A3

Пётр Лопаткин

23 ноября 2024 г.

1 Введение

В рамках данной задачи был разарботан класс **ArrayGenerator**, для автоматической генерации тестов. Его можно найти в соответствующем файле внутри репозитория. Параметры генерации использованы согласно заданию, но было проигнорировано условие про выделение подмассивов меньшего размера из большего, т.к в этом случае данные получались недостаточно случайными(на мой взгляд). Для самого тестирования был разработан класс **SortTester**, который также можно найти внутри репозитория. В результате тестирования гибридного алгоритма сортировки и классического алгоритма сортировки QuickSort было выявлено следующее.

2 Эмпирический анализ

В ходе прогона тестов были получены следующие результаты:

2.1 Рандомно сгенерированные массивы

На рисунке 1 приведены результаты тестрования гибридного алгоритма сортировки и классического qiuckSort. Тренд графиков выглядит линейно, т.к координаты расположены не равномерно, но на результаты анализа не влияет. На графике легко увидеть, что гибридная сортировка работает быстрее, но незначительно, что в целом логично, т.к в асимпотике мы не сильно выигрываем тут. Статистически QuickSort работает не хуже HeapSort или InsertionSort. Однако можно заметить, что где у QuickSort резкие скачки во времени у Introsort они в обратном направлении, то есть в случае ухода в глубокую рекурсию срабатывает HeapSort, что позволяет избежать резкого скачка времени как у QuickSort. Однако у Introsort всё равно присутствуют резкие скачки, которые я объяснить не смог. Есть гипотеза, что это из-за использования InsertionSort, либо особенностей c++(многочисленные вызовы функций и дерево рекурсии).

