ЭКСПЕРИМЕНТАЛЬНОЕ СРАВНЕНИЕ СОРТИРОВОК

ЦЕЛЬ

Экспериментальное определение временной сложности сортировок различных массивов разными методами и сравнительный анализ полученных результатов

МАССИВЫ

По наполнению:
- Случайные числа в диапазоне от 0 до 5
- Случайные числа в диапазоне от 0 до 4000
- Почти отсортированный (замена в каждой сотне)
- Числа в обратном порядке

По размеру:
- От 50 до 300 с шагом 50
- От 100 до 4100 с шагом 1000

Вся логика работы с массивами (заполнение, копирование, зануление, проверка на отсортированность, получение строкового представления) осуществляется с помощью класса `ArrayManager`

СОРТИРОВКИ

Итерационные:
- Сортировка пузырьком (обычная)
- Сортировка выбором минимума
- Сортировка бинарной вставкой

Линейные:
- Сортировка подсчетом (устойчивая)
- Цифровая сортировка (256ричная сс)

Рекурсивные:
- Сортировка кучей
- Быстрая сортировка (опорный - средний)
- Сортировка слиянием

Реализации самих сортировок описаны в классе `Sort`. Для удобства сигнатура методов одинаковая.

Все вспомогательные методы находятся в классе `Utility`

ТЕСТЫ

- Во время тестирования были отключены все сторонние процессы и интернет-соединение, чтобы избавиться от «шумов» в результатах
- Перед запуском тестов каждая сортировка прогоняется «вхолостую» 5 раз на массиве случайных чисел из 1000 элементов
- Генерируется 4 эталонных массива
- Тесты для каждой размерности запускаются 100 раз
- Каждый раз нужная часть массива копируется в буффер и сортируется там
- Методы запускаются через указатели
- Записывается средний результат за 100 запусков в наносекундах

Вся логика тестирования сформирована в классе `Test`

Запись результата осуществляется через класс `Writer`

ФАЙЛЫ

Код:

- `main.cpp` запуск тестирования
- `ArrayManager.h` работа с массивами
- `Writer.h` работа с файлами
- `Test.h` тесты
- `Sort.h` сортировки
- `Utility.h` инструменты

Результаты:

- `198_AbuAlLabanN_data50-300.csv`
  Результаты измерений для массивов размерностью 50-300 (сгенерирован программным кодом)
- `198_AbuAlLabanN_data100-4100.csv`
  Результаты измерений для массивов размерностью 100-4100 (сгенерирован программным кодом)
- `198_AbuAlLabanN.xlsx`
  Результаты измерений в наглядном виде + графики
- `198_AbuAlLabanN.pdf`
  Отчет

