Домашнее задание 5

В этом домашнем задании вам предлагается написать несколько алгоритмических функций, а также код, тестирующий эти функции. Реализации алгоритмов должны располагаться в файле исходного кода (**vector_algorithms.cpp**), объявления - в заголовочном файле (**vector_algorithms.h)**. В файле **main.cpp** разместите код, проверяющий работоспособность реализованных функций.

1. **Генератор массива случайных чисел.** Функция, возвращающая вектор случайных целых чисел (числа из диапазона [min, max]). Входные параметры: количество чисел в векторе, минимально возможное число, максимально возможное число.

   ```
   std::vector<int> GenerateRandomVector(size_t size, int min, int max);
   ```

   Таким образом, при вызове GenerateRandomVector(5, 1, 4) может быть возвращен вектор [1, 2, 4, 4, 1]. Это вектор из 5 чисел, каждое из которых находится на отрезке [1, 4].

   В этой функции используйте значения по умолчанию для параметров **min** и **max**. Дефолтное поведение для такой функции - генерация любых допустимых целых чисел. Минимальное и максимальное целые числа определены в <climits>.

   Для генерации случайного числа воспользуйтесь функцией rand(). Её описание можно найти здесь**.** По этой же ссылке вы можете найти пример использования и способ генерации чисел в заданном интервале.

   Напоминаю, что добавить элемент в вектор можно следующим образом:
   ```
   std::vector<int> myVector; // создаём пустой вектор целых чисел
   myVector.push_back(5); // добавляем в вектор число 5
   ```

2. **Линейный поиск**. На вход подается вектор целых чисел и целое число, наличие которого в векторе необходимо проверить. Если это число присутствует в массиве, то необходимо вернуть индекс первого найденного элемента, если элемента в массиве нет, то верните -1.

   ```
   int LinearSearch(const std::vector<int>& numbers, int value);
   ```

3. **Алгоритм сортировки**. Реализуйте любой алгоритм сортировки вектора целых чисел. Функция должна сортировать вектор целых чисел и принимать на вход два параметра: вектор (который хотим отсортировать) и булев аргумент

reverse. Если значение аргумента **False** (сделайте его значением по умолчанию), то сортировка должна быть выполнена по возрастанию, если **True** - по убыванию. Алгоритмы можно посмотреть в интернете, например, https://en.wikipedia.org/wiki/Bubble_sort
https://en.wikipedia.org/wiki/Insertion_sort
http://www.youtube.com/watch?v=lyZQPjUT5B4&list=PL5KGGnTtdXf4ypLQuQFwPGbIFECDByNB7

```
void Sort(std::vector<int>& numbers, bool reverse);
```

4. **Алгоритм проверки сортированности массива.** На вход подаются (как и в прошлой задаче) вектор чисел и направленность сортировки. Функция возвращает True, если массив отсортирован в нужном направлении и False в противном случае.

```
bool IsSorted(const std::vector<int>& numbers, bool reverse);
```

5. **Алгоритм бинарного поиска**. Вам дан отсортированный по возрастанию массив целых чисел и число. Как и в задаче 2, вам нужно вернуть индекс минимального элемента вектора, равного заданному числу (или -1, если такого числа не нашлось), но теперь сделайте это более эффективно. Описание алгоритма можно найти, например, здесь:
https://en.wikipedia.org/wiki/Binary_search_algorithm

```
int BinarySearch(const std::vector<int>& numbers, int value);
```

Вы можете смотреть описания алгоритмов в интернете, но не код. Обратите внимание на так называемые “крайние случаи”: пустые массивы; массивы, заполненные одним и тем же элементом и другие. Они должны быть корректно обработаны.

После реализации функций проведите следующие эксперименты (код разместите в **main.cpp**). Функции возвращают True, если все тесты пройдены:

1. Проверьте, что все элементы созданного случайного вектора лежат в заданных границах. Функция генерирует **nTests** раз случайный вектор небольшого размера со случайными параметрами **min** и **max**, и проверяет, что все числа, лежащие в этом векторе находятся в интервале [**min, max**]

```
bool RandomValuesInRangeTest(size_t nTests);
```

2. Проверьте, что алгоритм сортировки работает корректно. Напишите функцию, которая создает **nTests** раз случайный вектор и сортирует по возрастанию/по убыванию, затем по убыванию. Проверьте, что в отсортированном векторе каждое следующее число не меньше/не больше предыдущего с помощью функции **IsSorted**.

```
bool SortTest(size_t nTests);
```

3. Проверьте, что алгоритмы **LinearSearch** и **BinarySearch** работают корректно. Для этого напишите функцию, которая сначала случайный вектор в заданном интервале. Проверьте с помощью функций, что в нем отсутствуют числа вне этого интервала и присутствует какие-то числа из интервала. Проведите эксперимент **nTests** раз.

```
bool SearchTest(size_t nTests);
```