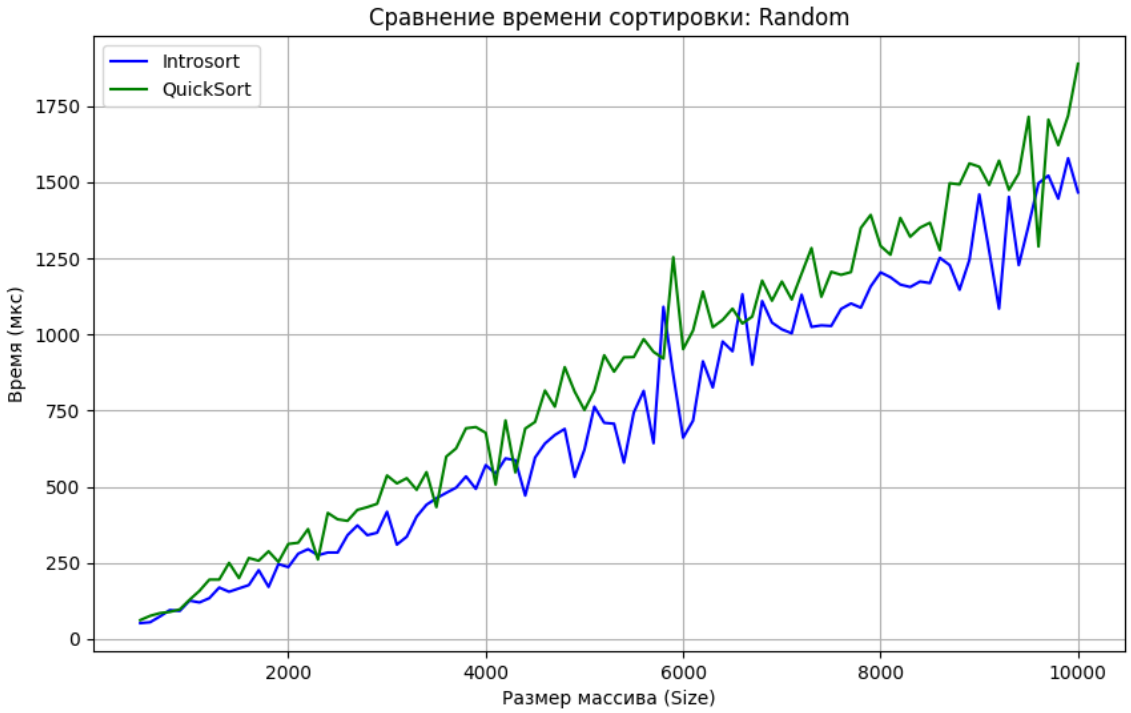

Рис. 1. Скорость сортировки рандомно сгенерированных массивов

2.2 обратно отсортированные массивы

На рисунке 2 приведены результаты тестрования гибридного алгоритма сортировки и классического mergeSort. Тренд графиков выглядит линейно, т.к координаты расположены не равномерно, но на результаты анализа не влияет. В случае с почти отсортированными массивами отрыв между алгоритмами уже более значительный. Также отсутствуют скачки там, где они есть у QuickSort, что подтверждает эффективность гибридности. Но все еще есть отдельные скачки.

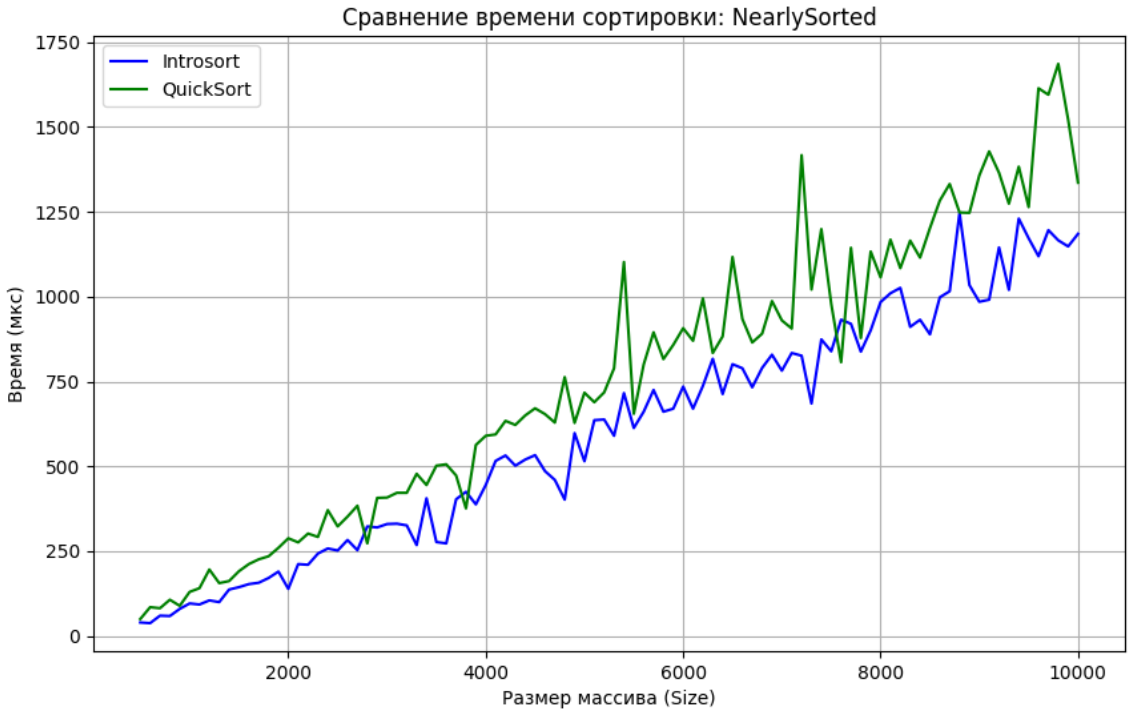

Рис. 2. Скорость сортировки почти отсортированных массивов.

2.3 Почти отсортированные массивы

На рисунке 3 риведены результаты тестрования гибридного алгоритма сортировки и классического mergeSort. Тренд графиков выглядит линейно, т.к координаты расположены не равномерно, но на результаты анализа не влияет. Тут Отрыв уже не такой значительный, но он есть. Также тут уже не все пики QuickSort имеют контрпики Introsort. Мне кажется, что это связано с тем, что мы в любом случае уходим сначала в глубокую рекурсию, а потом уже используем HeapSort. Обратно отсортированный массив, это худший случай для QuickSort, поэтому в глубокую рекурсию мы тут уйдем в любом случае. В остальном алгоритм все еще работет, причем чем больше массивы, тем больше отрыв.

