By Oleg Sidorenkov, BSE204

Отчёт по КДЗ ПиАА.

1. Введение

Данная работа написана с целью дать ответ на тревожащий всех: и студентов, и профессоров, и джунов, и тимлидов, вечный вопрос: “Какая сортировка лучше?” Спойлер - это *std::sort*. Однако нашей целью является исследование сортировок, написанных с помощью собственных мозгов и магии CLion с настроенными санитайзерами.

В представленном файле находится только самая важная и/или интересная информация по мнению автора. Чтобы ознакомиться с остальными графиками, посмотрите файл *analysis.ipynb*.

2. Терминология

- Логарифмическая сортировка (для краткости) - сортировка, работающая за $O\left(N \log N\right)$.

3. Использованные инструменты

В представленной работе используются языки C++ и Python (для аналитики), IDE CLion и IDE DataSpell (для аналитики).

4. Архитектура проекта

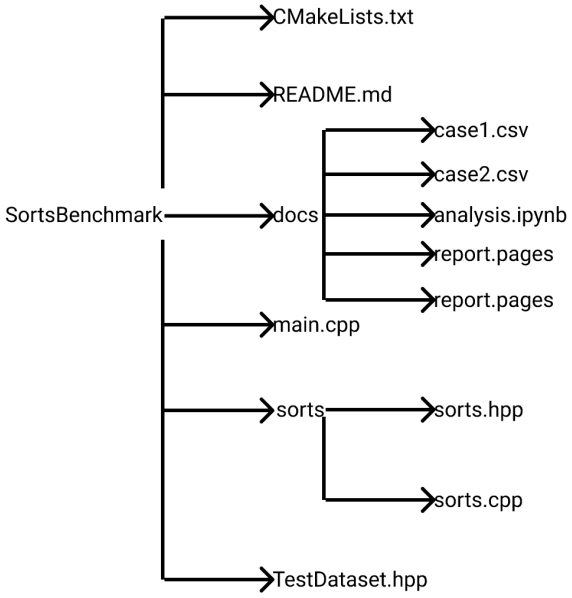

Рис. 1. Структура проекта.

- CMakeLists.txt - Настройки CMake
- README.md - Это чтобы потом отчёт красиво переписать для репозитория
- docs
  - case1.csv - Тесты с маленькими массивами
  - case2.csv - Тесты с большими массивами
  - analytics.ipynb - Jupyter Notebook с анализом данных и графиками, которые будут фигурировать ниже
  - report.pages - Если мою работу будет проверять Никита или другой маковод. Всё для Вашего удобства :)
- main.cpp - Тут всё для создания различных видов массивов, логов и записи в .csv
- sorts
  - sorts.hpp - Header-file с содержащимися функциями
  - sorts.cpp - реализация функций
- TestDataset.hpp - Создание и мутация массивов

5. Маленькие массивы.

5.1. Рандомно заполненный маленькими числами массив.

Рассмотрим, как ведут себя различные сортировки на массивах длиной $N \in [50;\ 300]$ и рандомным наполнением $x_i \in [0;\ 5]$.

