А1. Анализ строковых сортировок

Лазарева Александра Константиновна БПИ 234

Репозиторий: <https://github.com/lazarevaak/A1.AnalyzingStringSorts.git>

A1m: [321235575](https://dsahse.contest.codeforces.com/group/NOflOR1Qt0/contest/608956/submission/321235575)

A1q: [321171854](https://dsahse.contest.codeforces.com/group/NOflOR1Qt0/contest/608956/submission/321171854)

A1r: [321172997](https://dsahse.contest.codeforces.com/group/NOflOR1Qt0/contest/608956/submission/321172997)

A1rq: [321233188](https://dsahse.contest.codeforces.com/group/NOflOR1Qt0/contest/608956/submission/321233188)

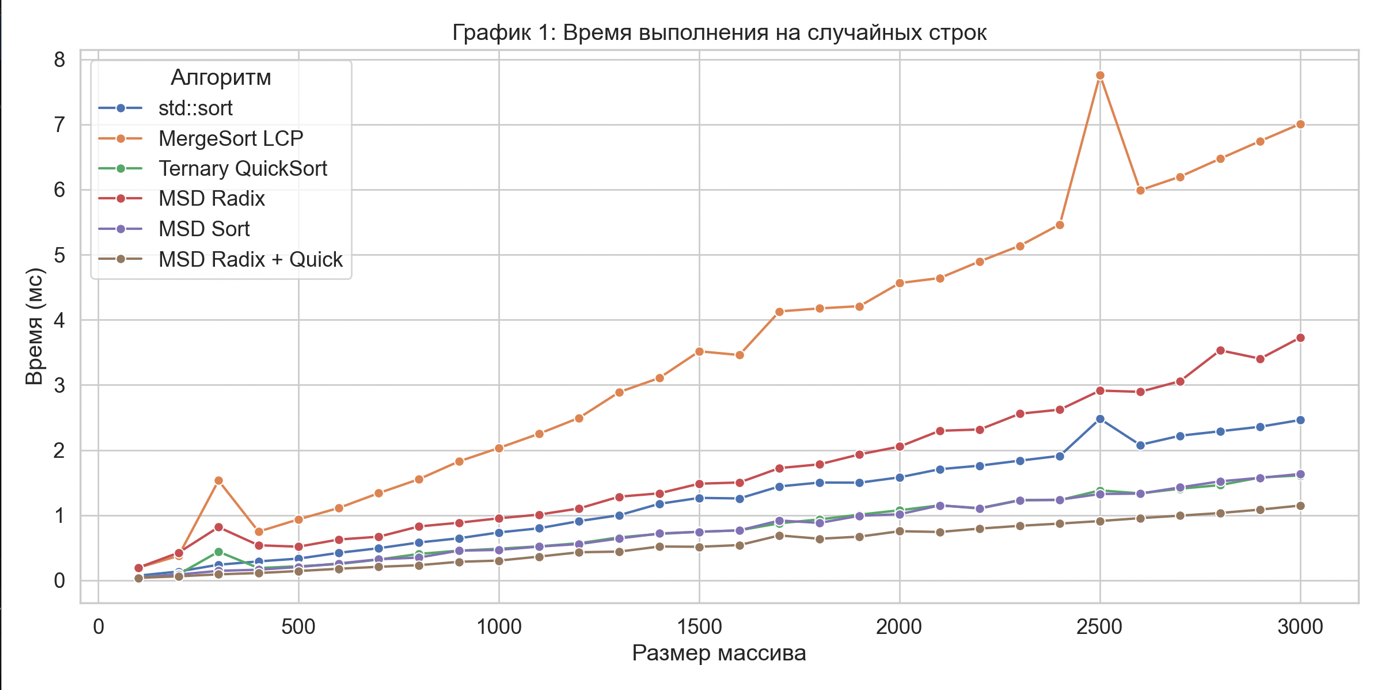
Цель данного исследования — сравнение эффективности различных алгоритмов сортировки строк на практике. В отличие от сортировки чисел, сортировка строк требует посимвольного сравнения, что влияет на фактическое время выполнения и делает традиционные оценки сложности (например, Ω(n·log n)) неполными. Для более точной оценки производительности были реализованы и протестированы как стандартные, так и специализированные алгоритмы сортировки, учитывающие особенности строк.

