Анализ работы алгоритмов сортировки массивов в C/C++

Автор: Сарайкин Георгий РЛ6-21

Основной частью программы является цикл for, который вызывает функцию-враппер, заполняющую массив случайными значениями, вызывающую функцию сортировки, замеряющую время её работы и записывающую результаты в текстовый файл, который потом передаётся в программу Gnuplot. В целях оптимизации массив максимально возможного размера создаётся до работы цикла и удаляется после окончания его работы. Для более точных результатов сортировка проводится несколько раз (для представленных ниже графиков — 30), а в файл записывается среднее арифметическое длительностей каждой сортировки.

```
int* a = (int*)malloc(sizeof(int) * MAX_STEPS * GLOBAL_SCALE_FACTOR * max(130 * NUM_OF_THREADS, 5600));

for (int i = 1; i < MAX_STEPS; i++)
{
    sort_wrap(a, i, bubsort, 80 * GLOBAL_SCALE_FACTOR, bubblesort, 0);
    sort_wrap(a, i, qsort, 2800 * GLOBAL_SCALE_FACTOR, quicksort, 1);
    sort_wrap(a, i, inssort, 140 * GLOBAL_SCALE_FACTOR, insertion_sort, 2);
    sort_wrap(a, i, selsort, 100 * GLOBAL_SCALE_FACTOR, selection_sort, 3);
    sort_wrap(a, i, mergesort, 1400 * GLOBAL_SCALE_FACTOR, merge_sort, 4);
    sort_wrap(a, i, flashsort, 5600 * GLOBAL_SCALE_FACTOR, flash_sort, 5);
    sort_wrap(a, i, bubsort_swap_check, 80 * GLOBAL_SCALE_FACTOR, bubblesort_swap_check, 6);

    sort_wrap(a, i, inssort_mt, 150 * NUM_OF_THREADS * GLOBAL_SCALE_FACTOR, insertion_sort_multithread, 7);

    cout << "Sorting arrays, " << i << " of " << MAX_STEPS <<"...\n";
}

delete_a(a);
```

Пояснения:

1. Создаётся массив максимально возможного размера.

2. Вызываются функции сортировок.

3. В консоль выводится информация о том, сколько шагов программа сделала.

4. Массив удаляется.

Множители размера массива подбирались так, чтобы максимальное время работы каждой сортировки было примерно одинаковым.

Функция-враппер:

```
void sort_wrap(int* a, int len, ofstream& file_writer, int scale_factor, void (*sort)(int*, int, int), int sort_index)
{
    Uint64 av_time = 0;

    file_writer << len * scale_factor << "\t";

    for (int i = 0; i < NUMBER_OF_ITERATIONS; ++i)
    {
        generate_array(a, len * scale_factor);

        //flash_sort(a, len * scale_factor - 1, 0); //for measuring time it takes to sort an already sorted array

        auto begin = chrono::high_resolution_clock::now();

        sort(a, len * scale_factor - 1, 0);

        auto end = chrono::high_resolution_clock::now();

        if(i == 1)
        {
            int result = check_array(a, len * scale_factor);

            if (result == -1)
            {
                cout << "\n\nRuntime error: " << sorts[sort_index] << " failed.\n\n" << endl;
                system("pause");
                exit(1);
            }
        }

        av_time += chrono::duration_cast<std::chrono::nanoseconds>(end - begin).count();
    }

    av_time /= NUMBER_OF_ITERATIONS * 1000;

    file_writer << av_time << "\n";
}
```

Пояснения:

1. Пишем в файл размер массива.

2. В цикле: заполняем массив случайными значениями, запоминаем время начала, вызываем функцию сортировки, запоминаем время окончания. Для замера времени используется библиотека chrono, поскольку она имеет большую точность, чем GetTickCount(). На одной из итераций проверяем, отсортирован ли массив, и останавливаем программу, если сортировка отработала неправильно.

3. Вычисляем среднее время работы и переводим его из нс в мкс.

4. Пишем в файл время работы.