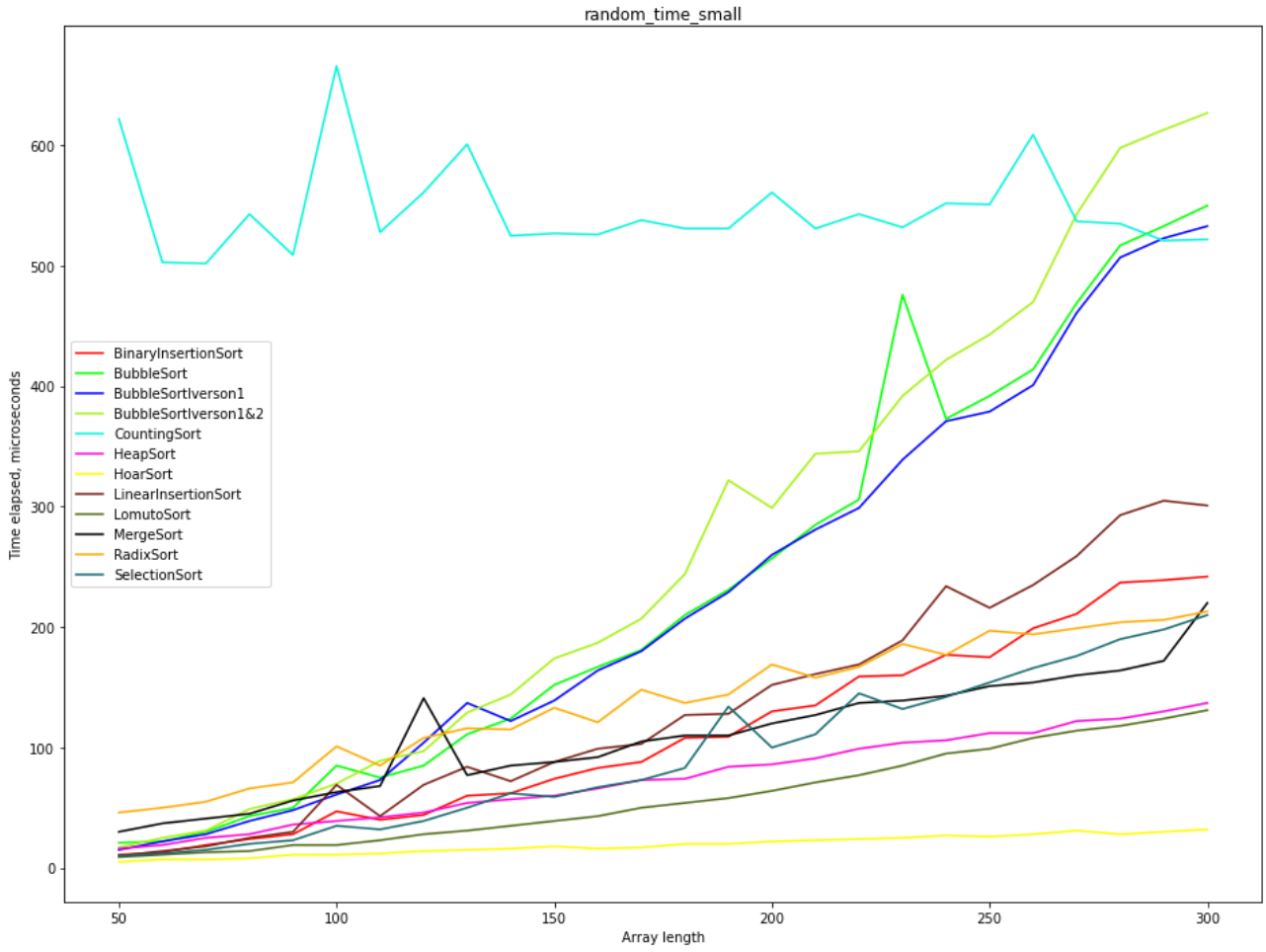

Рис. 2. Рандомно заполненный маленькими числами массив.

Результаты достаточно ожидаемые: пузырёк растёт быстрее остальных, а для других квадратичных сортировок 300 элементов маловато, чтобы радикально отличаться от логарифмических сортировок. Ярко отличаются лишь Counting Sort и Hoar Sort, очень хорошо подчёркивая своё название (quick sort).

5.2. Рандомно заполненный массив с большими числами.

Рассмотрим, как ведут себя различные сортировки на массивах длиной $N \in [50;\ 300]$ и рандомным наполнением $x_i \in [0;\ 4000]$.

Теперь все квадратичные сортировки стали более близко друг к другу находиться. Это произошло потому что Selection Sort и Insertion Sort стали делать больше шагов по причине увеличения количества уникальных значений.

