Задание A2

Этап 1.

Для генерации тестовых данных был создан класс ArrayGnerator, в котором сначала создаются 3 массива массивов: с рандомными числами от 1 до 6000, с отсортированными массивами в том же диапазоне и с почти отсортированными. Для этого в каждой группе массивов сначала создается массив с 10000 случайными элементами. Для второй и третьей группы массивы сортируются по убыванию. А для третьей дополнительно меняются несколько случайных элементов. Остальные массивы в тестовых группах являются подмассивами нужной длины исходного сгенерированного массива из 10000 элементов.

```
class ArrayGenerator {
private:
  std::vector<std::vector<int>> randomArrays;
  std::vector<std::vector<int>> sortedArrays;
  std::vector<std::vector<int>> almostSortedArrays;
public:
  ArrayGenerator() {
    std::random_device rand_dev;
    std::mt19937 generator( sd: rand_dev());
    std::uniform_int_distribution<> distr( a: 1,  b: 6000);
    randomArrays.emplace_back( n: 10000);
    sortedArrays.emplace_back( n: 10000);
    almostSortedArrays.emplace_back( n: 10000);
    for (int i = 0; i < 10000; ++i) {
      randomArrays[0][i] = distr( &: generator);
      sortedArrays[0][i] = distr( &: generator);
      almostSortedArrays[0][i] = distr( &: generator);

    }
    std::stable_sort( first: sortedArrays[0].begin(),  last: sortedArrays[0].end());
    for (int i = 0; i < sortedArrays[0].size() / 2; ++i) {
      std::swap( &: sortedArrays[0][i],  &: sortedArrays[0][10000 - i - 1]);
    }
    int n = almostSortedArrays[0].size();

    std::stable_sort( first: almostSortedArrays[0].begin(),  last: almostSortedArrays[0].end());
    for (int i = 0; i < n / 2; ++i) {
      std::swap( &: almostSortedArrays[0][i],  &: almostSortedArrays[0][10000 - i - 1]);
    }

```

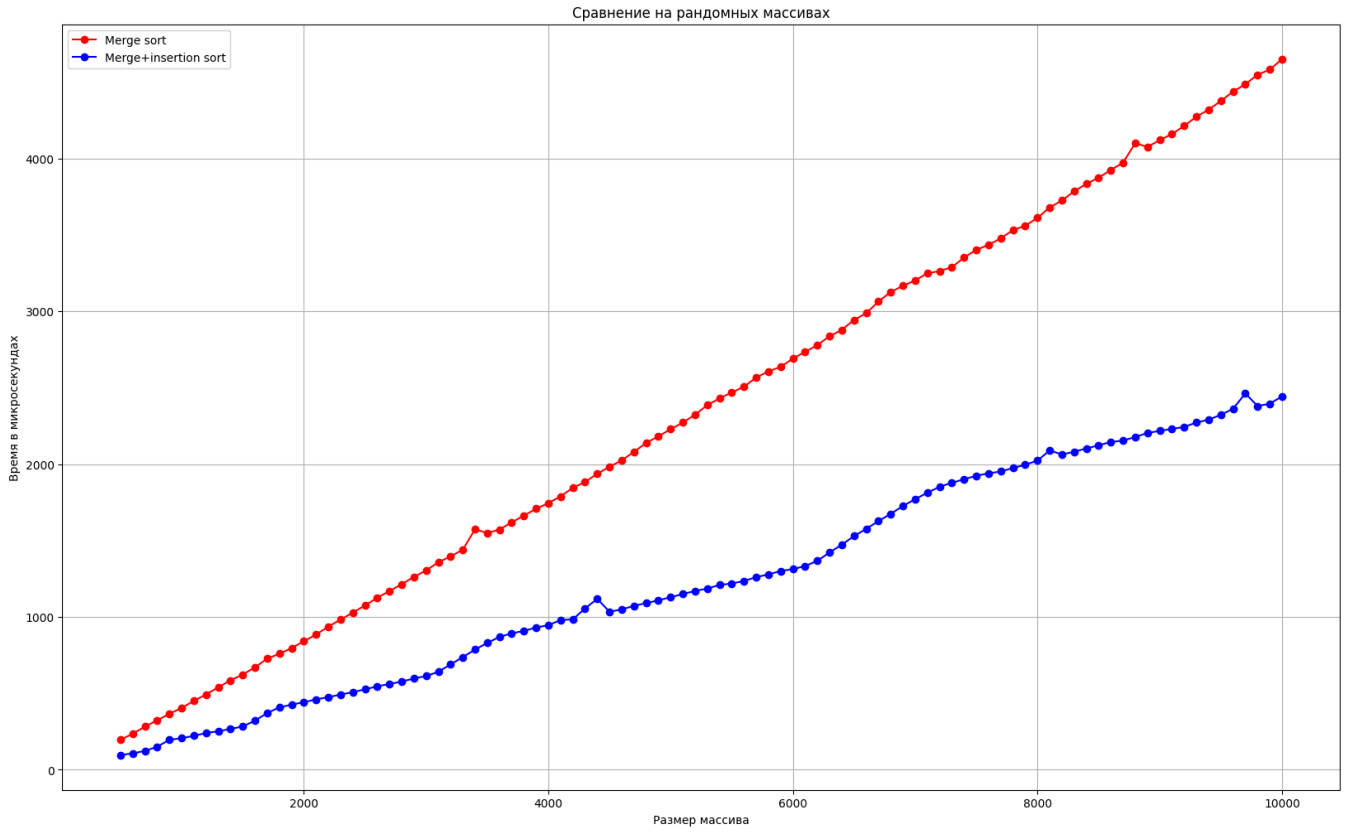
```
    for (int i = 0; i < 10; ++i) {
      std::swap( &: almostSortedArrays[0][distr( &: generator) % n], &: almostSortedArrays[0][distr( &: generator) % n]
    }
    int k = 1;
    for (int i = 500; i < 10000; i += 100) {
      randomArrays.emplace_back( n: i);
      sortedArrays.emplace_back( n: i);
      almostSortedArrays.emplace_back( n: i);
      int start = distr( &: generator) % (10000 - i);
      for (int j = 0; j < i; ++j) {
        randomArrays[k][j] = randomArrays[0][start + j];
        sortedArrays[k][j] = sortedArrays[0][start + j];
        almostSortedArrays[k][j] = almostSortedArrays[0][start + j];
      }
      k += 1;
    }
  }
  std::vector<std::vector<int>> getRandomArrays() {
    return randomArrays;
  }
  std::vector<std::vector<int>> getSortedArrays() {
    return sortedArrays;
  }
  std::vector<std::vector<int>> getAlmostSorted() {
    return almostSortedArrays;
  }
};
```

Этап 2-3.
Для тестирования был создан класс SortTester, в котором реализован замер времени сортировки mergeSort. В основной программе вызывается метод Test этого класса для трех групп массивов. Сначала тестируется обычный mergeSort, а потом для merge + insertion sort. Результаты выполнения записываются в файл и потом используются для построения графиков.

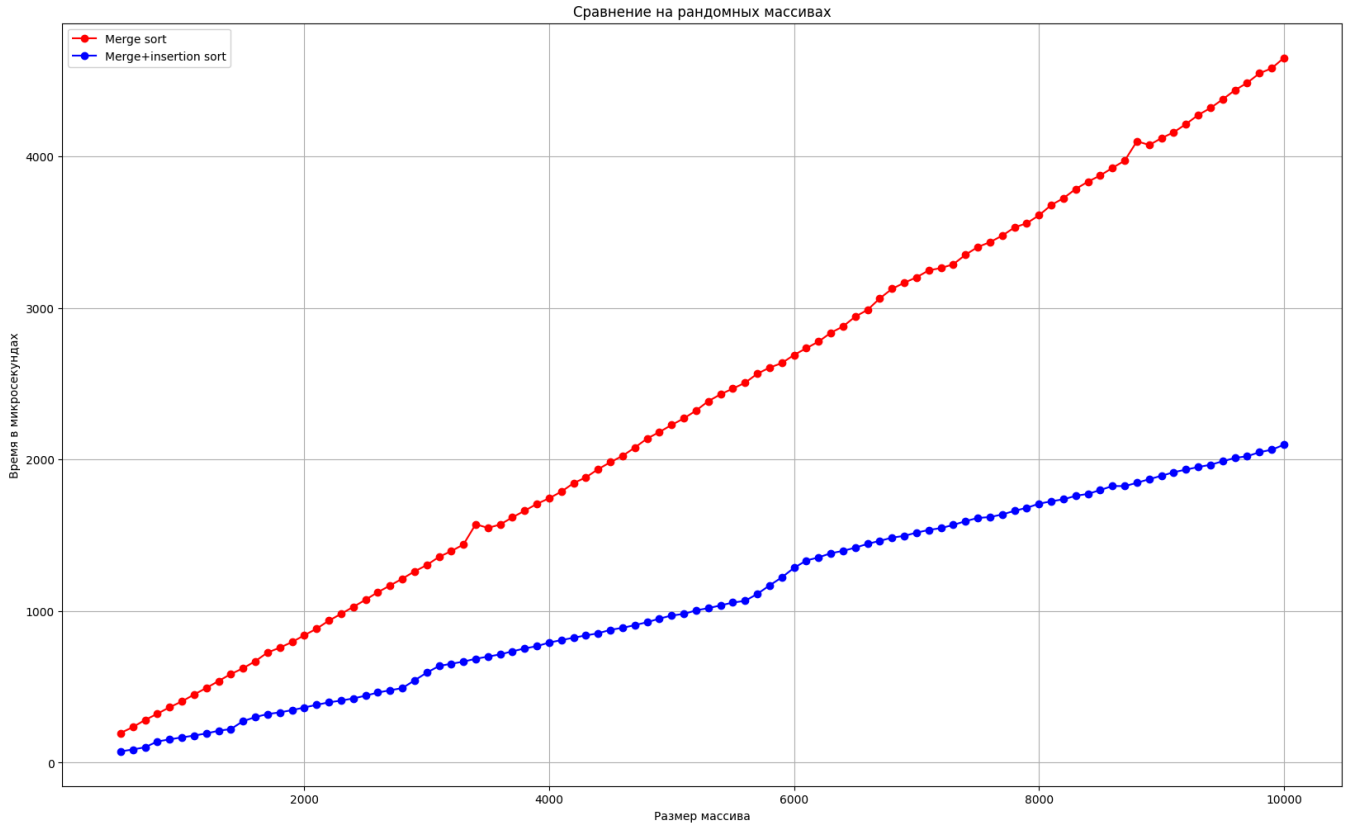
```
class SortTester {
public:
  long long Test(std::vector<int> A) {
    auto start : time_point<...>  = std::chrono::high_resolution_clock::now();
    mergeSort( &: A,  left: 0,  right: static_cast<int>(A.size() - 1));
    auto elapsed : duration<...>  = std::chrono::high_resolution_clock::now() - start;
    long long msec = std::chrono::duration_cast<std::chrono::microseconds>( d: elapsed).count();
    return msec;
  }
};
```

Этап 4.
Данные представлены в виде графиков, где по оси X - размер массива, по оси Y - время работы алгоритма в микросекундах, синим цветом изображены данные для гибридной сортировки, красным - для стандартного merge sort.