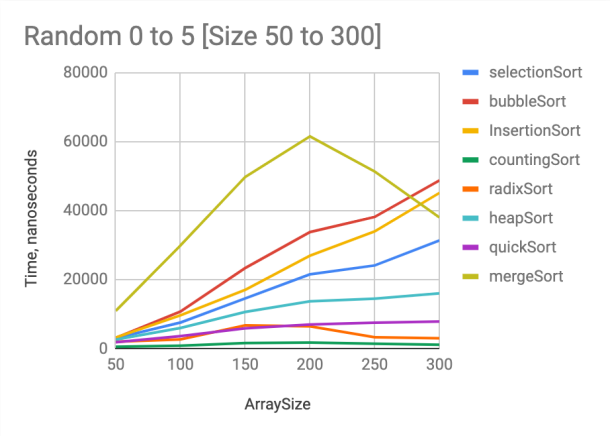
СРЕДА РАЗРАБОТКИ

Clion 2020.2
C++14

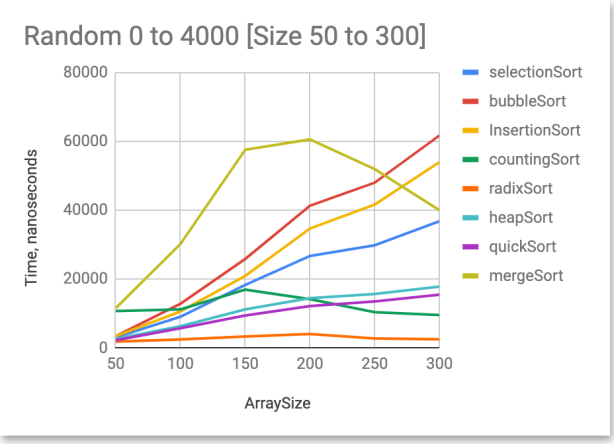
АНАЛИЗ РЕЗУЛЬТАТОВ

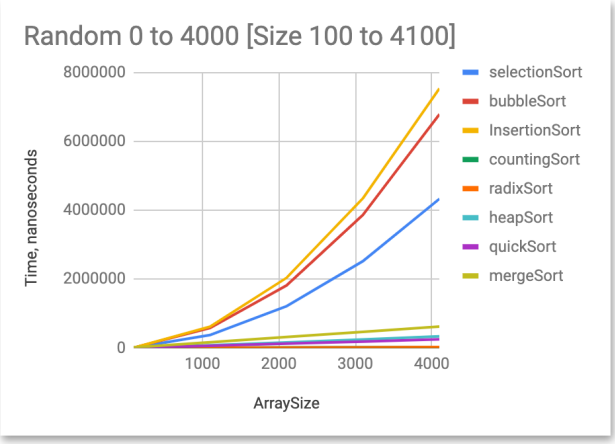
1. Графики по Видам Массивов

- Случайные числа от 0 до 5



- Случайные числа от 0 до 4000

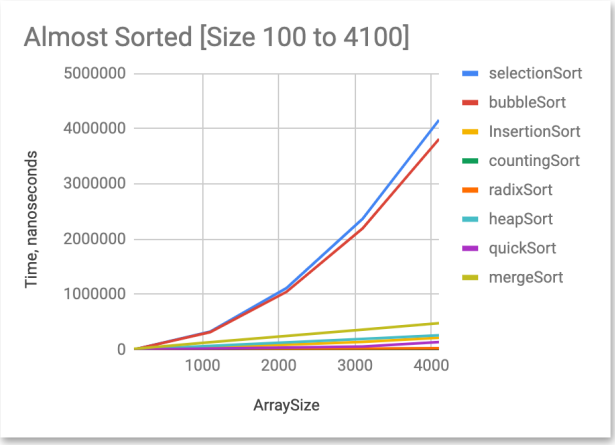




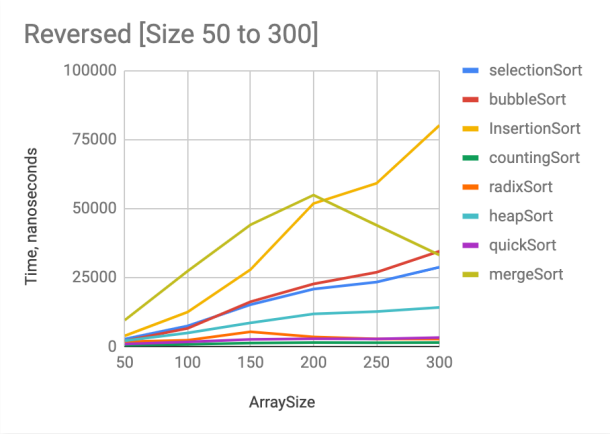
На графиках выше можно заметить, что на маленьких размерностях сортировка слиянием заметно отстает от остальных сортировок, хотя по сложности алгоритма такого быть не должно. Работу сортировки в данном случае затормаживает копирование массива, которое происходит внутри функции `merge`, поэтому мы и наблюдаем подобный результат на этих и последующих графиках.

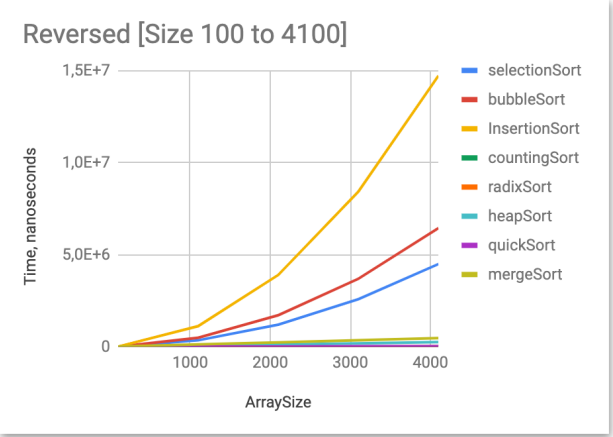
На графиках к массивам больших размерностей наблюдается сильный отрыв по скорости для итеративных сортировок. Медленнее всего в обоих случаях работает сортировка вставкой.