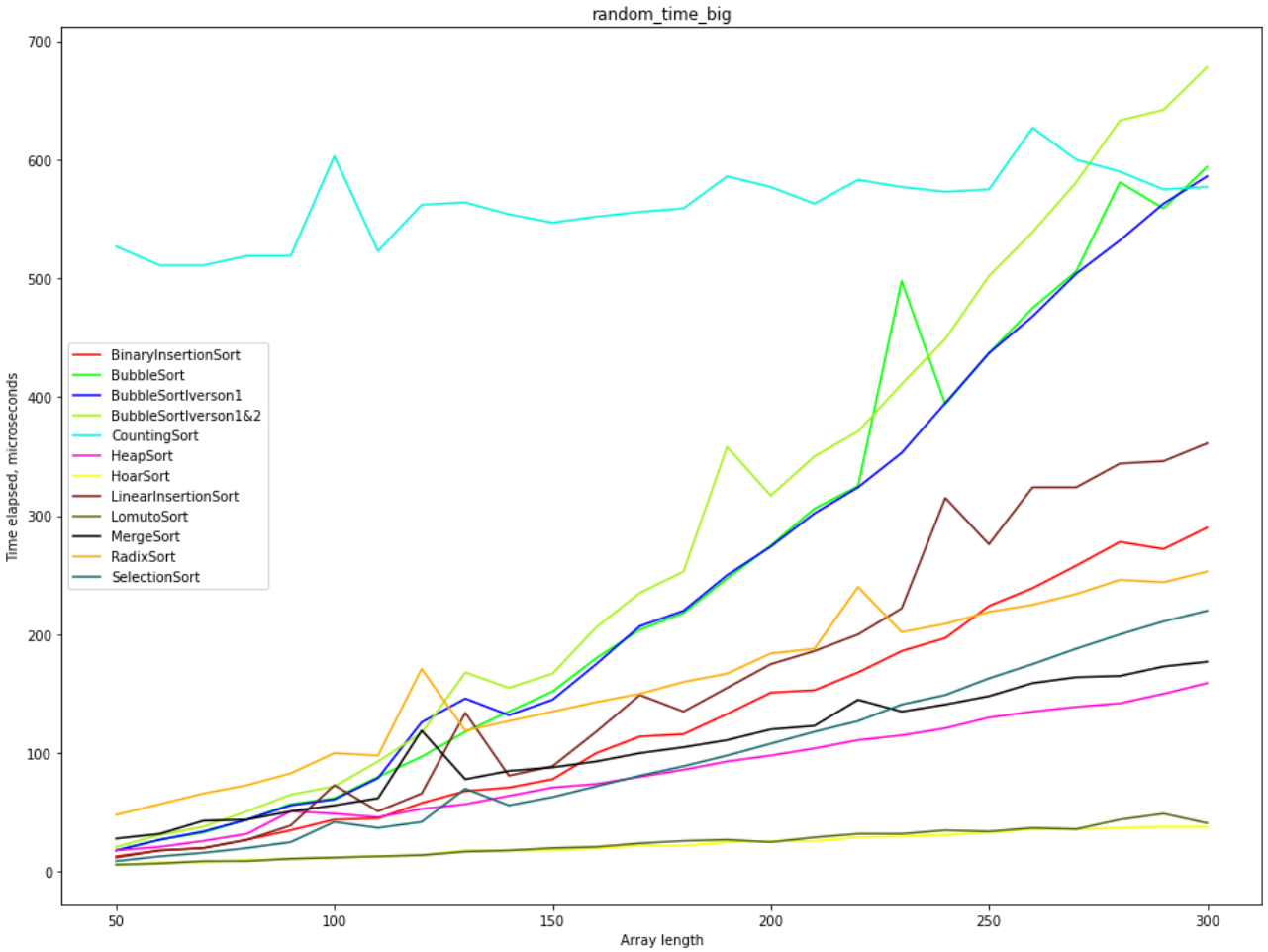

Рис. 3. Рандомно заполненный большими числами массив.

5.3. Почти отсортированный массив.

Рассмотрим, как ведут себя различные сортировки на почти отсортированных массивах длиной $N \in [50;\ 300]$.

И снова результат абсолютно ожидаем: Ломуто работает очень долго, так как крайним берётся последний элемент. Резкий скачок вниз у всех сортировок, завязанных на составлении отсортированной части массива, например, Selection Sort, а так же у сортировок, проверяющих, не отсортирован ли массив, например, Bubble Sort Iverson.