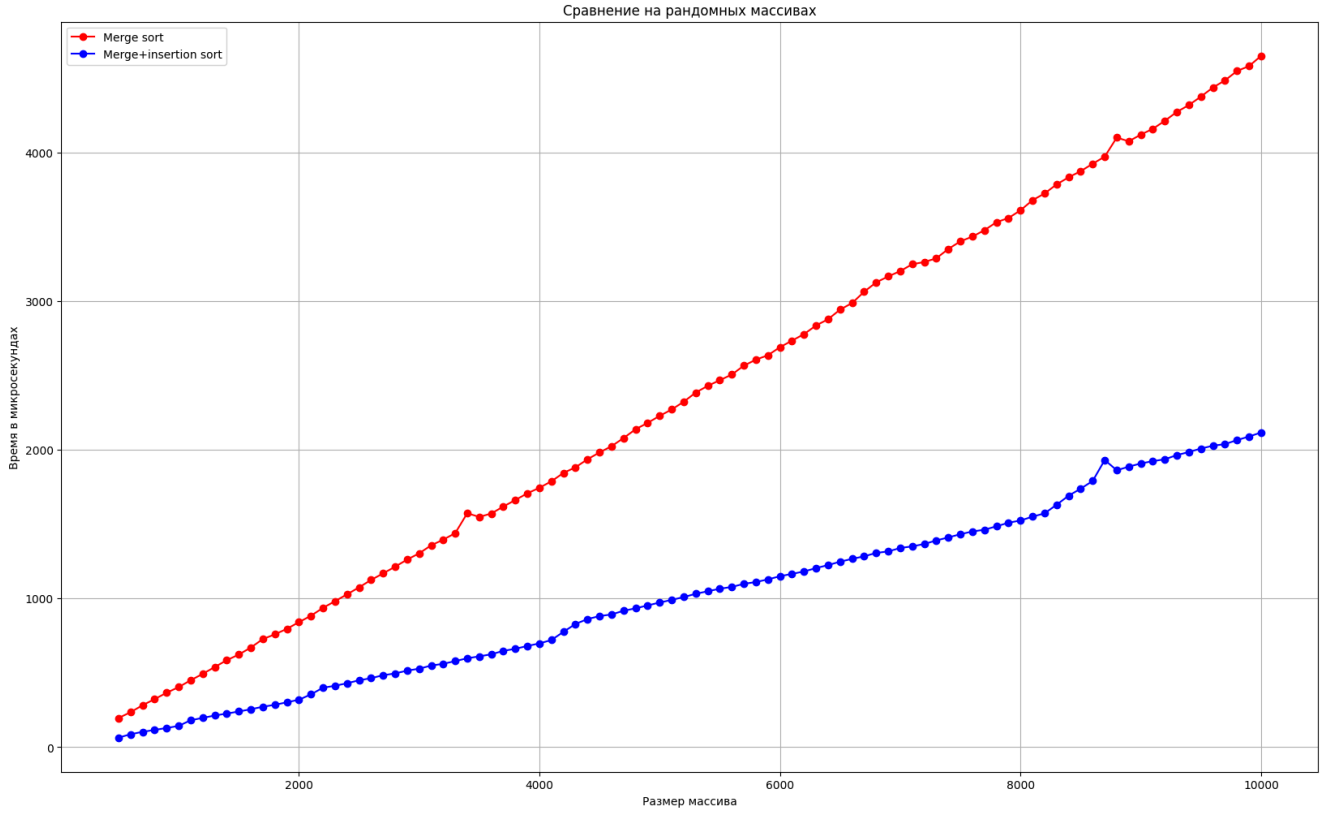
Для усреднения значений для каждого массива измерения были проведены 10 раз, чтобы взять среднее значение.
Сравнение работы алгоритмов на массивах с рандомными числами.
threshold = 5