Результаты работы программы

График всех сортировок:

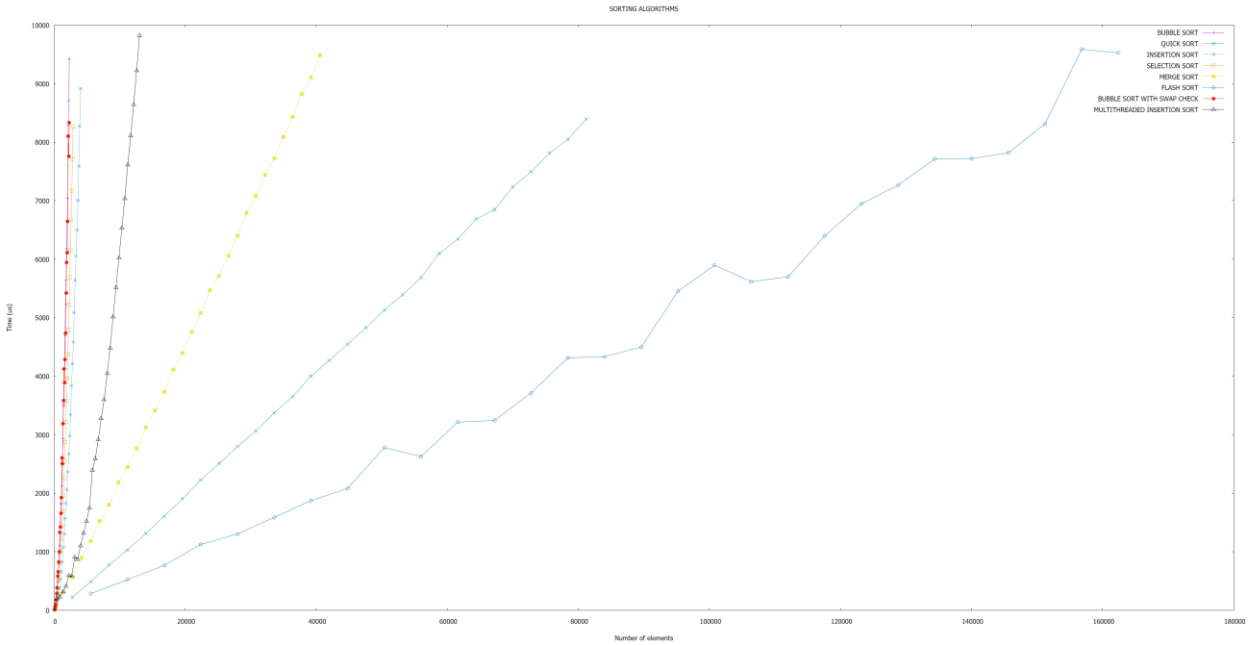
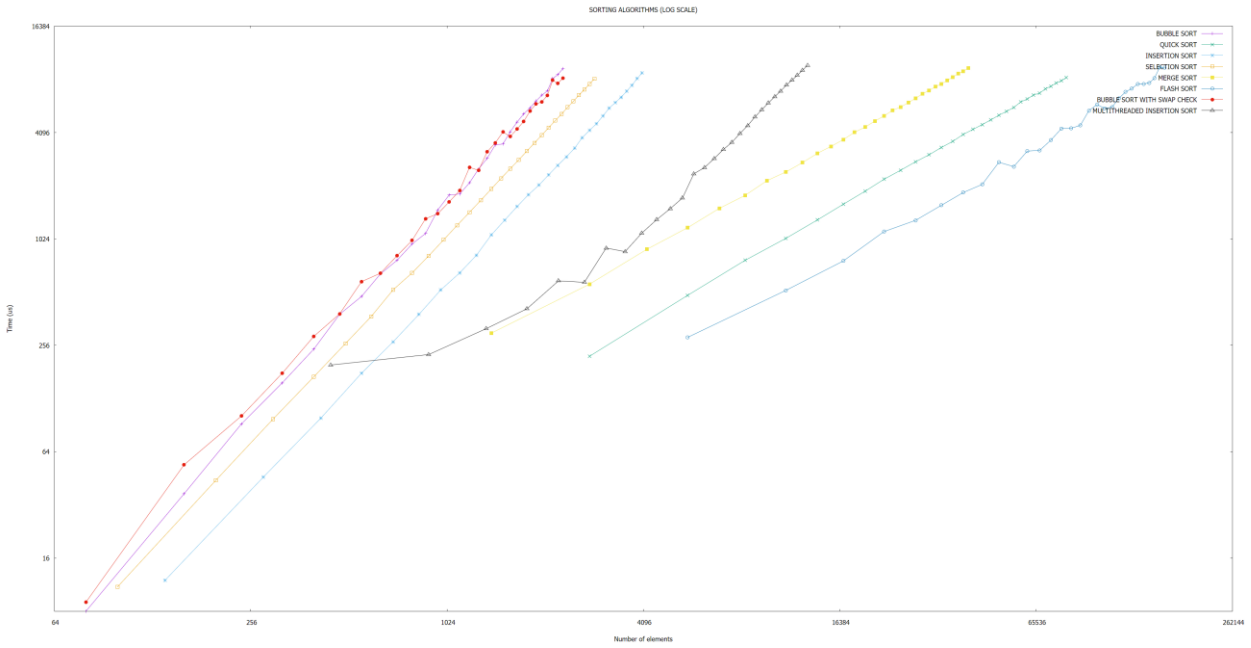
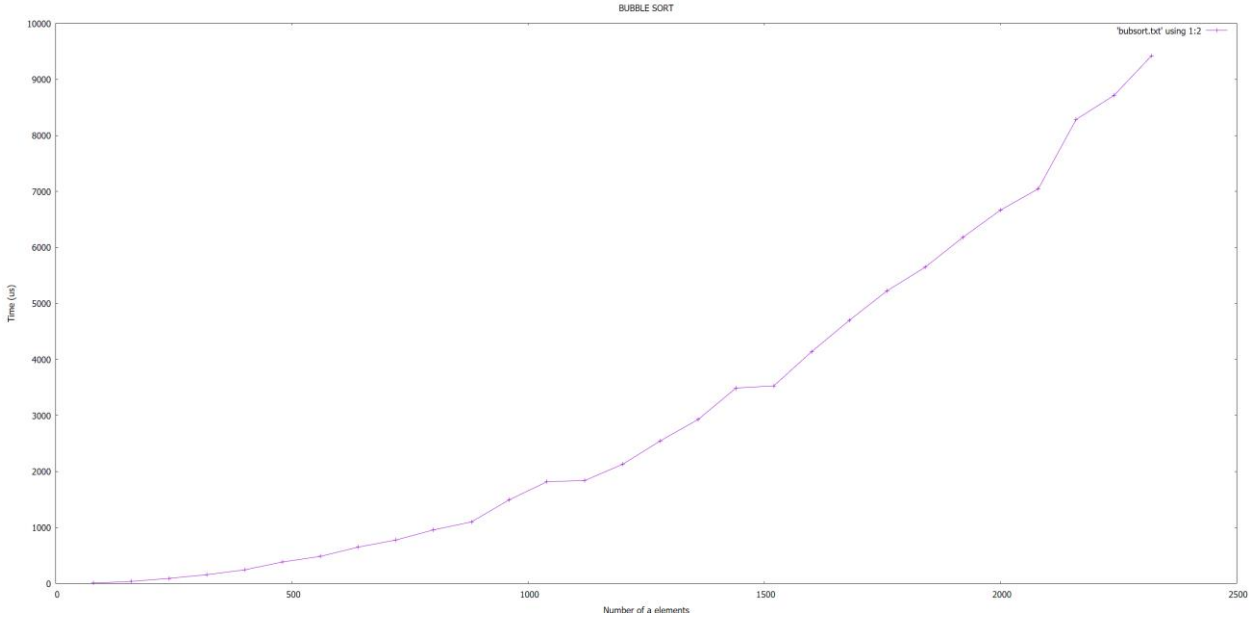