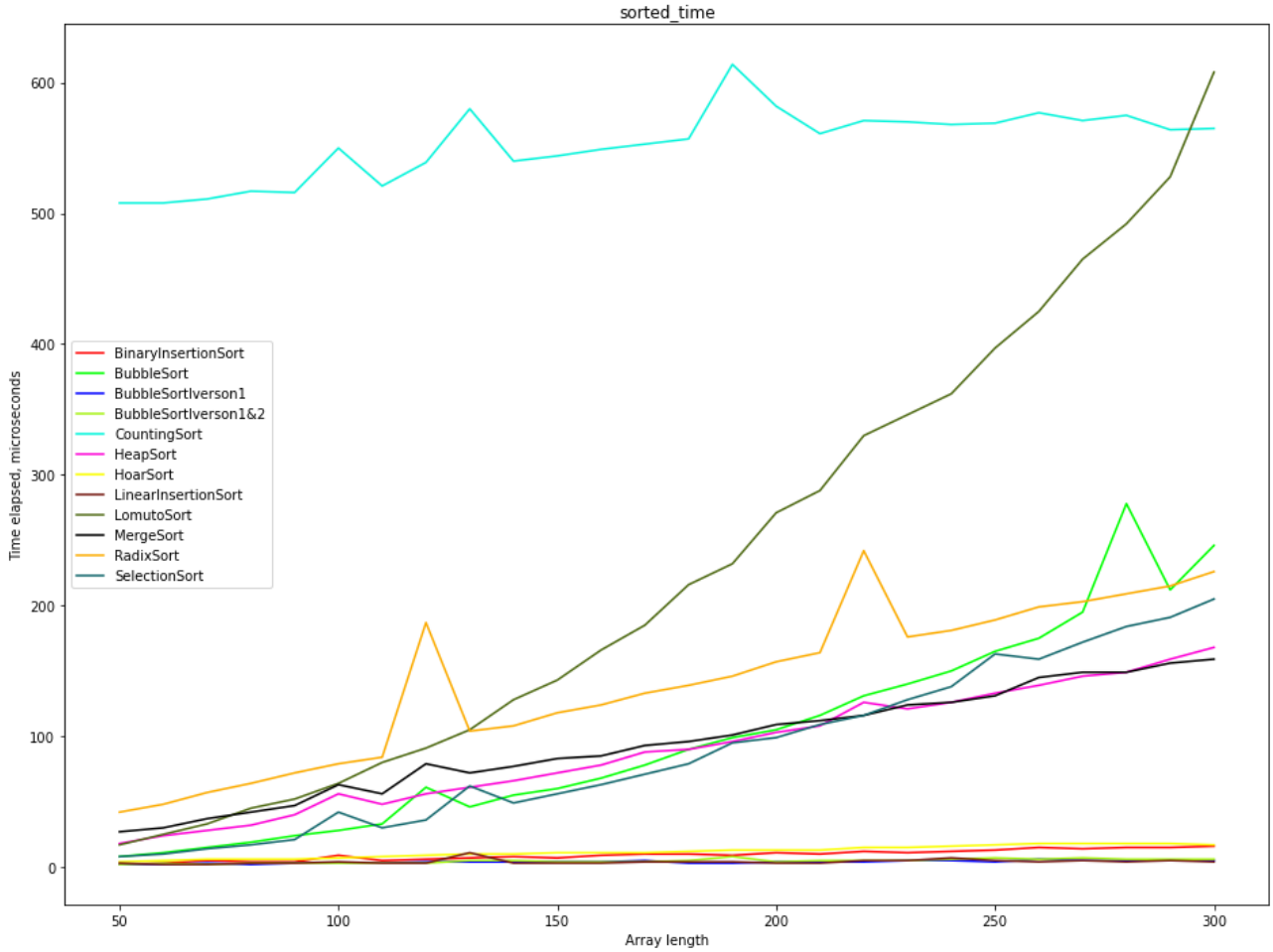

Рис. 4. Почти отсортированный массив.

5.4. Обратно отсортированный массив.

Рассмотрим, как ведут себя различные сортировки на почти обратно отсортированных массивах длиной $N \in [50;\ 300]$.

На этот раз очень сильно прибавили во времени оба Selection Sort. Это было ожидаемо, так как им приходится перемещать очень много элементов влево, намного больше, чем при обычном рандомном массиве. Так же, очень быстро отработал Hoar Sort, скорее всего так как для отражения нужно не так много операций.

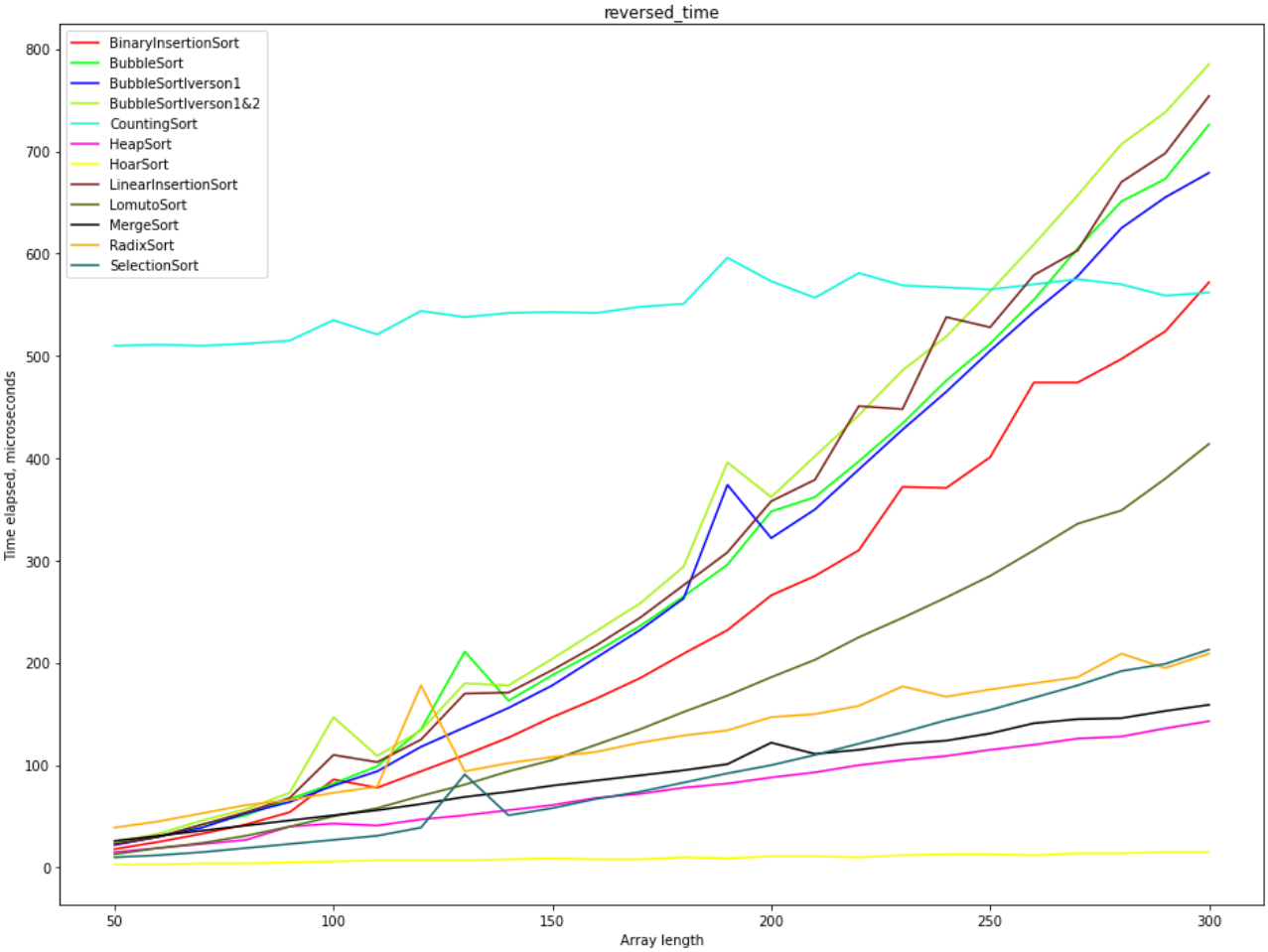

Рис. 5. Обратно отсортированный массив.