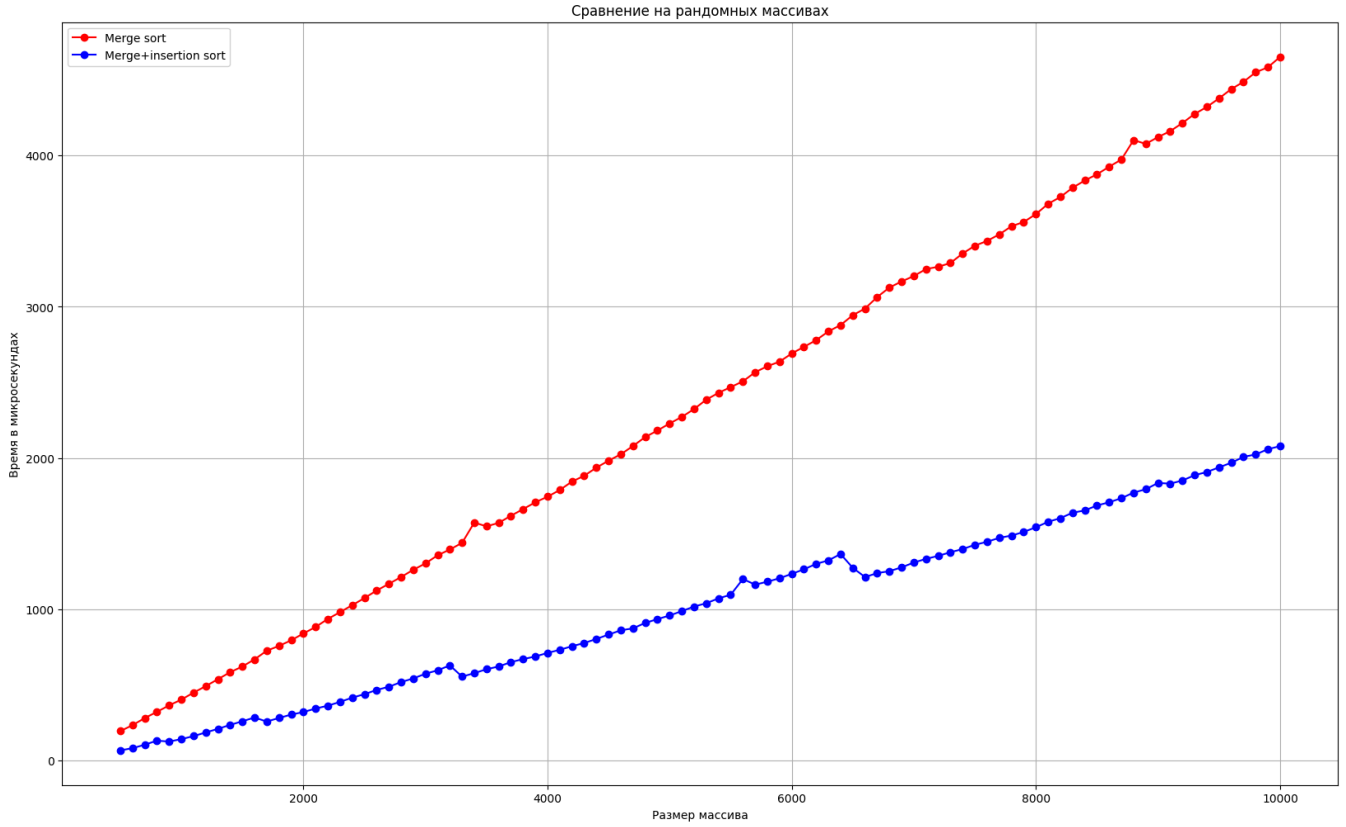
threshold = 10



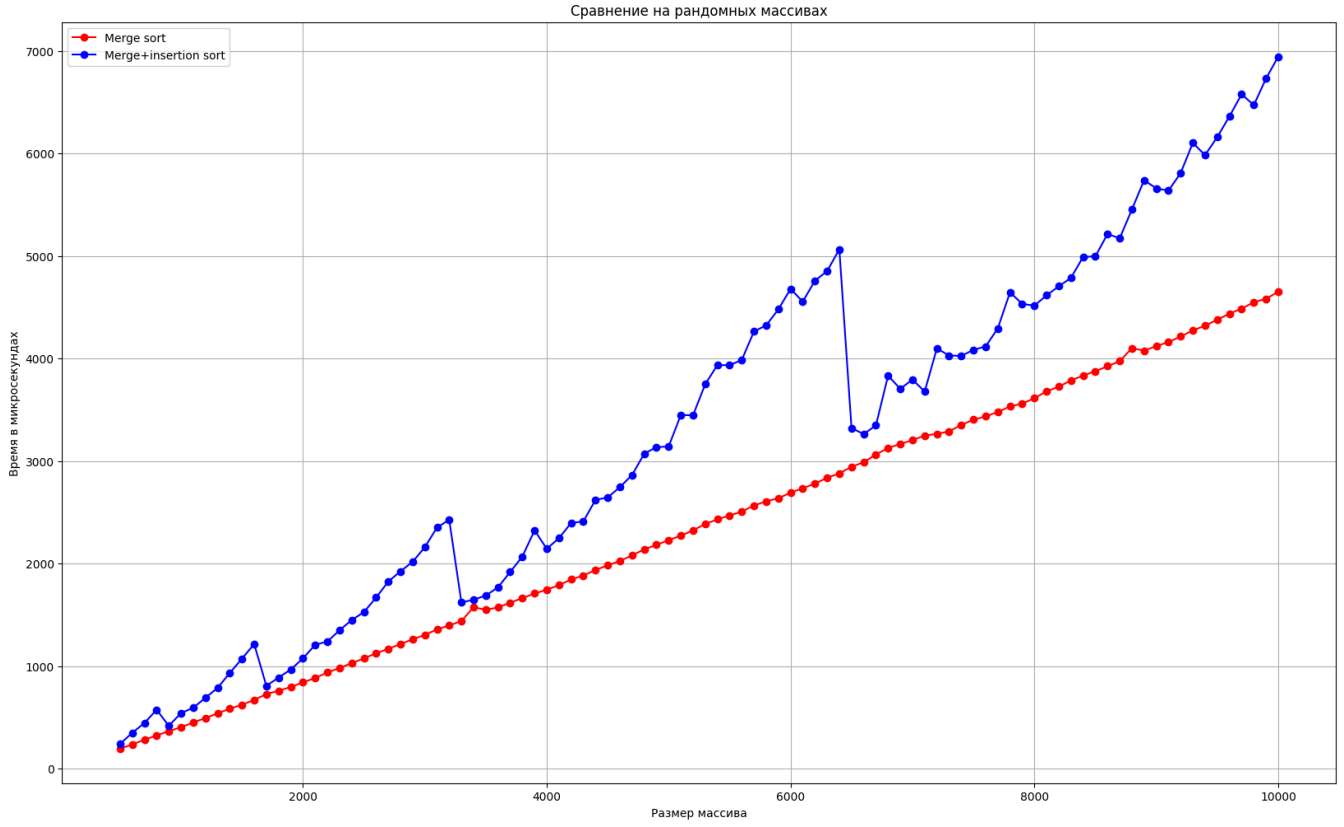
threshold = 15



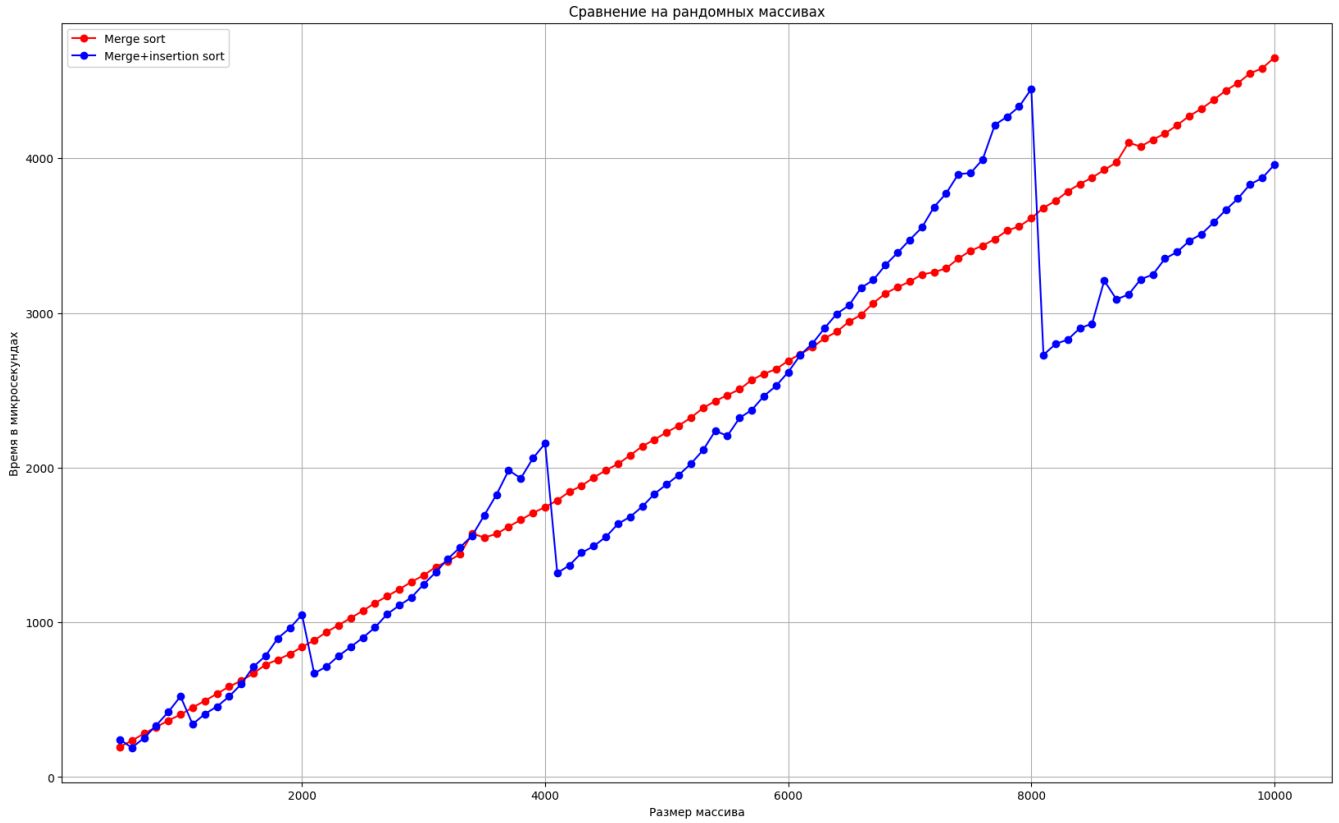
threshold = 100



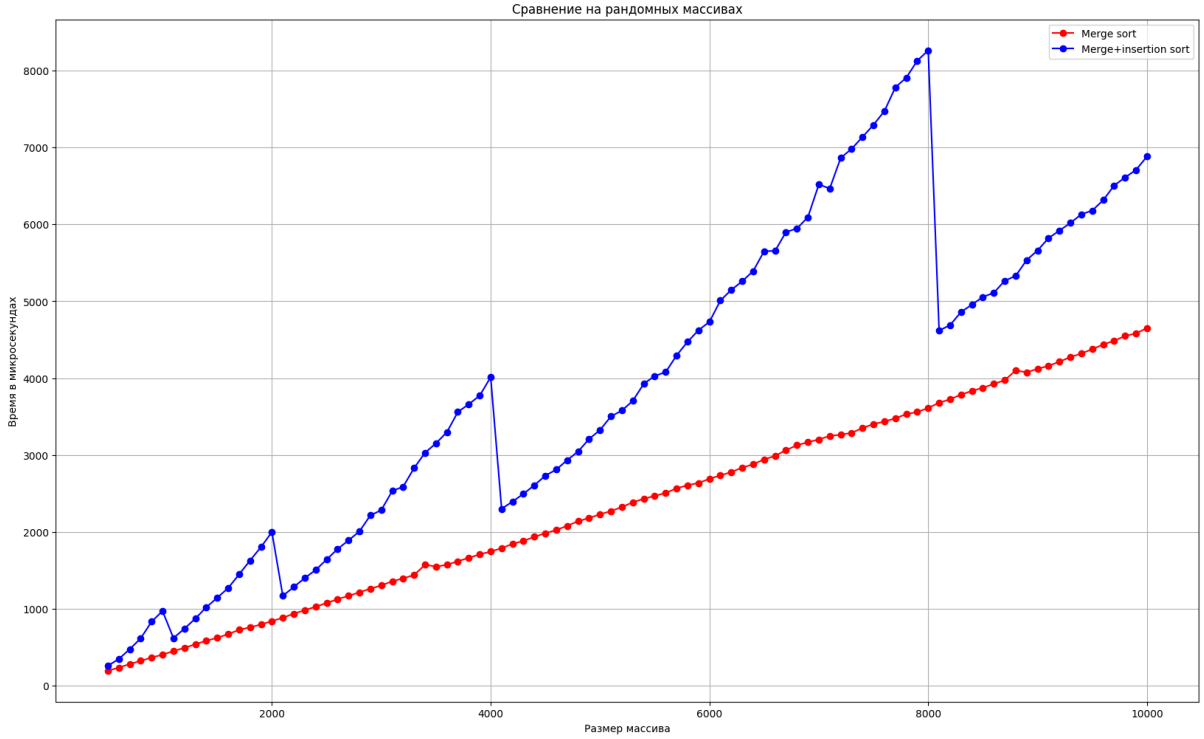
threshold = 400



threshold = 500



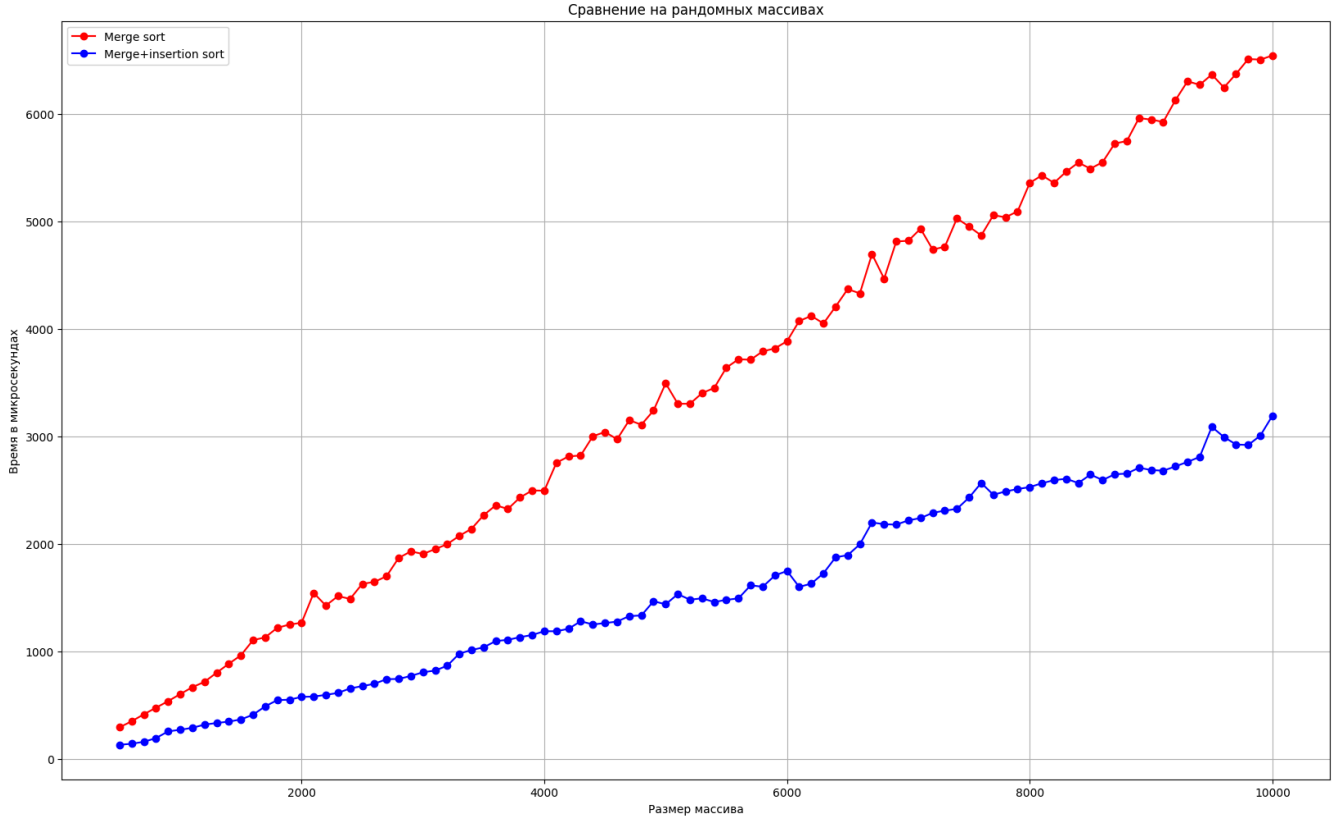
threshold = 1000



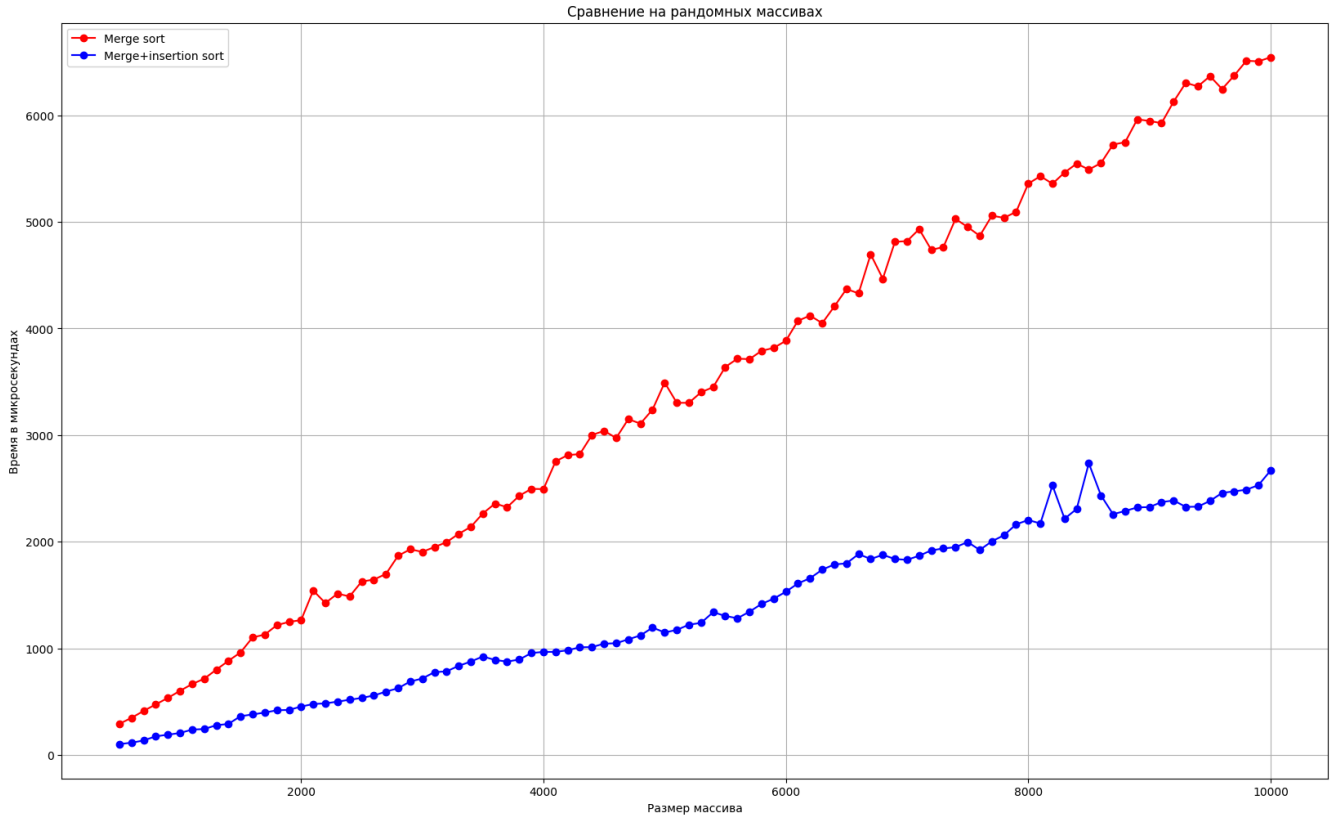
Можно заметить, что до threshold = 400 алгоритм mergeSort работает значительно медленнее, чем merge+insertion sort. На размерах матрицы до 1000 значения для двух сортировок находятся близко друг другу, однако