- Почти отсортированный массив



- Отсортированный в обратном порядке массив



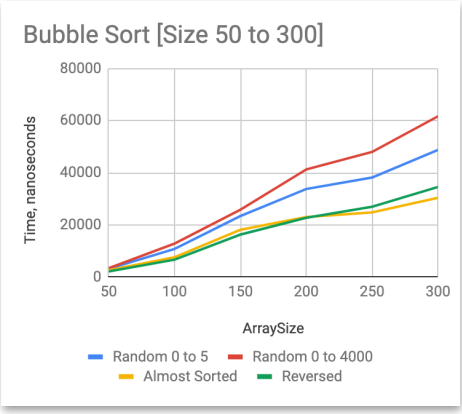


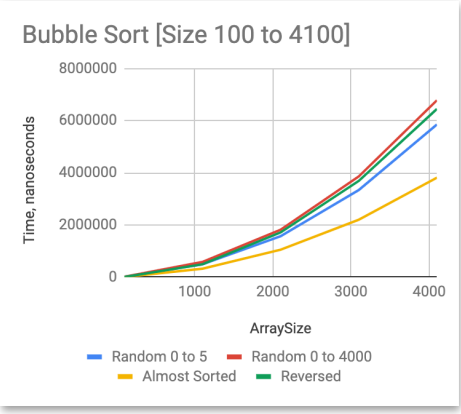
На этих графиках мы можем наблюдать такую же ситуацию с сортировкой слиянием, как и ранее. Также бросается в глаза временная шкала на графике для отсортированных в обратном порядке массивов больших размерностей: она сбилась, поскольку сортировка бинарной вставкой на таком массиве работает слишком долго, так как это «худший» случай для применения сортировки вставкой, здесь она работает за $O(n^2)$. Значение в этом эксперименте достигло 14 706 857 наносекунд.

2. Графики по Видам Сортировок

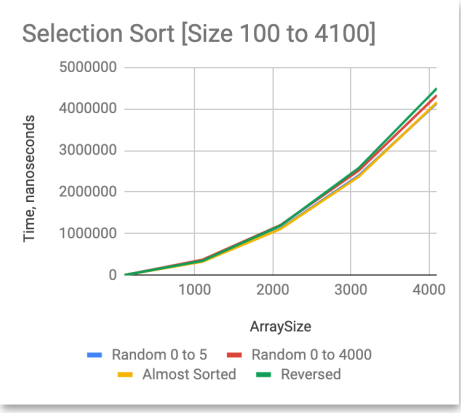
Итерационные:

- Сортировка пузырьком (обычная)