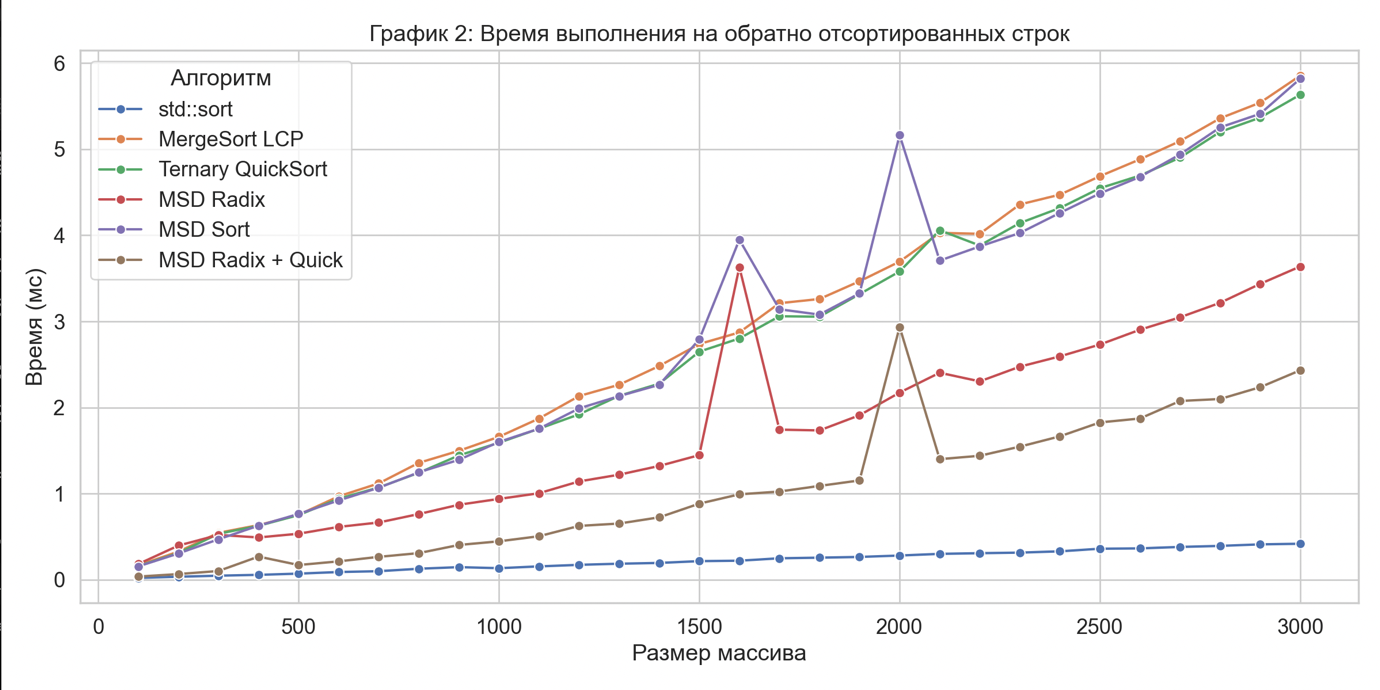
В рамках эксперимента были реализованы: стандартная сортировка std::sort, адаптированный MergeSort с учётом общего префикса (LCP), тернарный QuickSort, MSD Radix Sort и MSD Radix Sort с переключением на QuickSort при малых подмассивах. Каждый алгоритм замерялся по двум критериям: среднее время выполнения и количество посимвольных сравнений.

Для объективного тестирования использовались четыре типа входных данных: случайные строки, обратно отсортированные, почти отсортированные и строки с общим префиксом. Генерация строк была реализована в отдельном модуле и позволяла варьировать длину массива от 100 до 3000 с шагом 100, а также длину самих строк от 10 до 200 символов.