потом разница во времени становится существеннее с преимуществом по времени у гибридной версии алгоритма. При threshold = 400 merge sort становится быстрее, при threshold = 500 показатели времени колеблются, однако в большинстве случаев гибридная сортировка остается быстрее. И при threshold = 1000 можно увидеть преимущество обычной сортировки из-за того, что константа в гибридной сортировке перекрывается квадратом в insertion sort, который замедляет работу.

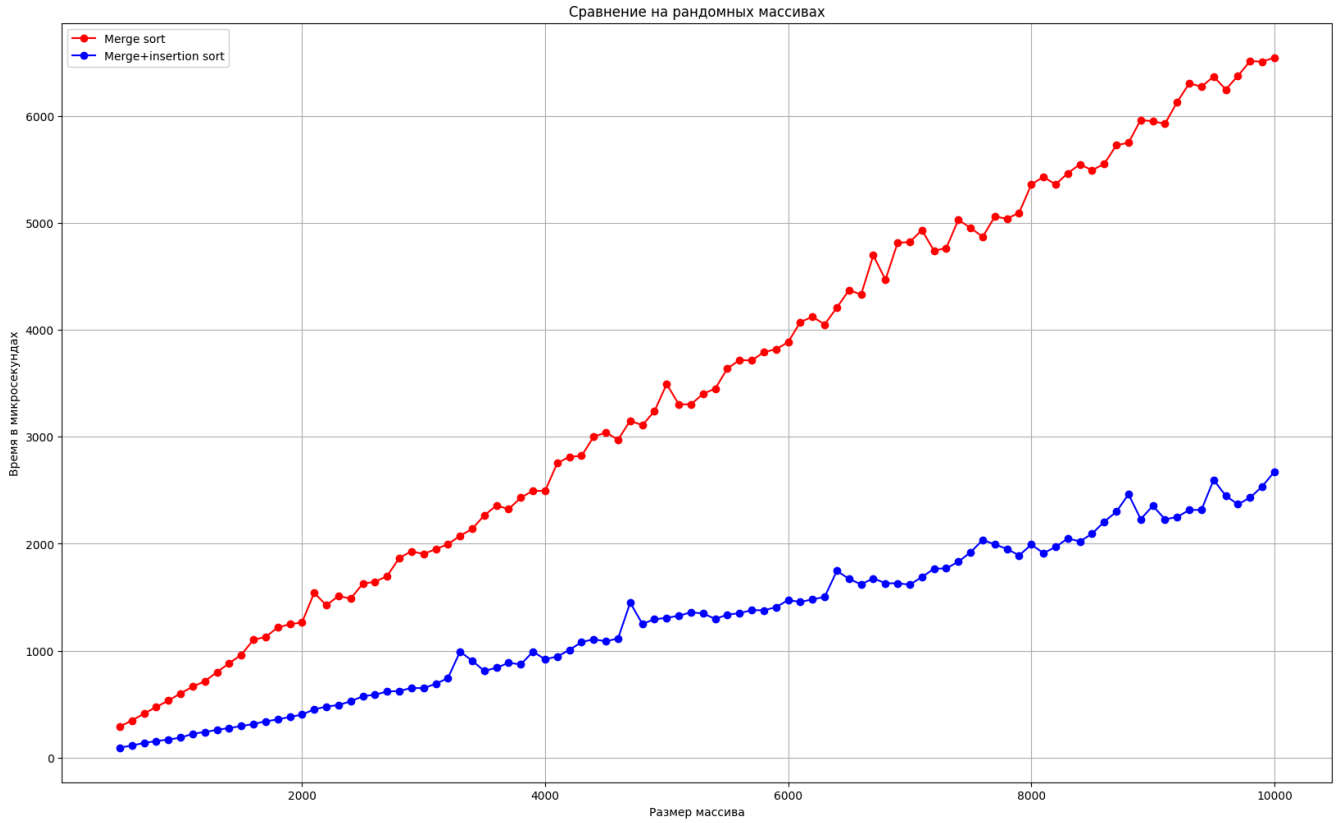
На отсортированных массивах
threshold = 5