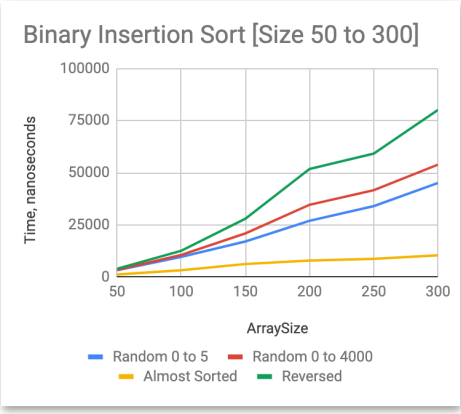


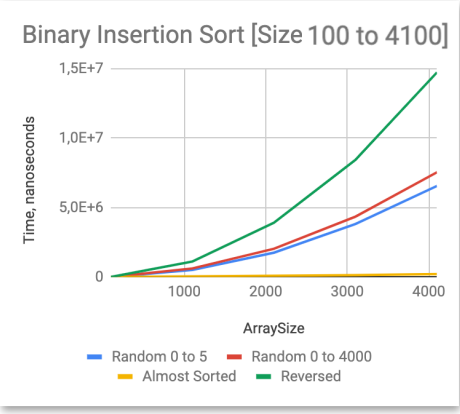


- Сортировка выбором минимума



- Сортировка бинарной вставкой

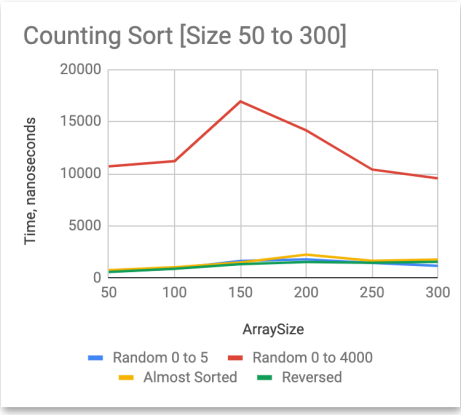


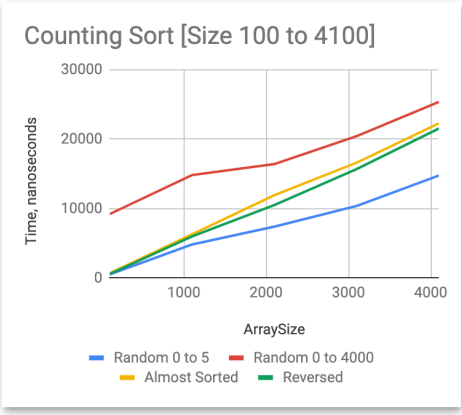


На графиках выше заметно, что в основном итеративные сортировки медленнее работают на случайно сгенерированных массивах. Отличается только сортировка бинарной вставки: как описано выше, эта сортировка медленно работает на отсортированных в обратном порядке массиве. Также на последних двух графиках можно заметить, что сортировка вставкой быстро работает на почти отсортированных массивах. Это происходит, потому что количество инверсий маленькое.

Линейные:

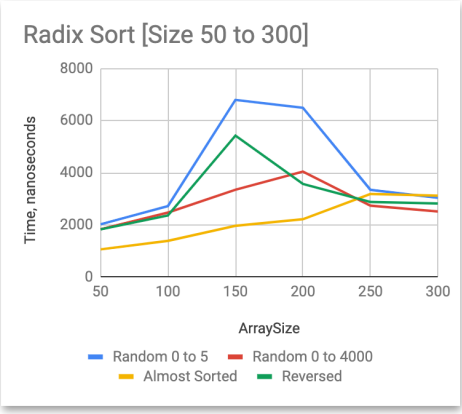
- Сортировка подсчетом (устойчивая)

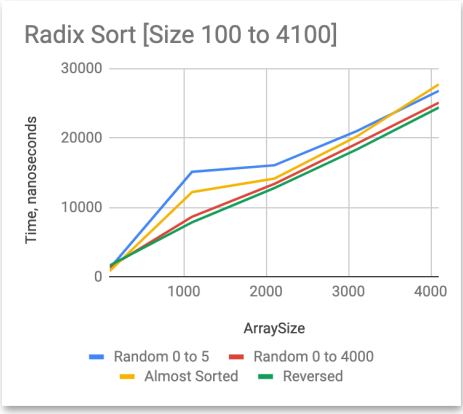




Сразу бросается в глаза отставание по времени массивов случайных чисел в диапазоне от 0 до 4000. Это объясняется тем, что при сортировке подсчетом создается вспомогательный массив размером `k = max - min`, где `max` и `min` — наибольшее и наименьшее числа основного массива соответственно, а позже необходимо пройтись по всему этому массиву. На больших числах размер вспомогательного массива велик, что и замедляет работу программы.