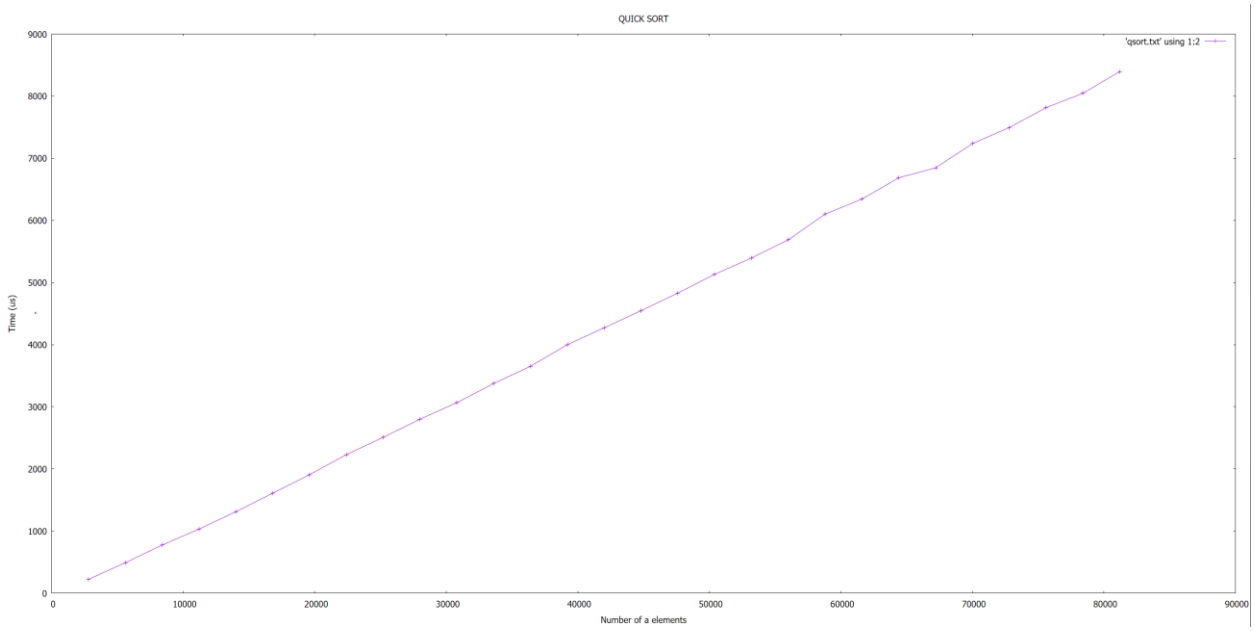
График всех сортировок (логарифмическая шкала):