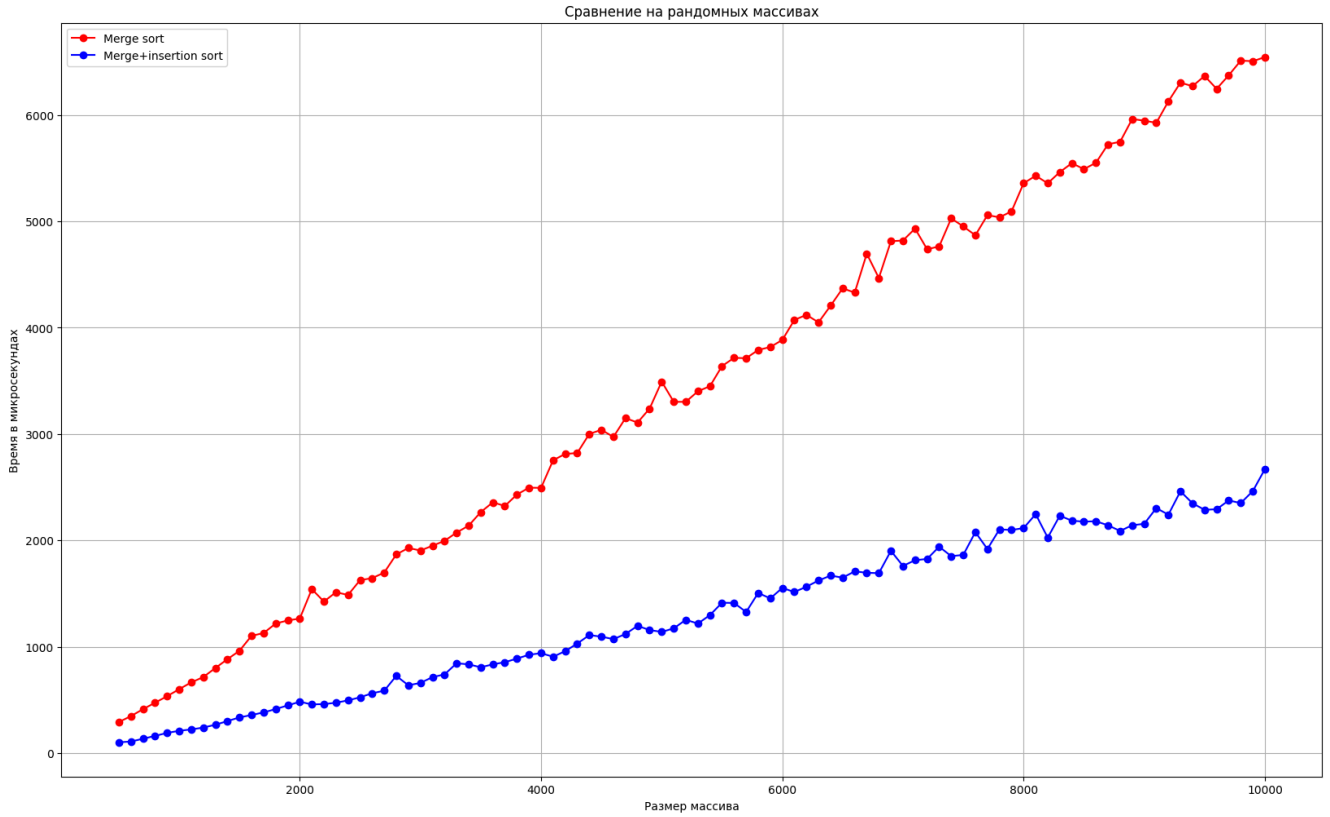
threshold = 10



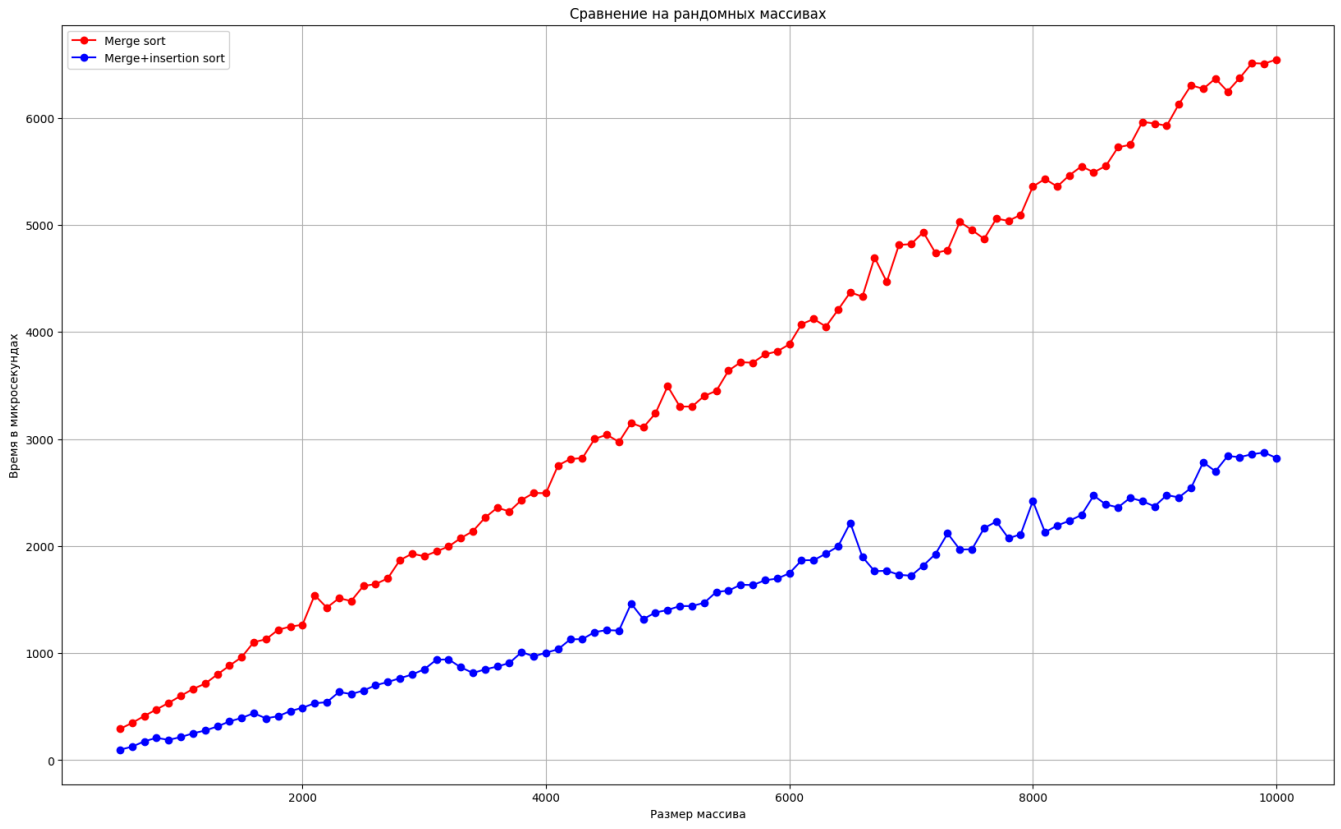
threshold = 15



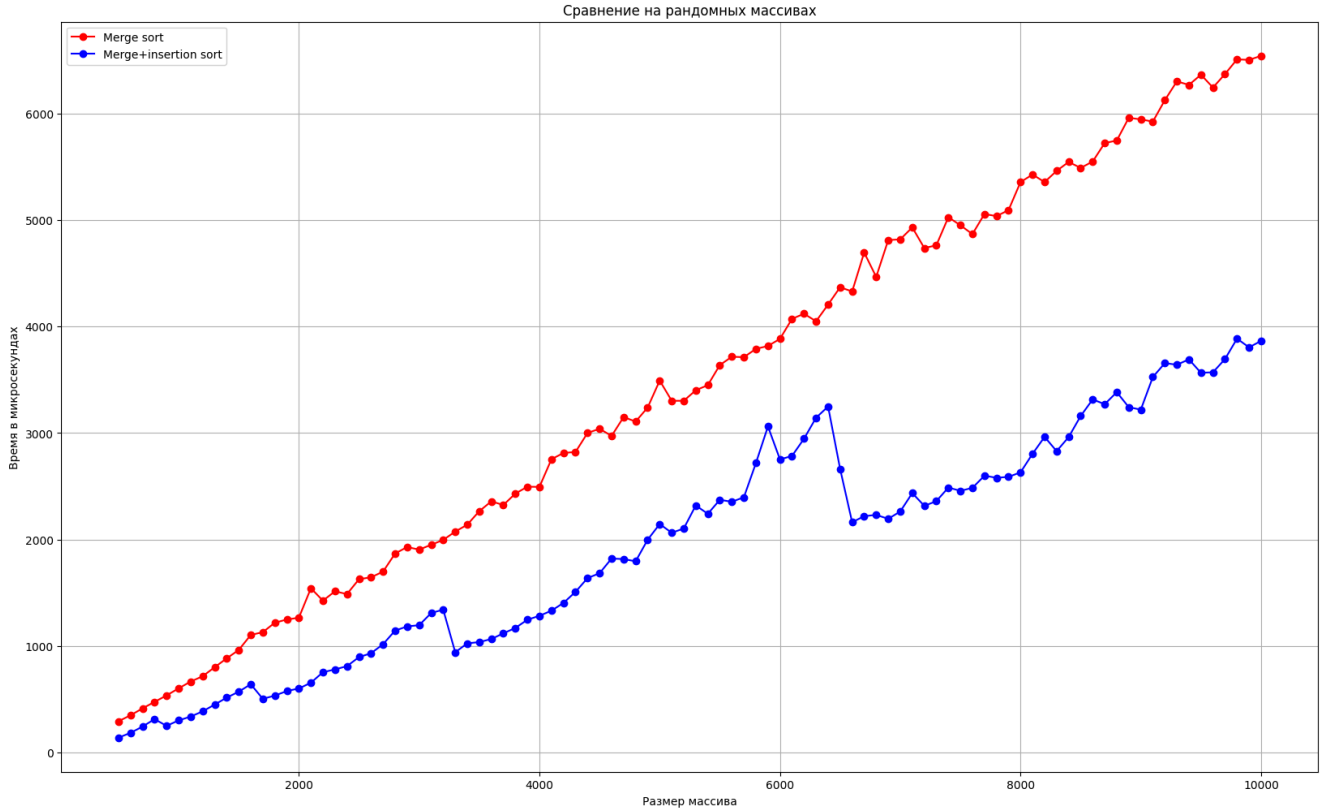
threshold = 30



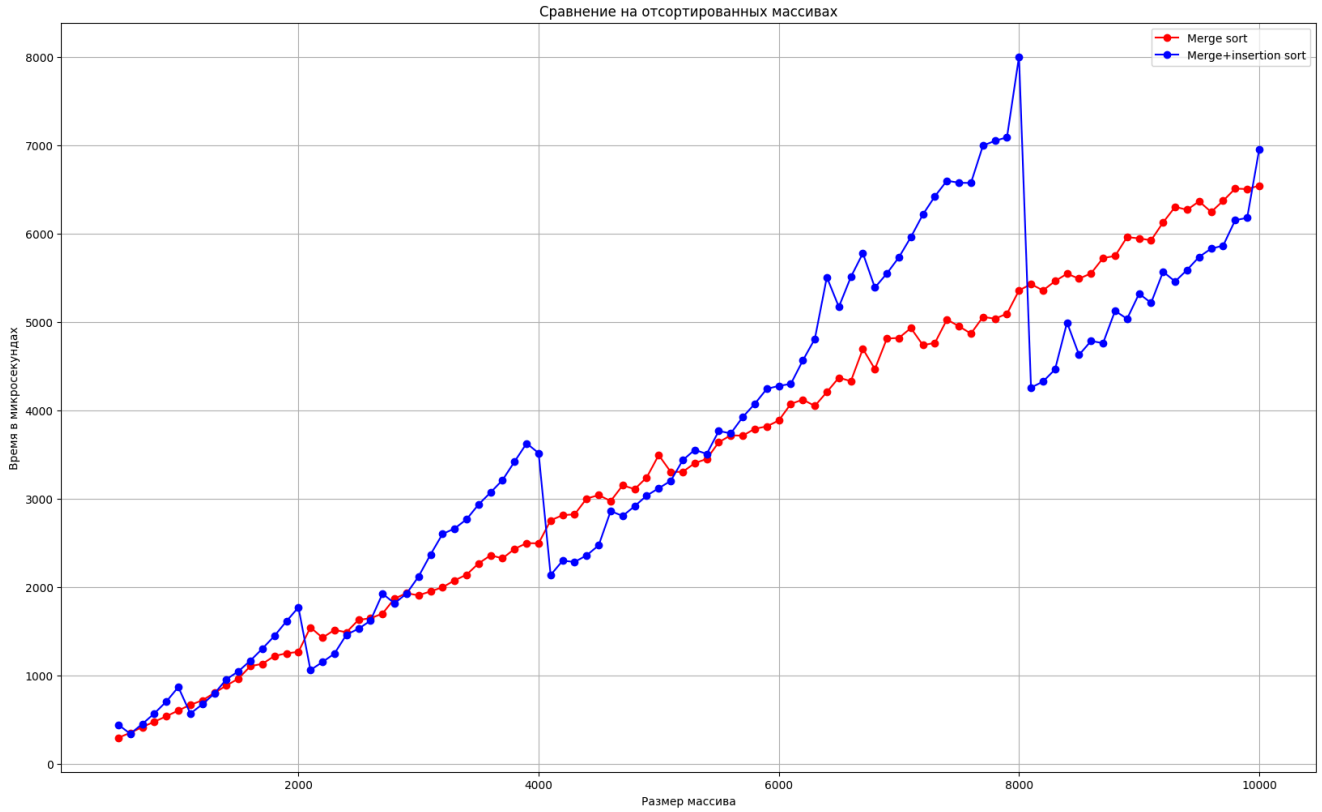
threshold = 50



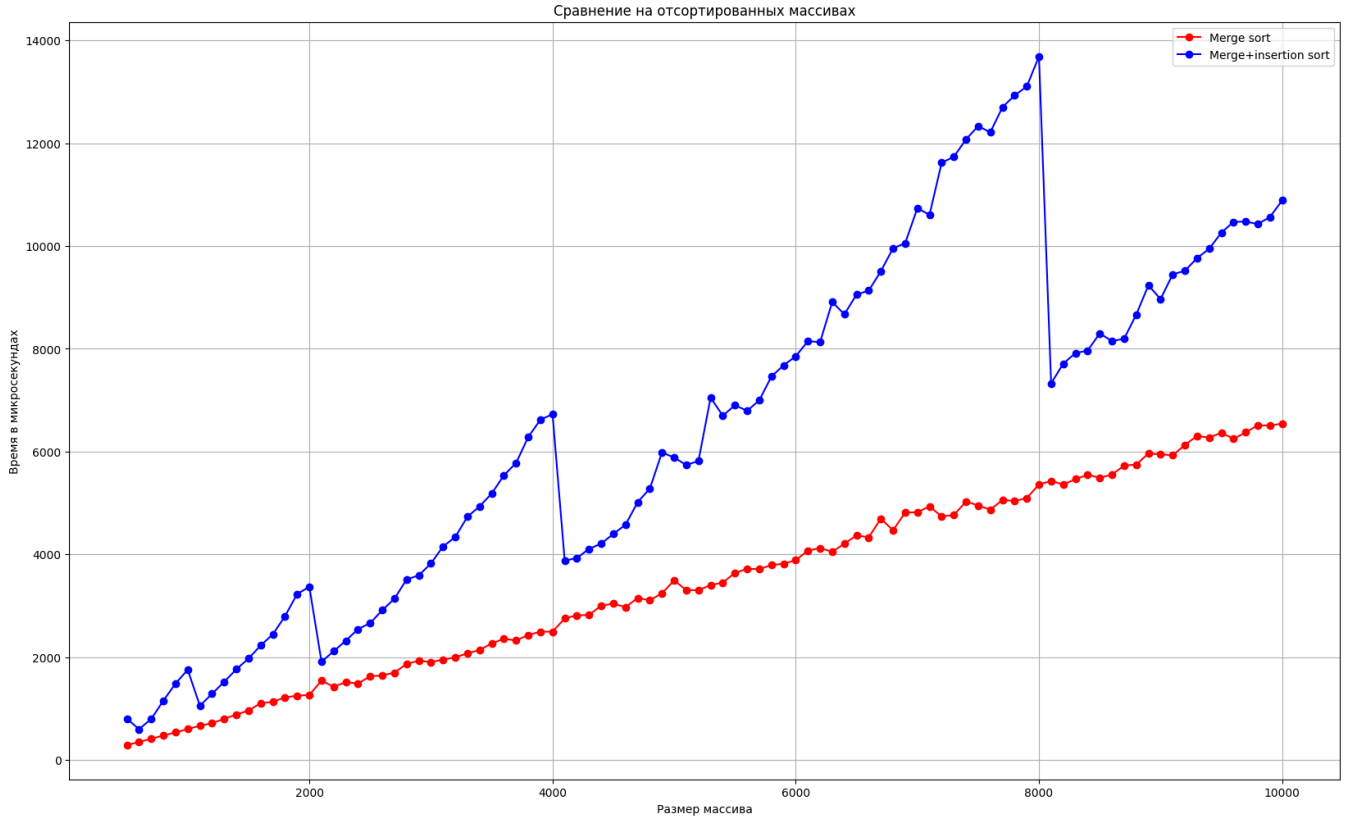
threshold = 100



threshold = 250



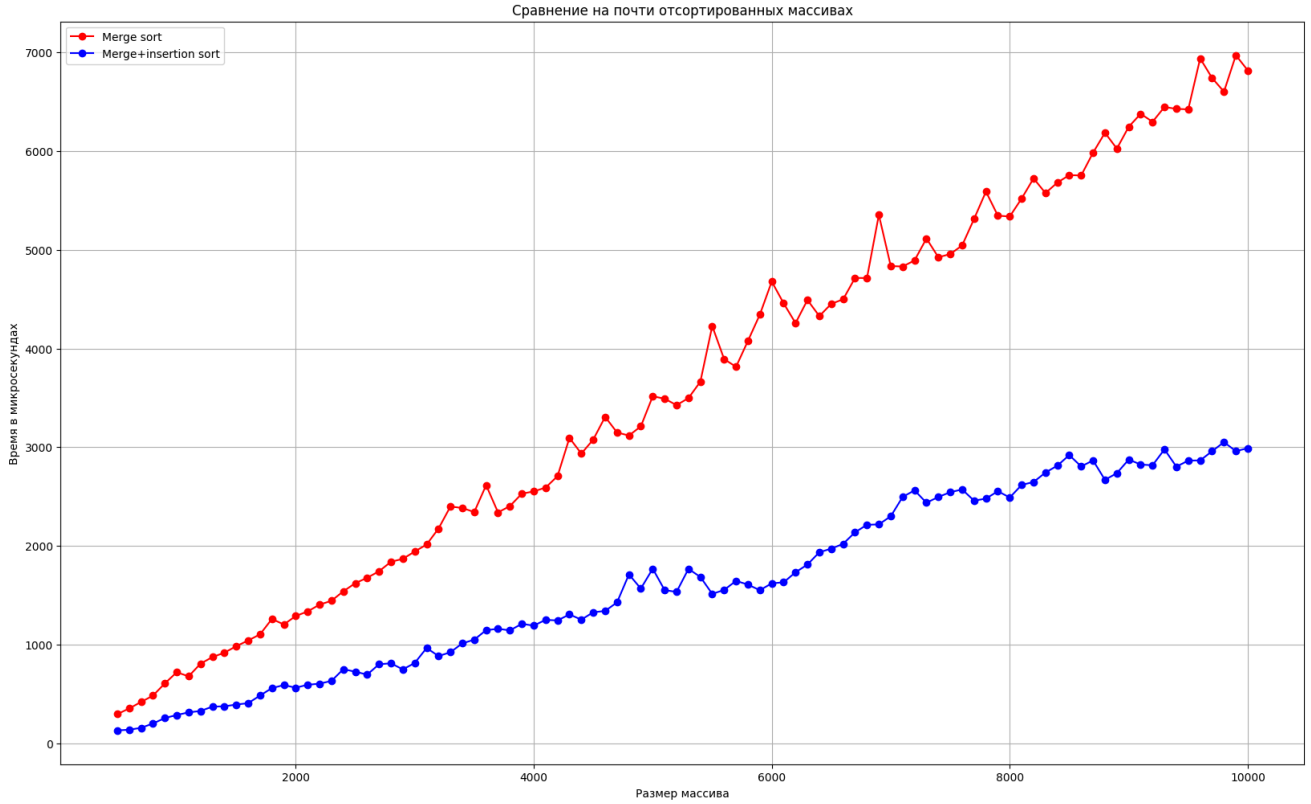
threshold = 500



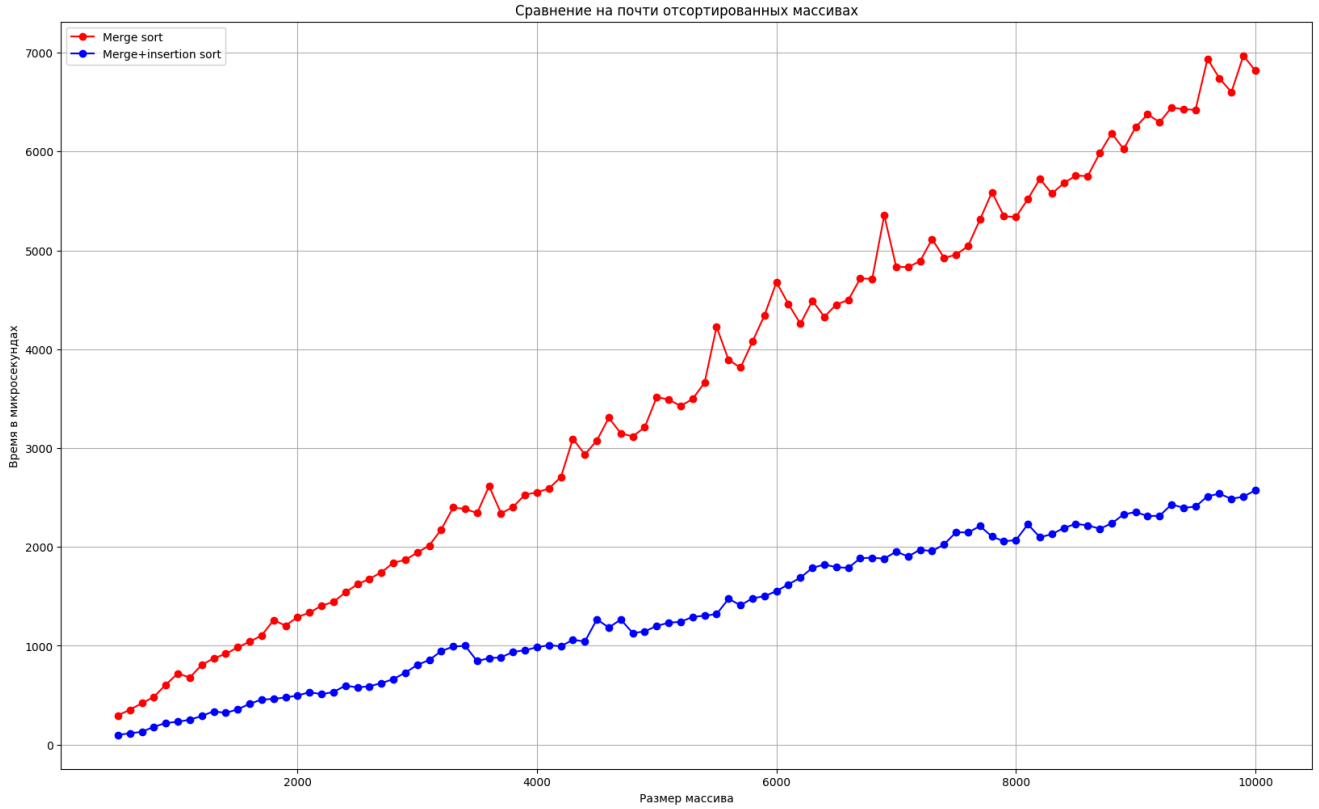
При threshold до 100 распределение данных выглядит примерно одинаково, гибридная сортировка сильно быстрее, чем стандартная, при threshold = 100 с размерами массива до 6400 временные показатели гибридной сортировки приближаются к показателям merge sort, однако потом снова становятся намного быстрее. При threshold = 250 уже заметно, что гибридная сортировка проигрывает по времени, лишь иногда становясь более эффективной. При threshold = 500 на всех массивах гибридная сортировка работает хуже. Сравнивая с неотсортированными данными, и гибридная, и в стандартная сортировка работают хуже и порог, когда merge sort начинает работать быстрее, чем merge+insertion sort ниже.

