна принимать два аргумента: begin - указатель на начало последовательности, end - указатель на конец последовательности (элемент следующий за последним). Поведение, ожидающееся от функции: меняет местами элемент на позиции begin с элементом на позиции end - 1 и восстанавливает пирамиду на диапазоне [begin, end-1). Гарантируется, что перед выполнением функции PopHeap диапазон [begin, end) является корректной пирамидой.

**Пример:**

int heap[7]{9, 5, 4, 1, 2, 3, 6}; (последний элемент не вставлен)\*

PushHeap(heap, heap + 7); // heap == {9, 5, 6, 1, 2, 3, 4} (вставили 6)

PopHeap(heap, heap + 7); // heap == {6, 5, 4, 1, 2, 3, 9} (извлекли 9)

Ваш код будет проверяться на скрытых от вас юнит-тестах, проверяющих корректность шаблонной реализации. Требование к реализации: из операций сравнения можно использовать только операцию "<".

Пришлите на проверку заголовочный файл, содержащий ТОЛЬКО определение шаблона. Он будет включен в cpp файл с тестами автоматически (main писать не нужно!).

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 4 7  1  2 9  2 4  2 9  2 9  2 7  1 | 9 |

# F. A+B

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 256Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Речь идёт о 10000-значных суммах...

## Формат ввода

Во входном файле два целых неотрицательных числа *A* и *B* *(0 ≤ A, B ≤ 1010000)* каждое в своей строке.

## Формат вывода

В выходной файл выведите одно число, равное сумме *A* и *B*.

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 3  5 | 8 |

# G. A-B

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 256Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

Лапидарно.

## Формат ввода

Во входном файле два целых неотрицательных числа *A* и *B* *(0 ≤ B ≤ A ≤ 1010000)* каждое в своей строке.

## Формат вывода

В выходной файл выведите одно число, равное разности *A* и *B*.

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 5  3 | 2 |

# H. A\*B

|  |  |
| --- | --- |
| Ограничение времени | 1 секунда |
| Ограничение памяти | 256Mb |
| Ввод | стандартный ввод или input.txt |
| Вывод | стандартный вывод или output.txt |

В этой задаче от вас требуется реализовать умножение (<https://en.wikipedia.org/wiki/Multiplication>).

## Формат ввода

Во входном файле два целых неотрицательных числа *A* и *B* *(0 ≤ A, B ≤ 1010000)* каждое в своей строке.

## Формат вывода

В выходной файл выведите одно число, равное произведению *A* и *B*.

## Пример

| **Ввод**  Скопировать ввод | **Вывод**  Скопировать вывод |
| --- | --- |
| 65536  216 | 14155776 |