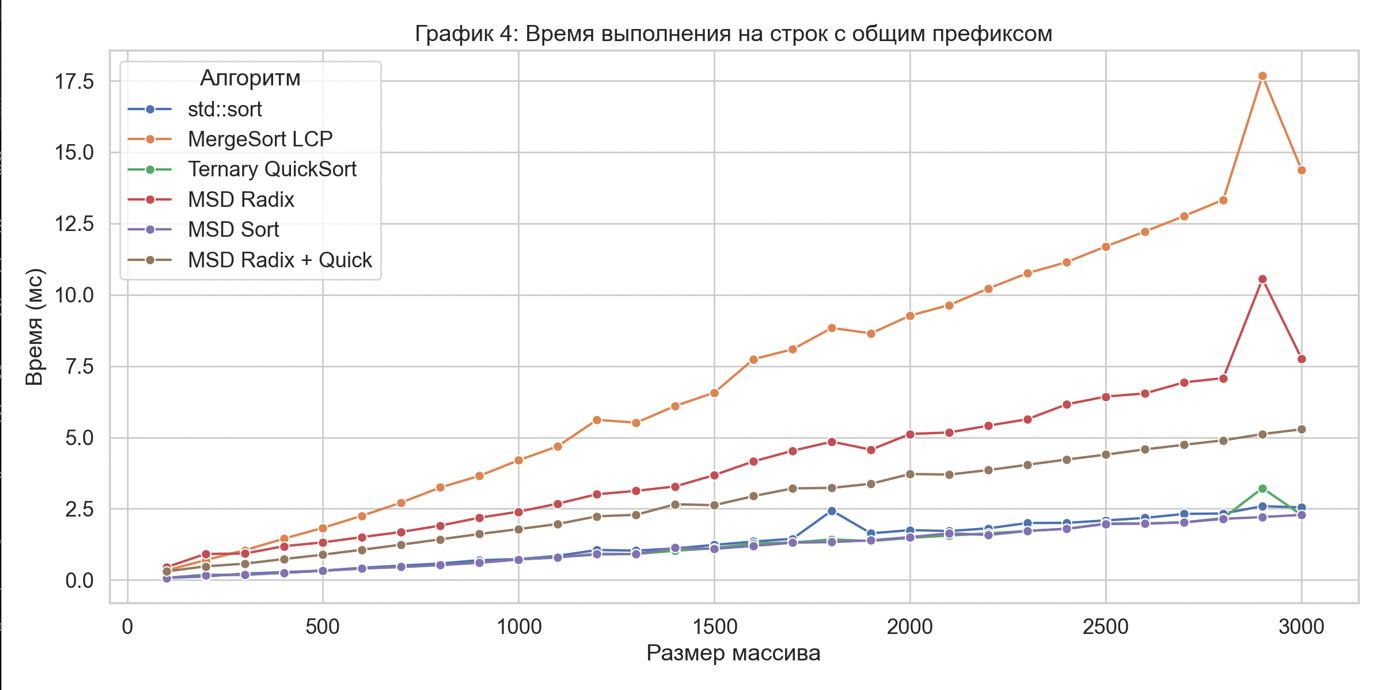
Результаты замеров были сохранены в CSV-файл и визуализированы с помощью графиков, отражающих зависимость времени работы и числа сравнений от размера входного массива для каждого алгоритма и каждого типа данных. Далее приведён подробный анализ полученных графиков.