6. Большие массивы.

Из-за сильного разрыва во времени выполнения квадратичных и логарифмических сортировок, где последние практически все и всегда были возле 0, было принято решение представить графики с логарифмическим масштабом оси y.

5.1. Рандомно заполненный маленькими числами массив.

Рассмотрим, как ведут себя различные сортировки на массивах длиной $N \in [100;\ 4100]$ и рандомным наполнением $x_i \in [0;\ 5]$.

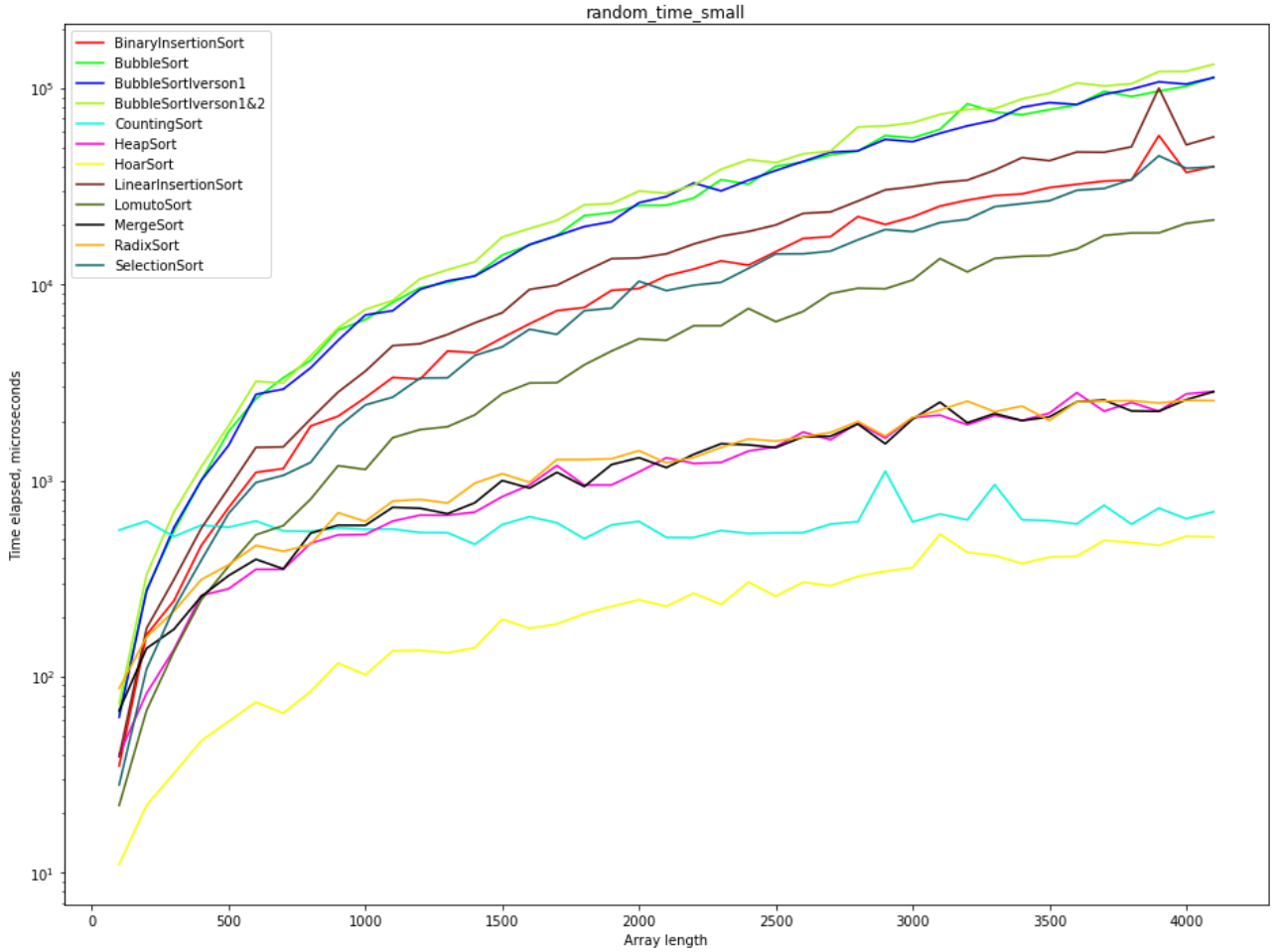

Рис. 6. Рандомно заполненный маленькими числами массив. Логарифмический масштаб оси y.