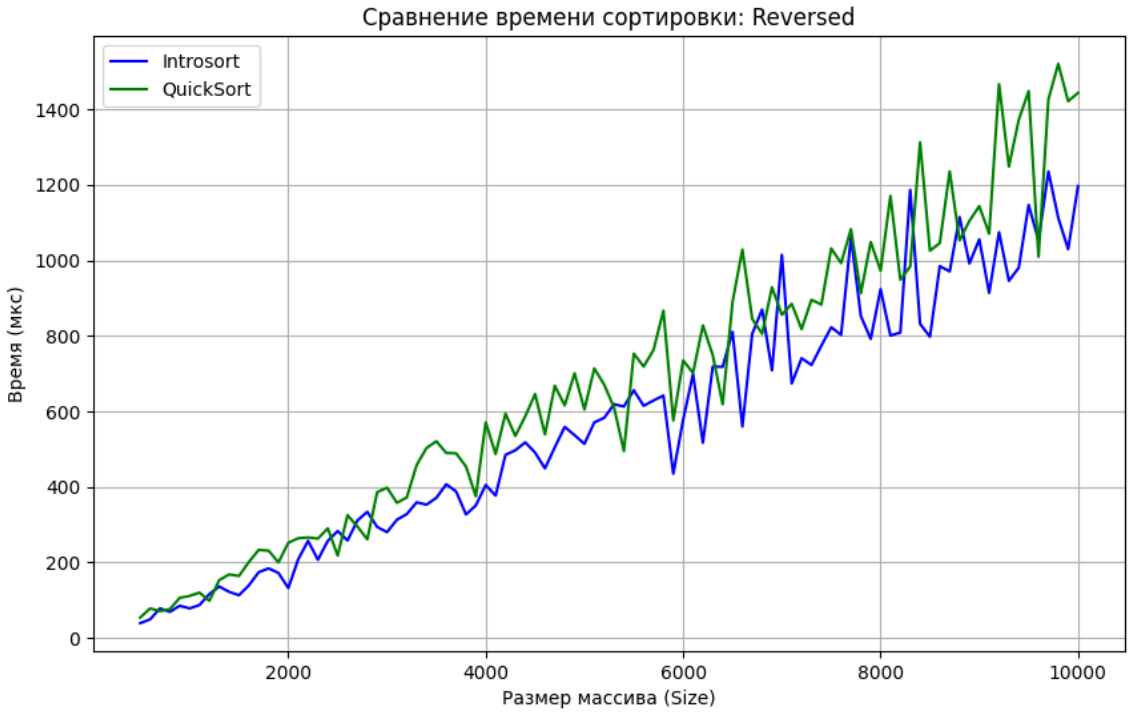

Рис. 3. Скорость сортировки обратно отсортированных массивов.

3 Заключение

По итогу гибридная сортировка работает значительно быстрее по времени, однако не сущетсвенно быстрее на тех значениях которые мы рассмотрели. Т.к QuickSort основан на случайных чисел, надо понимать, что чем больше у нас длинна массива, тем оптимальней будет работать гибридная сортировка. Также наша сортировка имеет более гладкий(крайне субъективно) график по сравнению с QuickSort, т.к часть пиков которые появляются из-за "плохих"случаев Quicksort мы компенсируем применением гибридной технологии. Но не все пики мы компенсируем и где-то даже создаем новые, которые уступают QuickSort. Откуда они берутся сложно сказать, т.к у нас случайный выбор опорного элемента. Мне кажется эта сортировка применима для крайнебольших массивов где мы абсолютно не уверены в том как выглядят наши данные.

4 Данные:

ID посылки: 292986898

Реализации классов: реализации

Ссылка на репозиторий со всеми данными репозиторий