- Цифровая сортировка (256ричная сс)



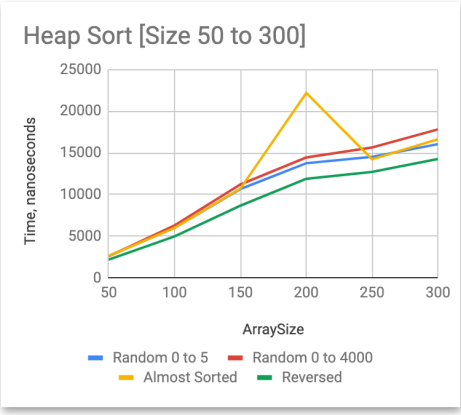


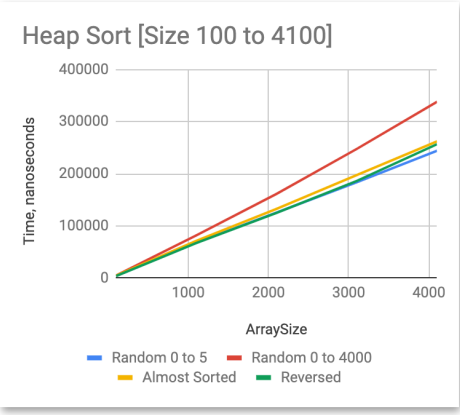
На этих графиках мы видим, что цифровая сортировка медленнее всего работает на маленьких числах.

(?)

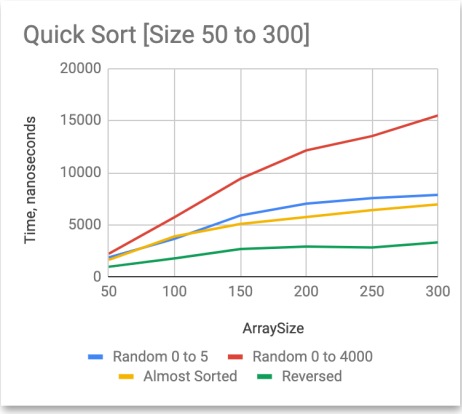
Рекурсивные:

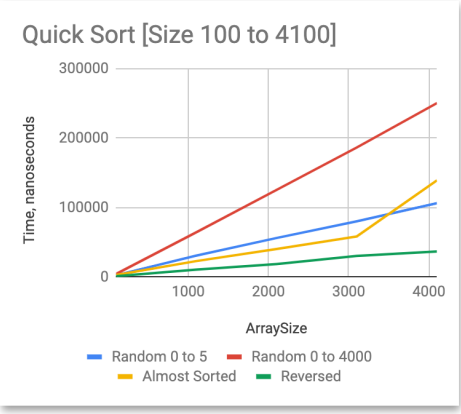
- Сортировка кучей



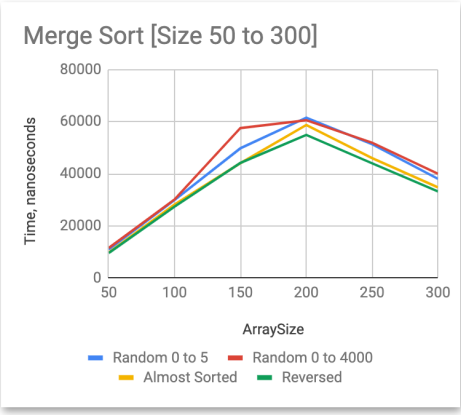


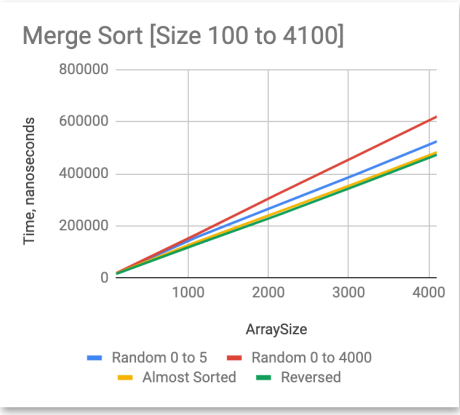
- Быстрая сортировка (опорный - средний)





- Сортировка слиянием





На графиках к рекурсивным сортировкам можно заметить, что они медленнее работают на больших значениях и быстрее всего работают на отсортированных в обратном порядке числах.

Помимо описанных выше наблюдений, взяв во внимание все графики, можно заметить, что наблюдаются странные скачки на размерностях 150-200. Предположительно, во время тестирования массивы в этих размерностях могли сгенерироваться с некоторым перекосом.