Очевидно, все логарифмические сортировки теперь стали выигрывать у квадратичных в несколько десятков (даже в определённых случаях сотен) раз. Это было абсолютно ожидаемо. Так же, в глаза резко бросается разница между разбиениями Хоара и Ломуто. Что удивительно, Hoar Sort выигрывает по производительности даже у Counting Sort. Предположительно, из-за того, что последняя была написана в более сложной своей реализации с дополнительным выделением памяти, что сильно сказалось на производительности.

6.2. Рандомно заполненный массив с большими числами.

Рассмотрим, как ведут себя различные сортировки на массивах длиной $N \in [100;\ 4100]$ и рандомным наполнением $x_i \in [0;\ 4000]$.

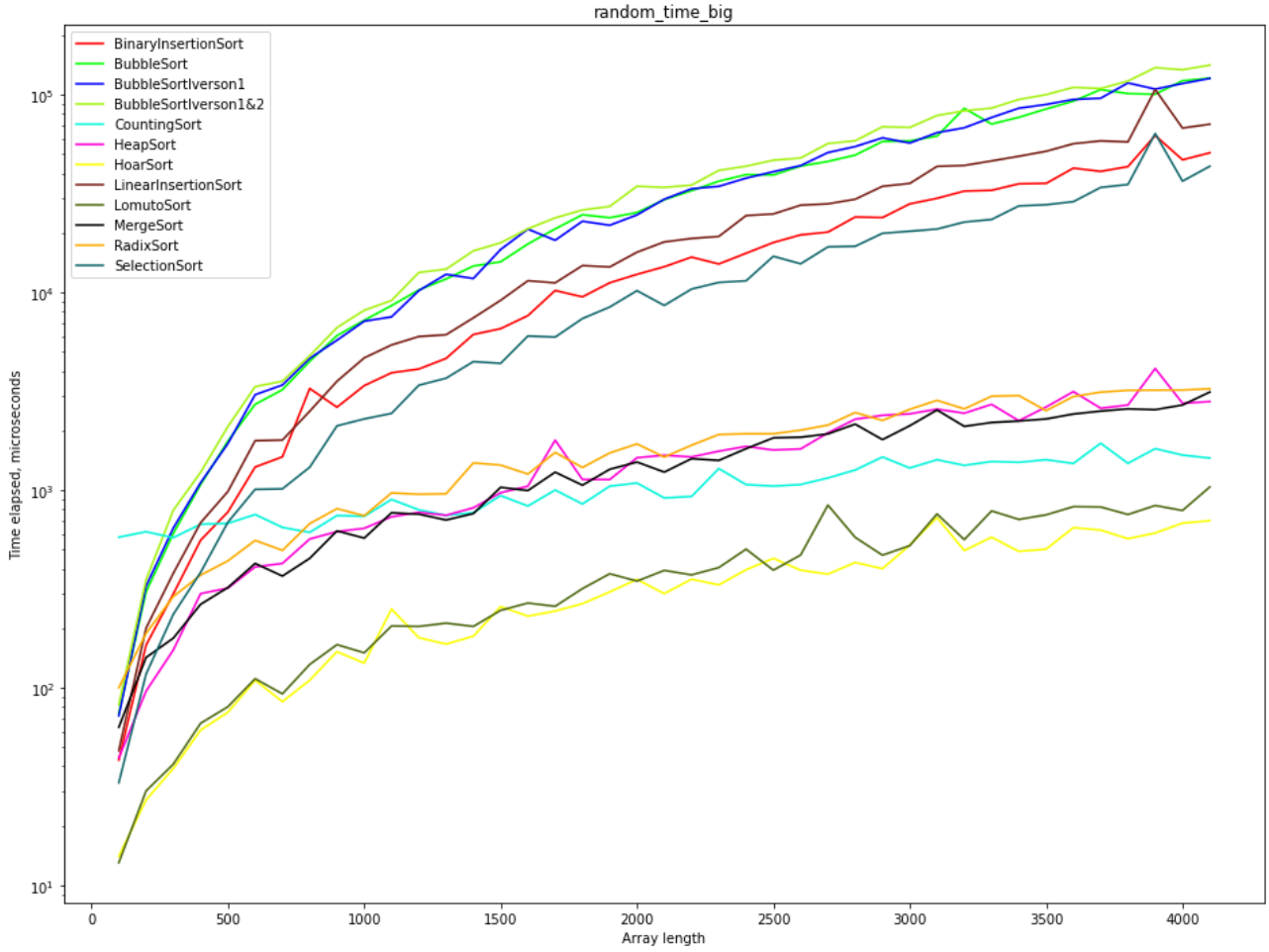

Рис. 7. Рандомно заполненный большими числами массив. Логарифмический масштаб оси y.

Картина практически не изменилась по сравнению с предыдущими результатами. Можно отметить только Counting Sort, который стал работать чуть дольше.

6.3. Почти отсортированный массив.

Рассмотрим, как ведут себя различные сортировки на почти отсортированных массивах длиной $N \in \left[100;\ 4100\right]$.