Почти отсортированные данные

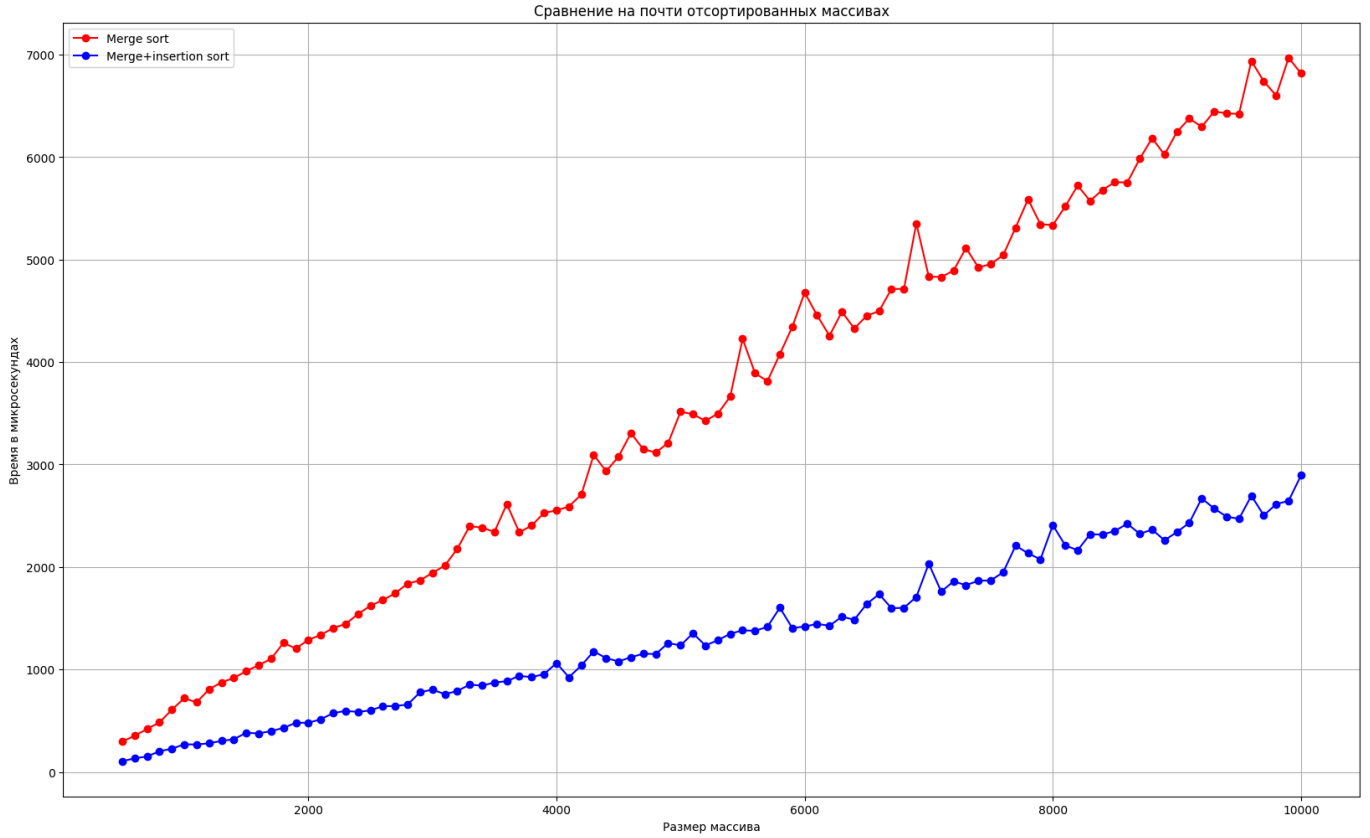
threshold = 5



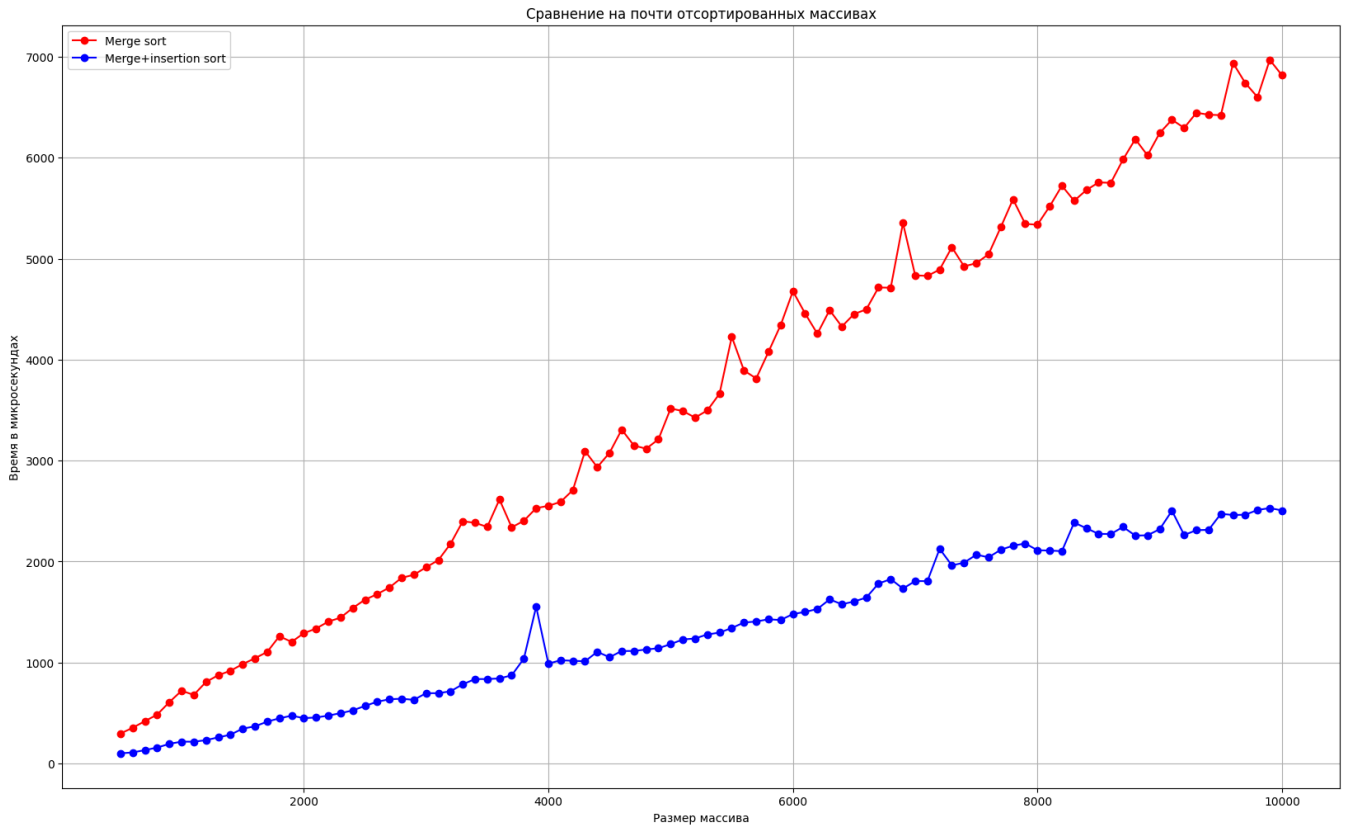
threshold = 10



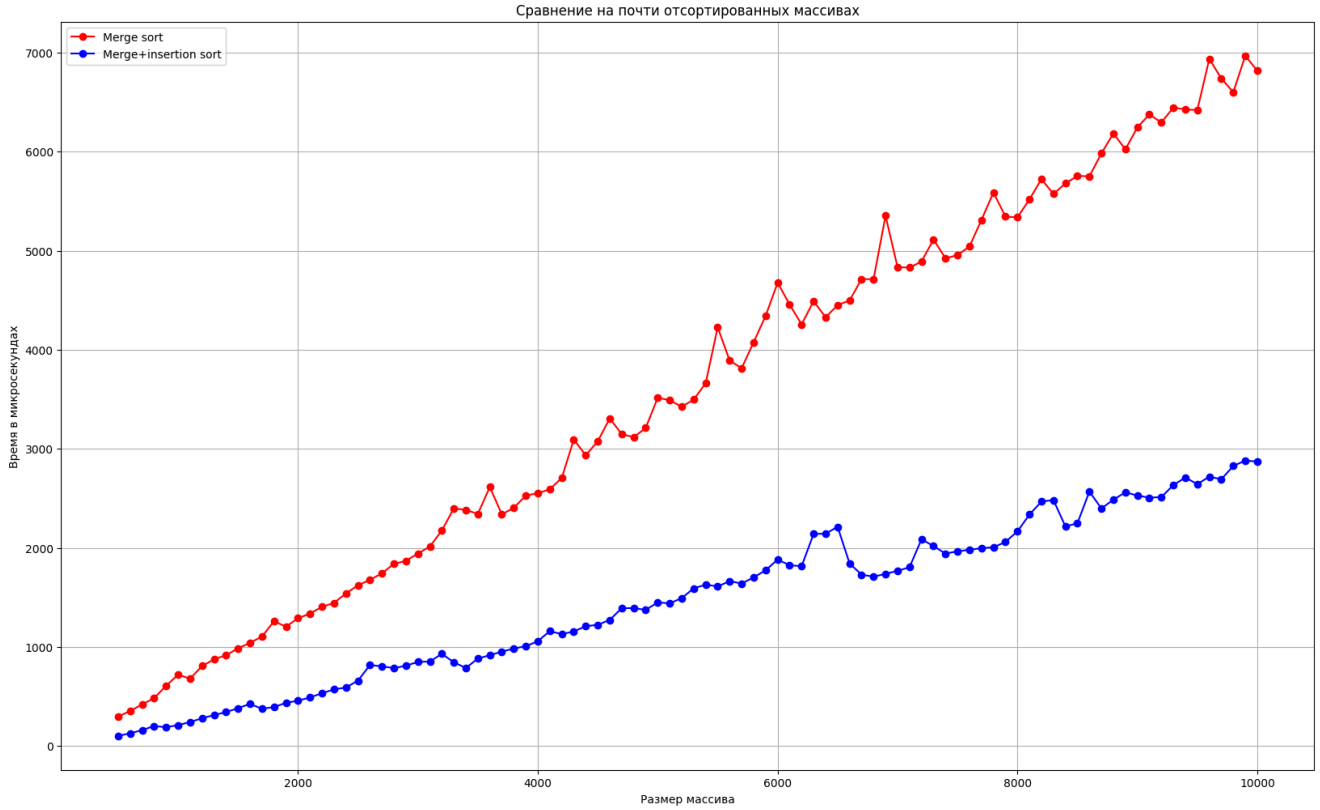
threshold = 15



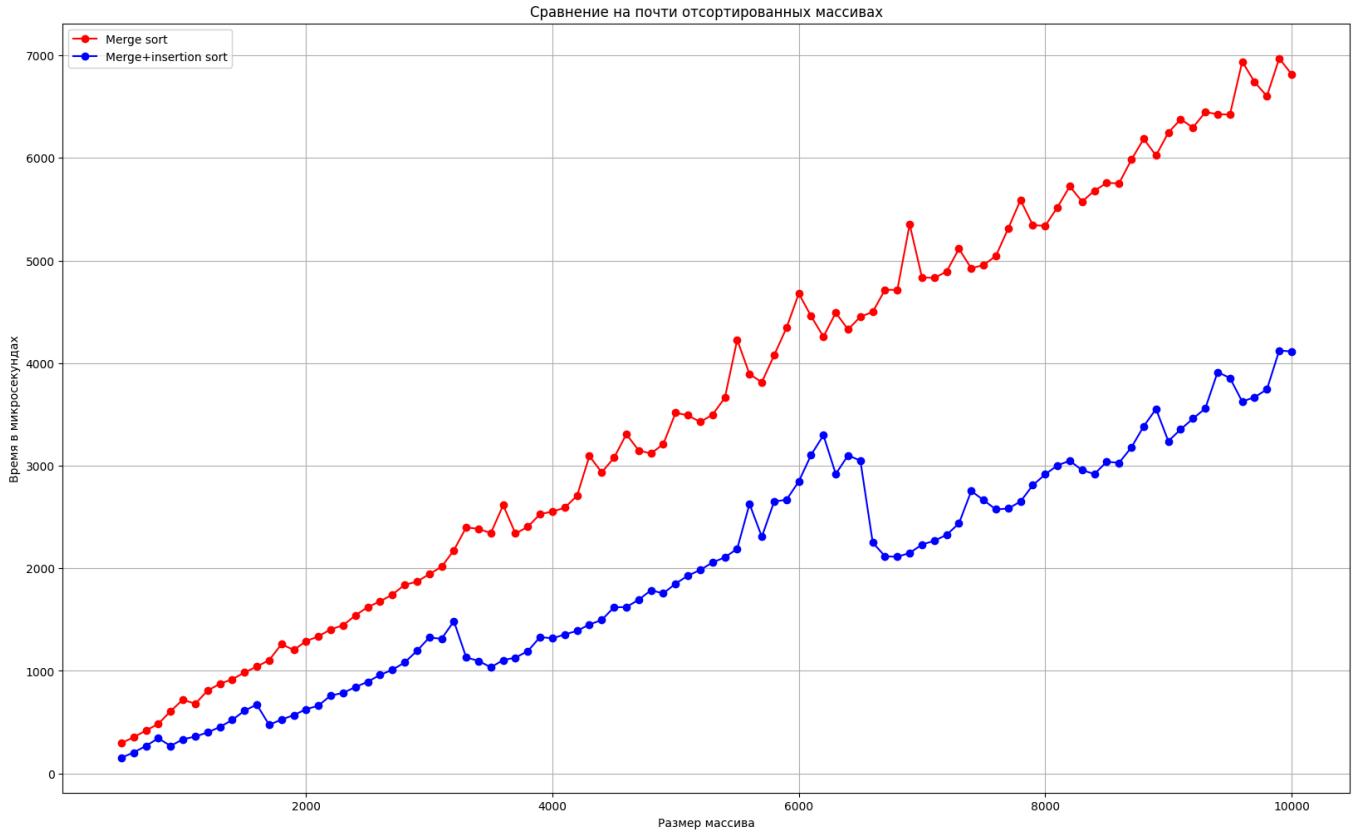
threshold = 30



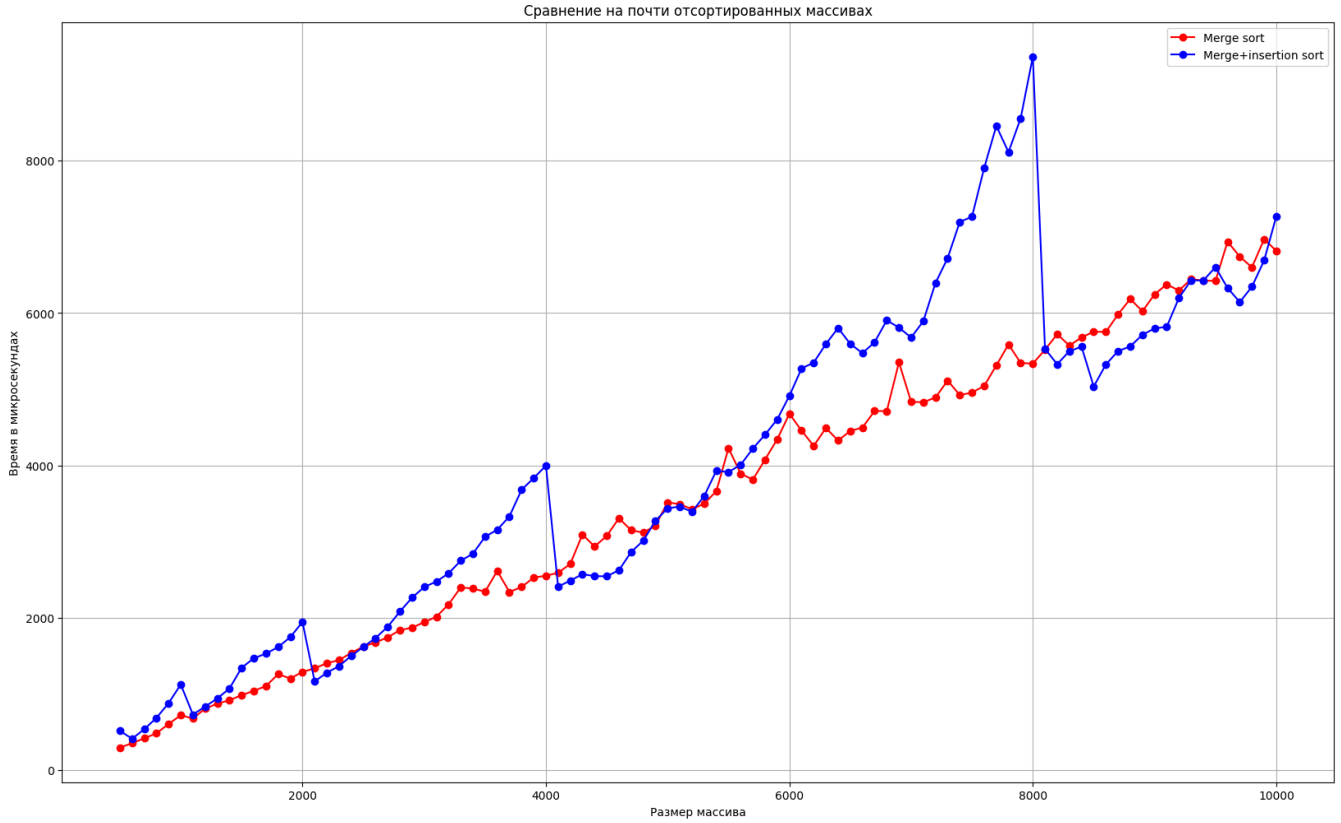
threshold = 50



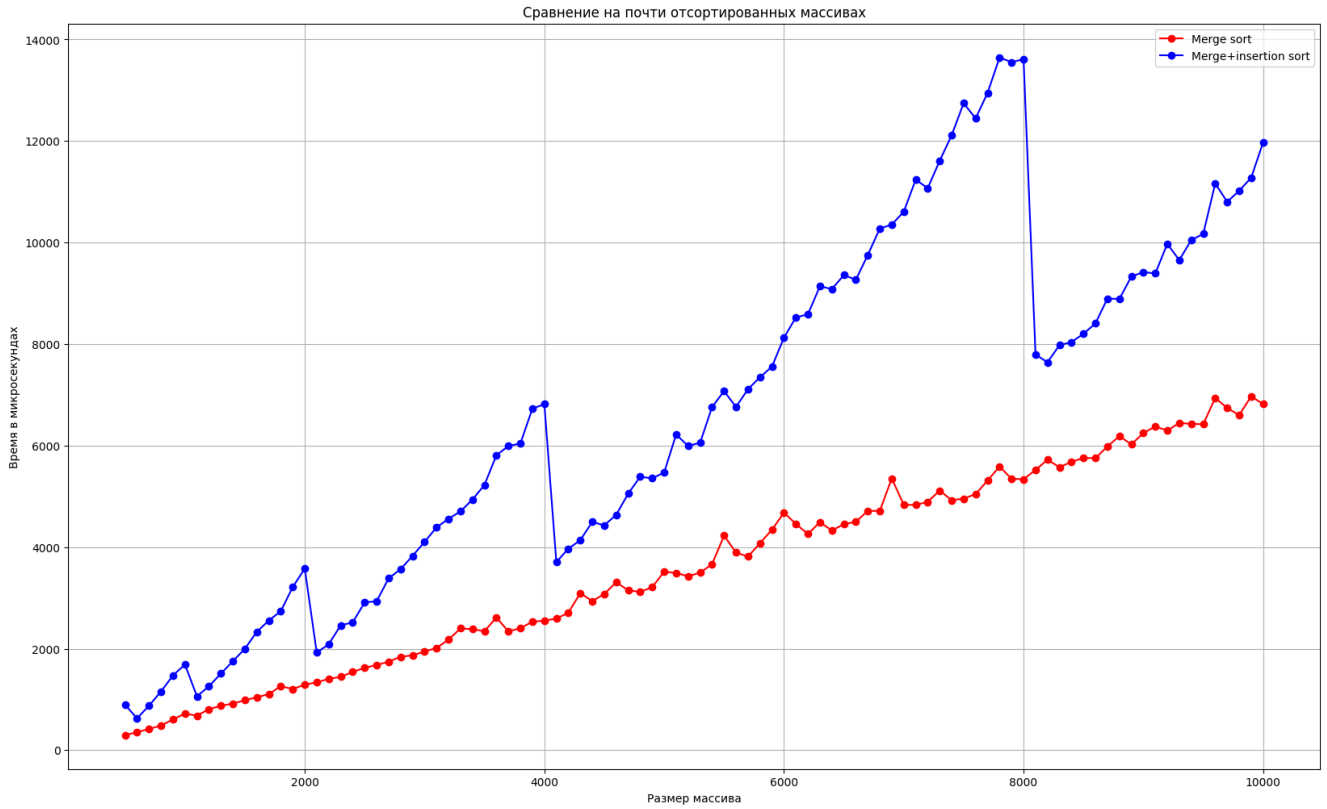
threshold = 100



threshold = 250



threshold = 500



Время работы стандартного merge sort на почти отсортированном массиве почти не отличается от работы на отсортированном массиве. Точно так же до threshold = 100 заметна сильная разница во времени работы гибридной и стандартной сортировки в пользу первой, при threshold = 100, значения начинают приближаться, и после 250 уже заметно, что гибридная сортировка начинает проигрывать, после 500 merge + insertion работает до 2x раз хуже.