#1: Quicksort

При реализации алгоритмов в виде библиотечных функций одним из требований к реализации является универсальность, чтобы однажды реализованный алгоритм можно было применять при необходимости к разным данным. Универсальность достигается через параметры, с их помощью можно настроить однажды реализованный алгоритм под конкретную задачу. В случае с алгоритмами сортировки такой настройкой является указание сортируемого массива и правила упорядочивания его элементов. Однако, в языках со статической типизацией (к каким относится и C++) возникает проблема, что при записи параметров функции требуется указать также их тип, в случае же универсального алгоритма хотелось бы иметь реализацию, не зависящую от типа сортируемых данных, и применять одну и ту же реализацию ко всем типам данных. Поэтому, большинство библиотек реализует алгоритмы в обобщенном (generic) виде. Механизмом обобщенного программирования в C++ являются шаблоны (templates). Общая идея заключается в том, что вместо использования конкретных типов данных при описании функции используются шаблонные параметры, вводимые при помощи специального синтаксиса. Например, функцию сортировки для массива целых чисел можно было объявить как

```
void sort(int *first, int *last)
```

здесь `int` – имя типа для целых чисел. При реализации алгоритма в обобщенном виде это имя можно заменить на параметр шаблона при помощи синтаксиса

```
template<typename T>
void sort(T *first, T *last)
```

Затем написать реализацию функции `sort()` как обычно, используя шаблонный параметр T везде, где необходимо имя типа элемента сортируемого массива (в первой реализации мы бы писали в этих случаях `int`). При вызове такой функции компилятор самостоятельно заменит шаблонный параметр на реально используемый тип данных и сгенерирует правильный код для его обработки.

Второй параметр, который мы бы хотели передавать – это правило упорядочивания элементов. На языке C++ такое правило будет записано в виде функции. Затем эту функцию необходимо будет передать в функцию сортировки. Для передачи функций в качестве параметров C++ поддерживает специальный тип данных – указатели на функции (function pointers) и функциональные объекты

(functional objects). Начиная с C++11 для этого также можно применять лямбды (lambda).

В вашу задачу входит разработка обобщенной функции `sort()`, реализующей алгоритм быстрой сортировки со следующими оптимизациями:

- Выбор опорного элемента, как медианы из первого, среднего и последнего элемента сортируемого интервала;
- Исключение хвостовой рекурсии: функция должна определять длины получившихся интервалов после разбиения и рекурсивно сортировать интервал меньшей длины, итеративно обрабатывая интервал большей длины;
- Использование алгоритма сортировки вставками для коротких интервалов (длину такого интервала необходимо подобрать экспериментально).

Функция должна принимать три аргумента: начало `first` и конец `last` сортируемого интервала и предикат сортировки `comp`, который принимает два аргумента – элементы массива `a` и `b`, и возвращает `true`, если `a < b`, то есть в отсортированном массиве элемент `a` должен располагаться *перед* `b`.

```
template<typename T, typename Compare>
void sort(T *first, T *last, Compare comp)
```

где предикат упорядочивания должен удовлетворять прототипу

```
template<typename T>
bool comp(const T &a, const T &b)
```

Сортировка массива из 100 целых чисел при помощи разработанной функции будет выглядеть следующим образом:

```
int a[100];
sort(a, a + 100, [](int a, int b) { return a < b; });
```

Для разработанной функции необходимо создать набор модульных тестов (с использованием любого тестового фреймворка, например googletest (https://github.com/google/googletest). Тесты должны быть репрезентативными, покрывать общий и максимальное количество граничных случаев. К каждому тесту сделайте небольшой комментарий, в котором опишите, почему выбрали именно такой набор данных.