С почти отсортированным массивом картина резко меняется. Как и в пункте *4.3*, Lomuto Sort очень сильно вырос, став практически полностью квадратичным. То же самое произошло и с Selection Sort. Снова же, у алгоритмов, выделяющих отсортированную часть (Selection Sort и Insertion Sort) массива и у сортировок, проверяющих его на упорядоченность (Iverson), скорость очень сильно выросла.

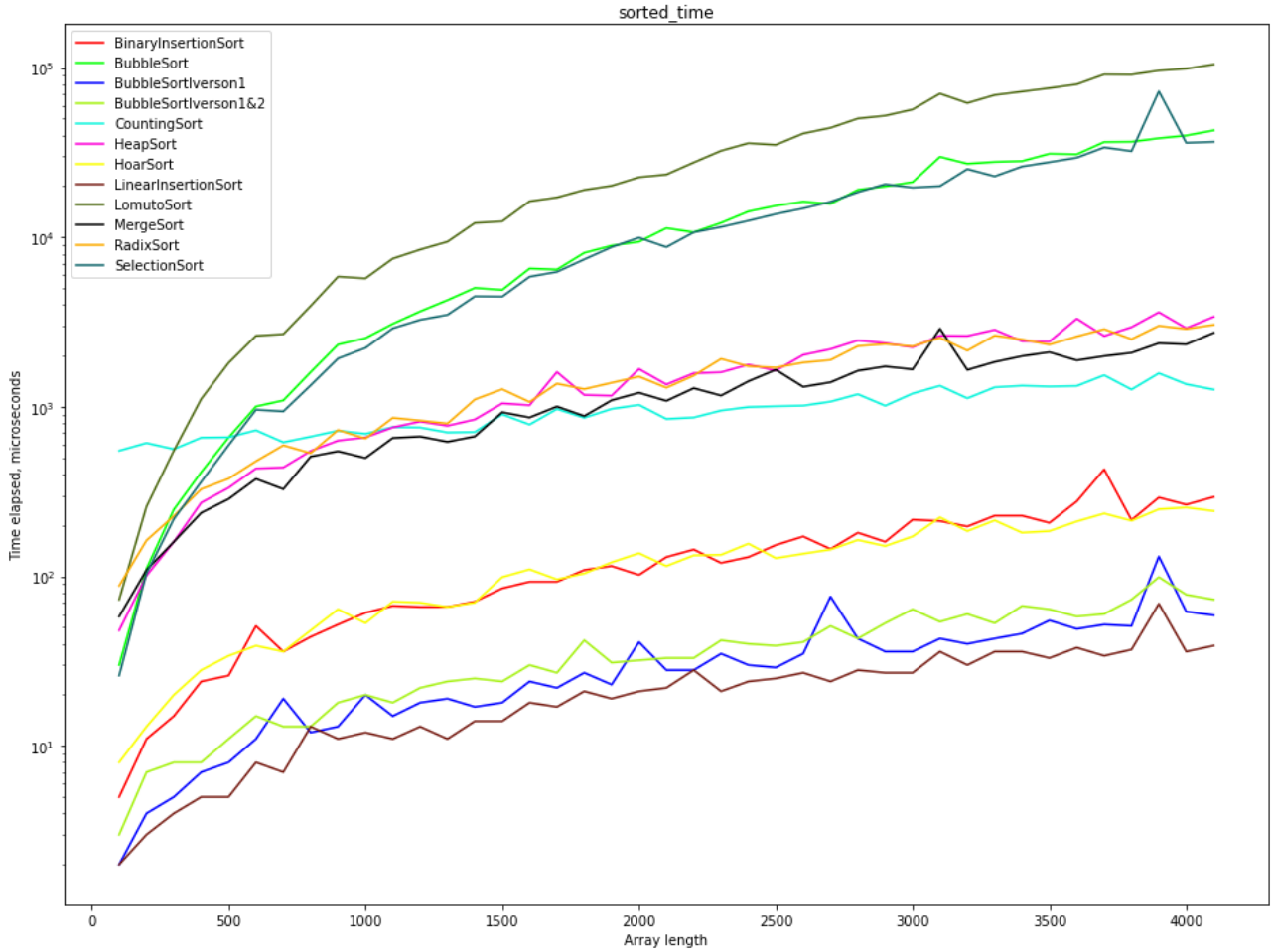

Рис. 8. Почти отсортированный массив. Логарифмический масштаб оси y.

6.4. Обратно отсортированный массив.

Рассмотрим, как ведут себя различные сортировки на почти обратно отсортированных массивах длиной $N \in [100;\ 4100]$.

Картина повторила прошлые замеры. Снова Hoar Sort вырвался вперёд, но на этот раз абсолютным победителем. Из таблицы видно, что в данном случае он работает за чистые $O\left(N \log\ N\right)$. Так же, в данном случае себя хорошо показали Counting Sort и Selection Sort (относительно квадратичных).