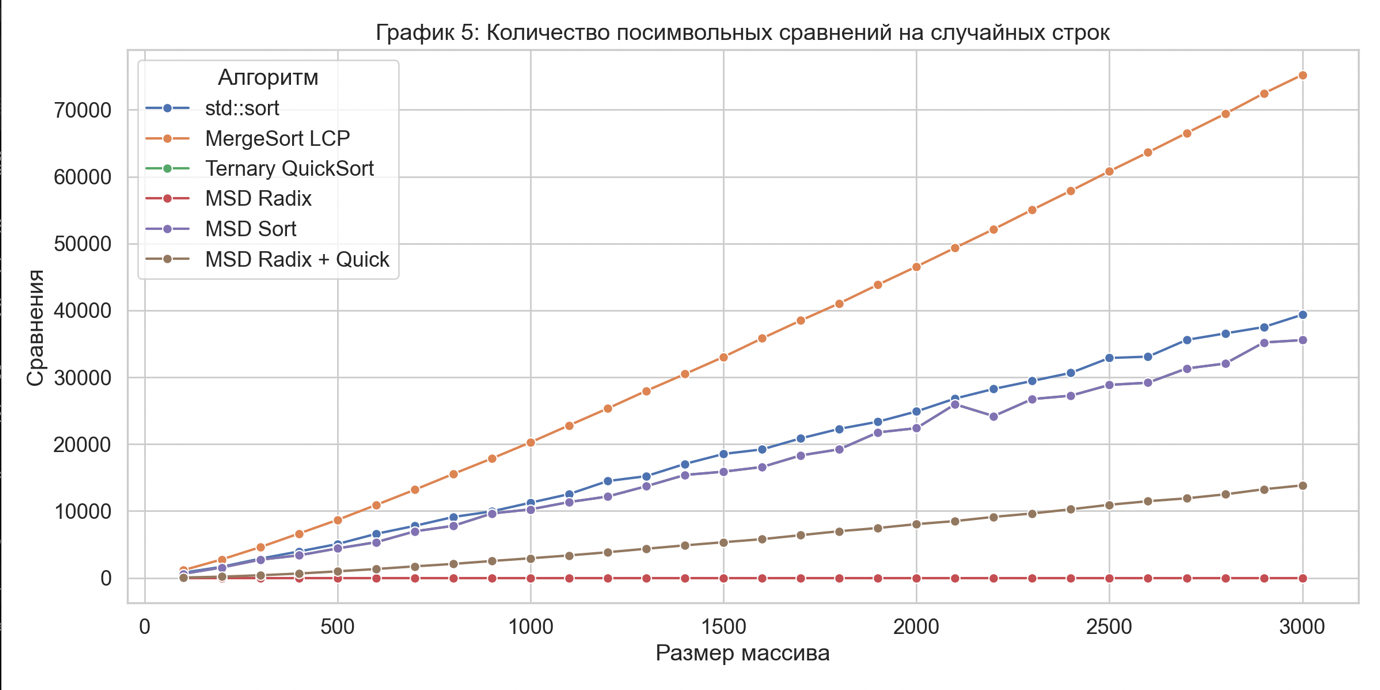
**Графики**

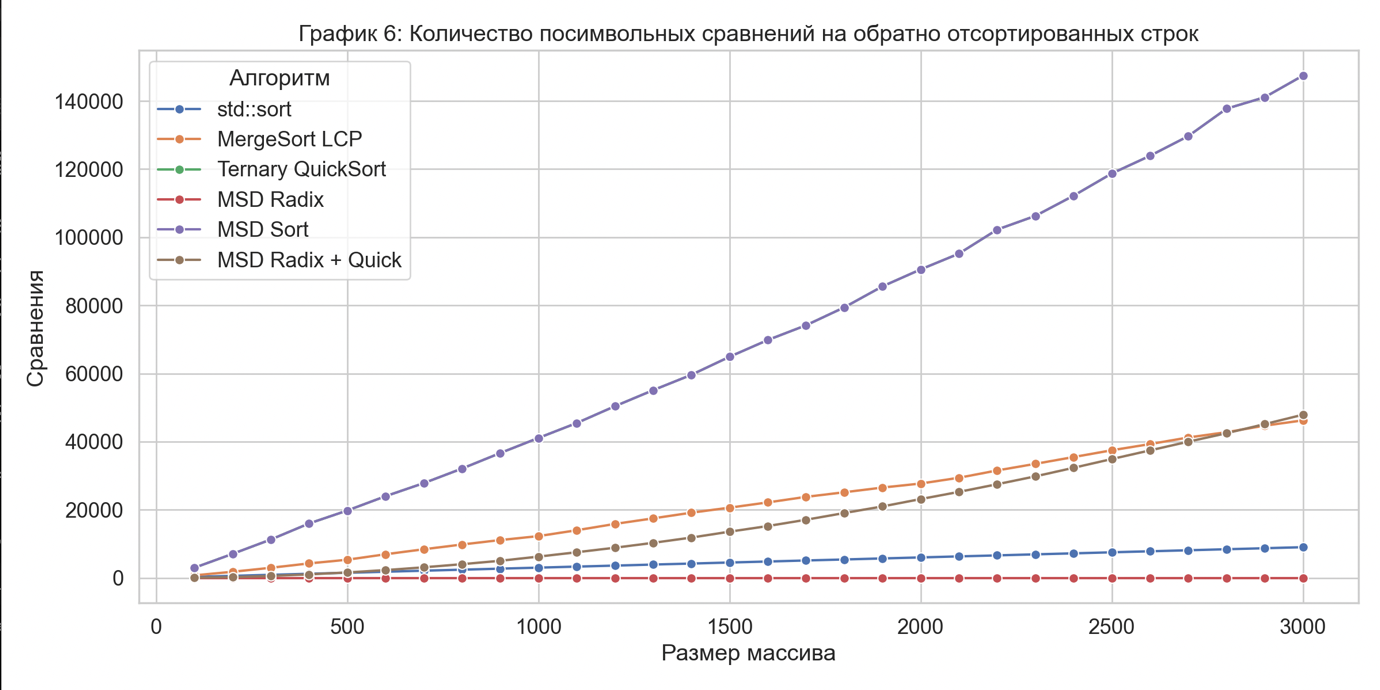


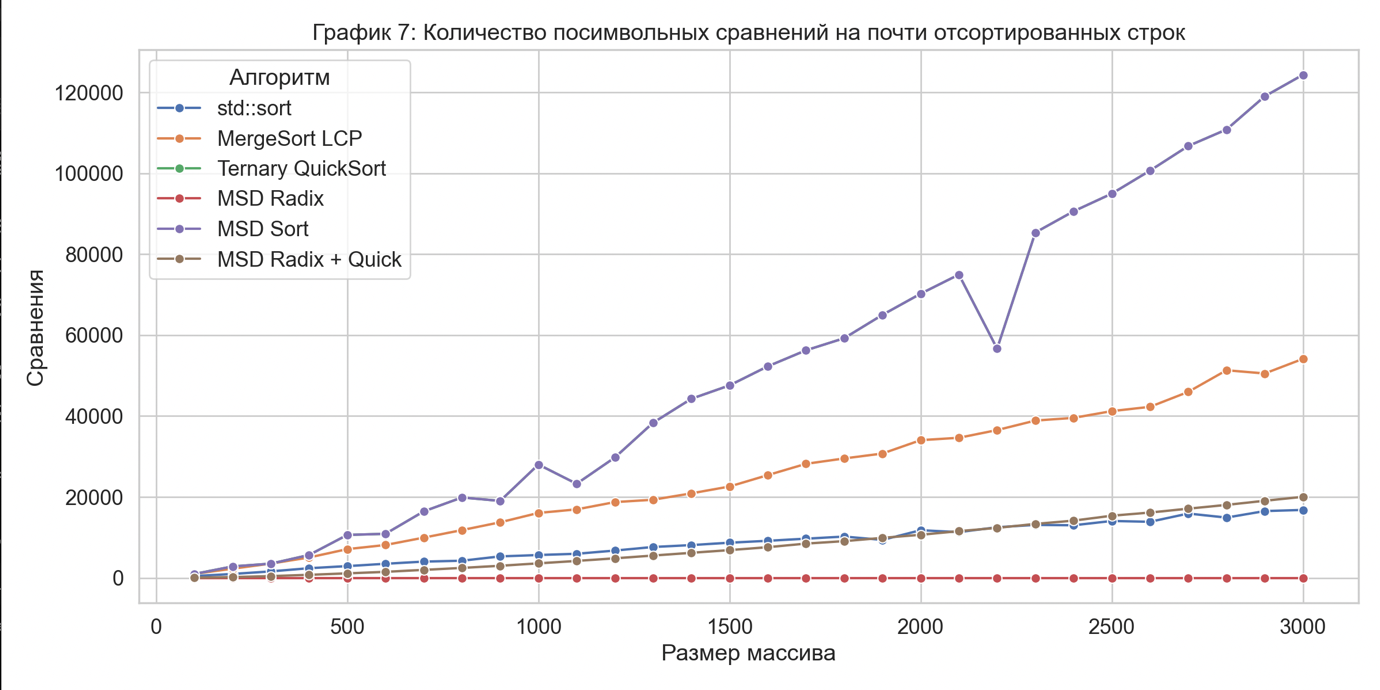


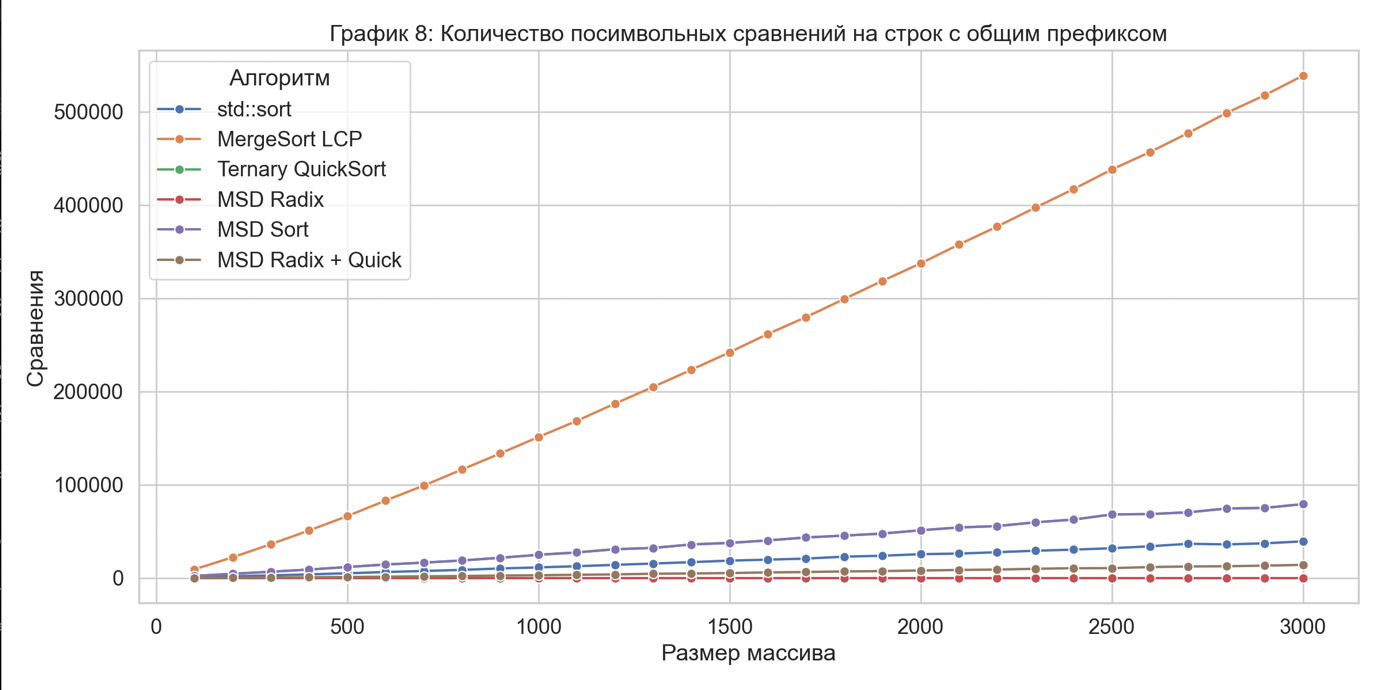












**Результаты и выводы:**

Алгоритмы на базе MSD (особенно MSD Radix + Quick) показывают наилучшие результаты по времени и числу сравнений в большинстве случаев, особенно при наличии общего префикса и большого размера данных. std::sort оказывается универсальным и надёжным выбором с предсказуемым временем. MergeSort LCP демонстрирует худшие результаты, что указывает на его низкую эффективность при работе с длинными строками и большим числом совпадающих символов. Таким образом, выбор алгоритма должен зависеть от структуры входных данных, но на практике рекомендуется использовать MSD Radix + Quick или std::sort для достижения наилучших результатов.