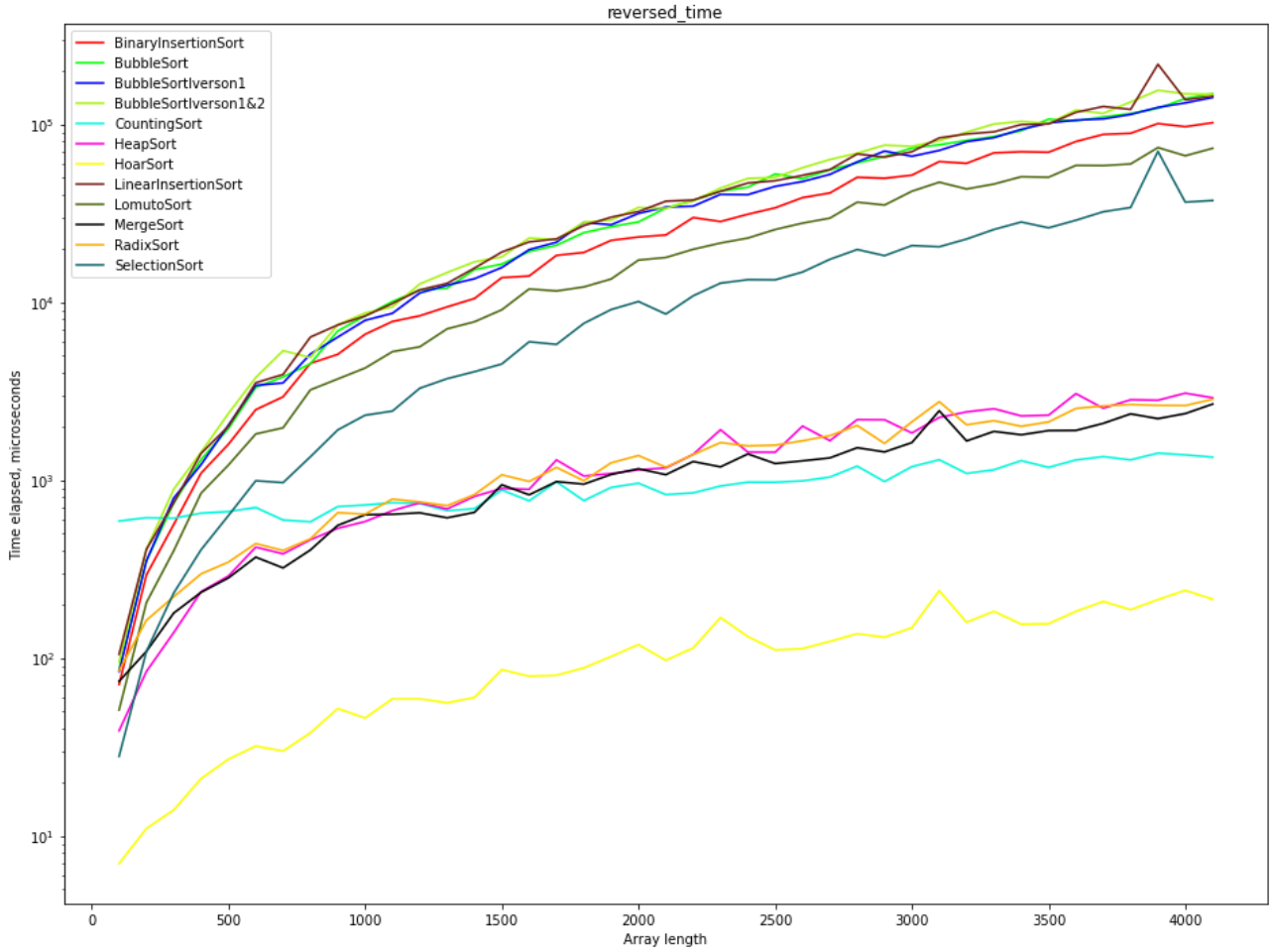

Рис. 9. Обратно отсортированный массив. Логарифмический масштаб оси y. Логарифмический масштаб оси y.

7. Выводы.

Давайте подводить итоги. Сделаем мы это в виде номинаций.

- Самая быстрая, не требующая лишней памяти и вообще максимально крутая - Hoar Sort. $\left(\text{Однако её можно заставить работать за квадрат с вероятностью} \frac{1}{N!}\right)$
- Самая медленная сортировка … Bubble Sort, кто бы мог сомневаться. Всегда $O\left(N^2\right)$.
- Самые стабильные по времени сортировки - Hoar Sort (очень редкие случаи с $O\left(N^2\right)$ не учитываем), Bubble Sort, Merge Sort, Radix Sort, Heap Sort.
- Сортировки, которые максимально хорошо показывающие себя в зависимости от ситуации (при сортированном массиве) - Bubble Sort Iverson 1, Bubble Sort Iverson 1&2, Linear Insertion Sort, Binary Insertion Sort
- Сортировка с самой необычной сложностью - Counting Sort с её $O\left(M+N\right)$.

Итого, абсолютный победитель - *HOAR SORT*!!!