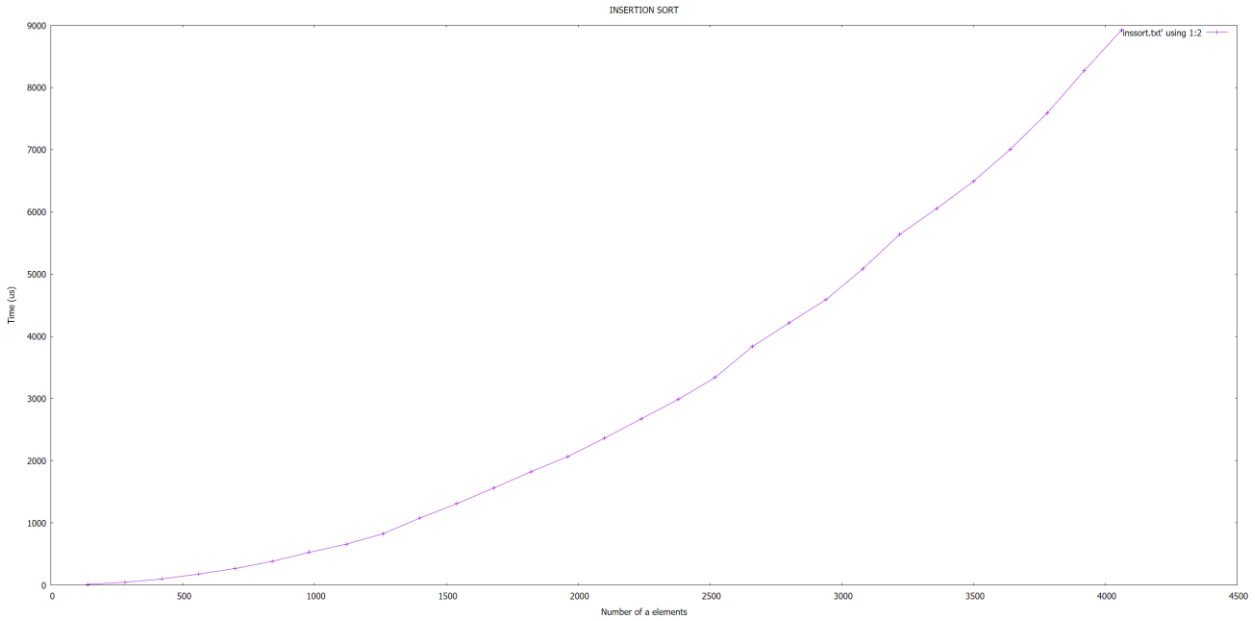
Сортировка пузырьком:



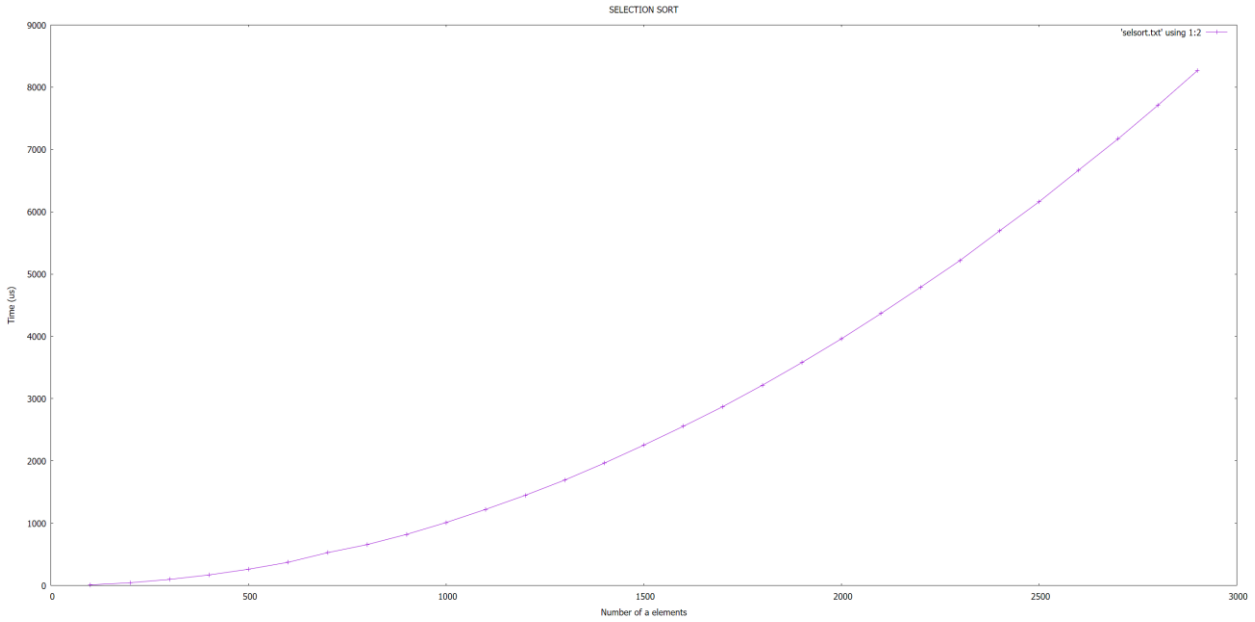
Быстрая сортировка:



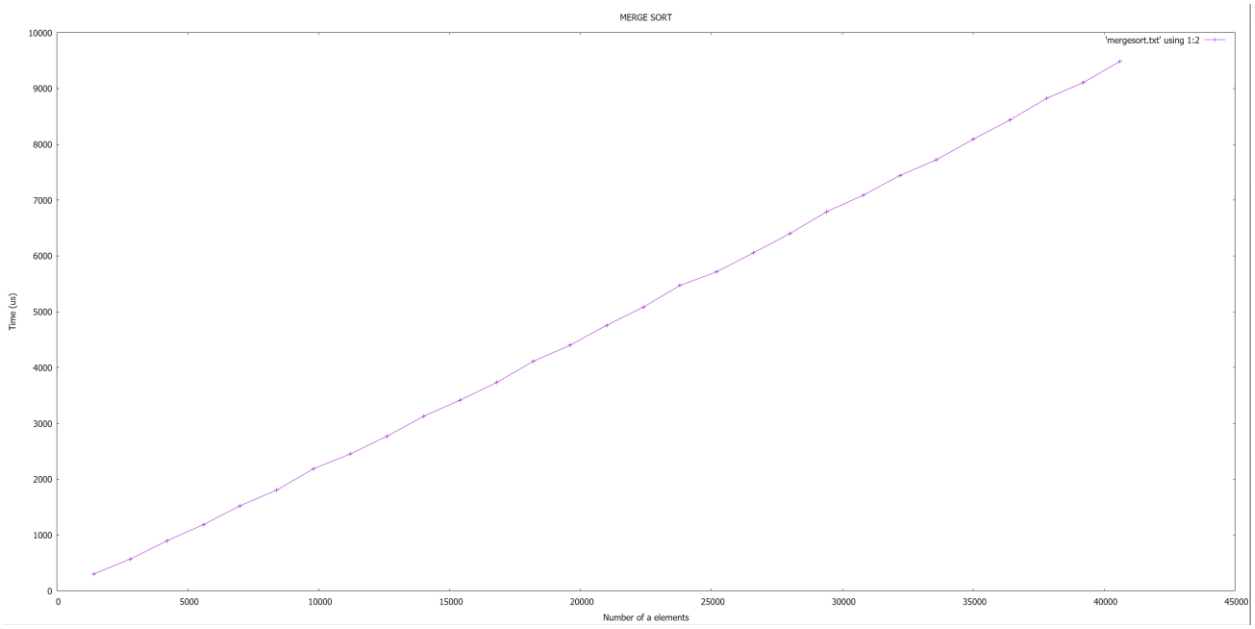
Сортировка вставками:



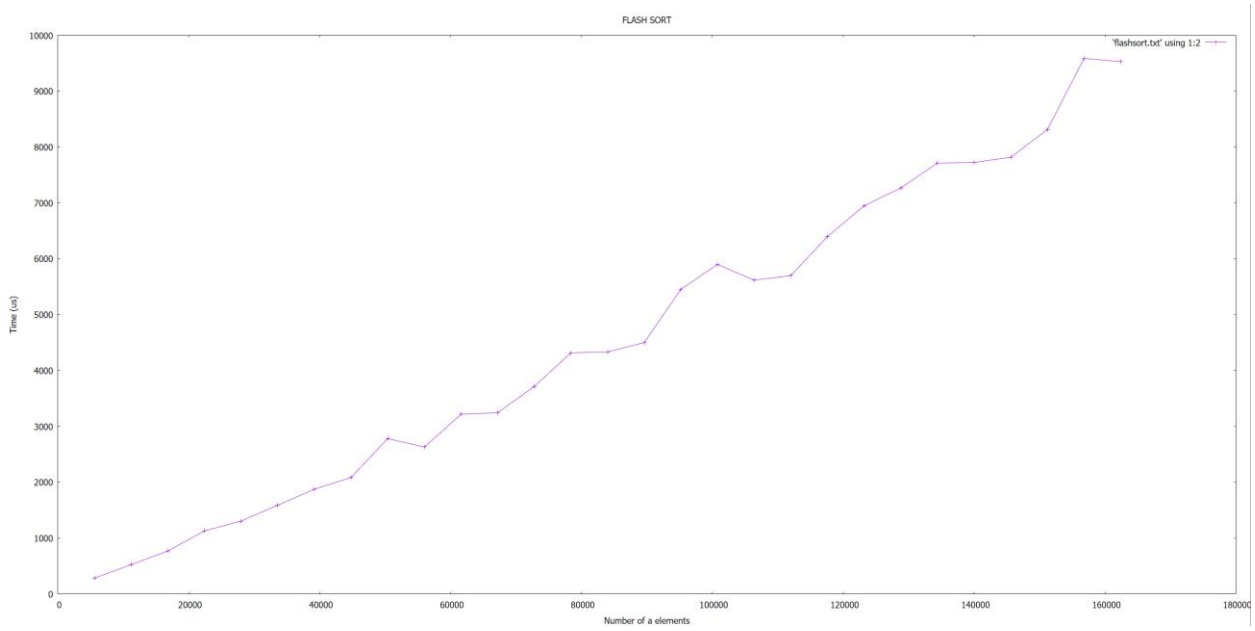
Сортировка выбором:



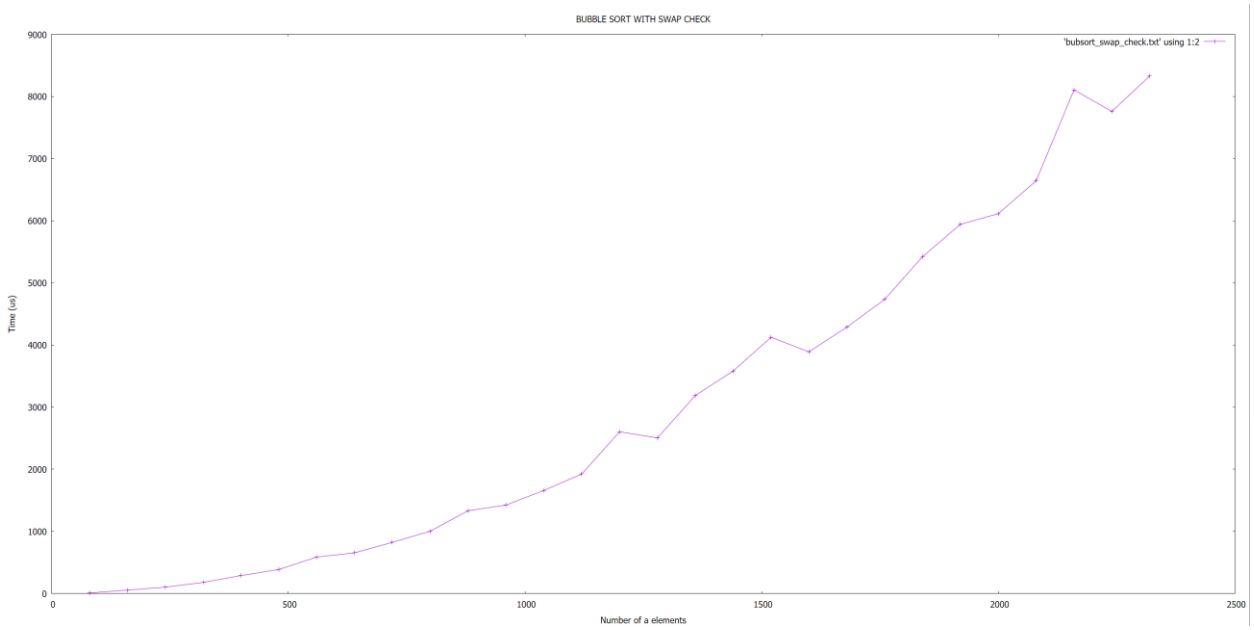
Сортировка слиянием:



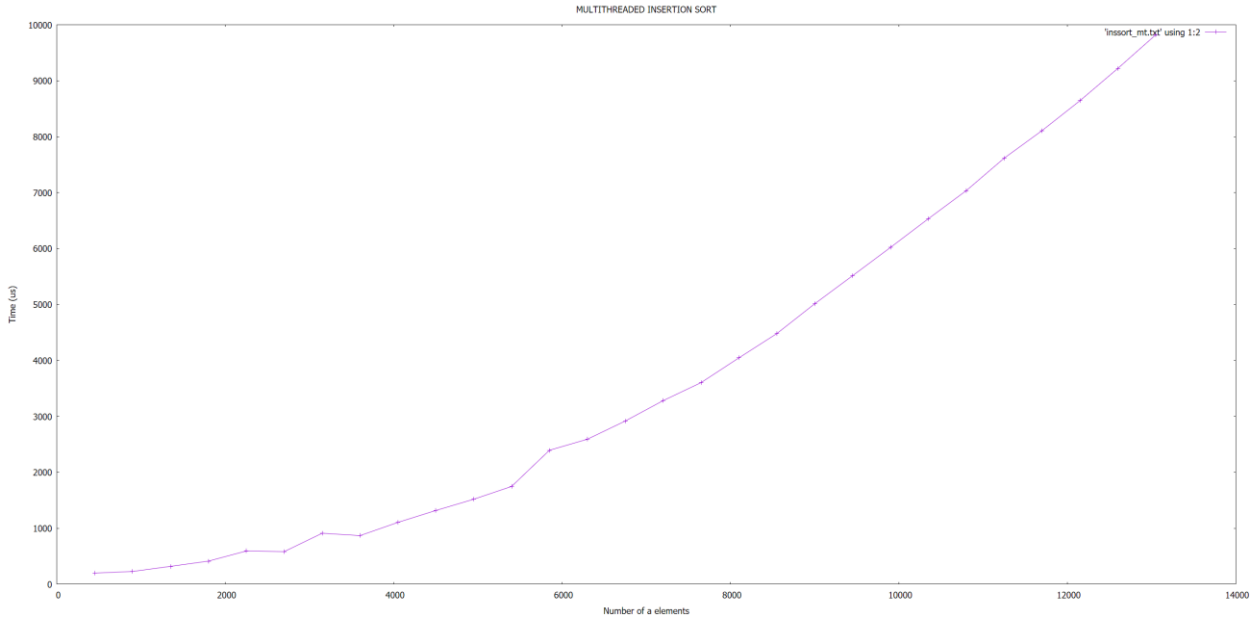
Flash sort:



Пузырьковая сортировка с проверкой того, была ли перестановка:



Многопоточная сортировка вставками (3 потока):



Анализ результатов

На малых размерах массивов невозможно определить, какая сортировка быстрее, потому что, помимо программы, на том же ядре процессора выполняются и другие потоки. Если же брать относительно большие массивы, то распределение скоростей (от самой медленной к самой быстрой) будет примерно таким:

1. Пузырьковая

2. Пузырьковая с проверкой того, была ли перестановка

3. Выбором

4. Вставками

5. Многопоточная вставками

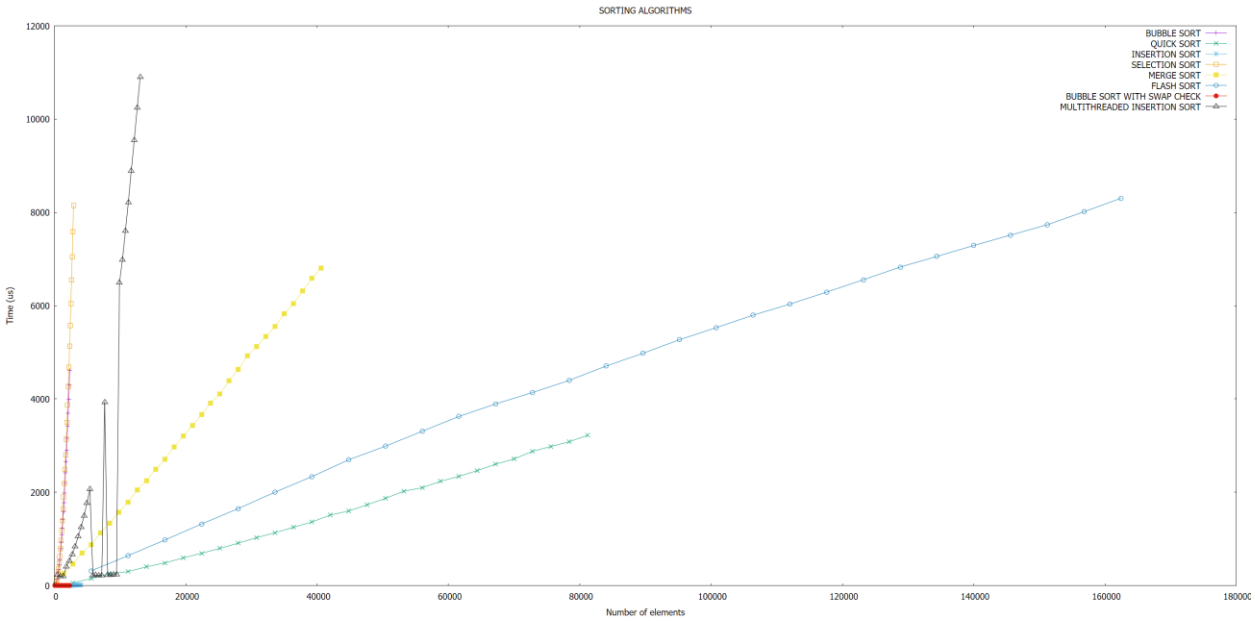
6. Слиянием

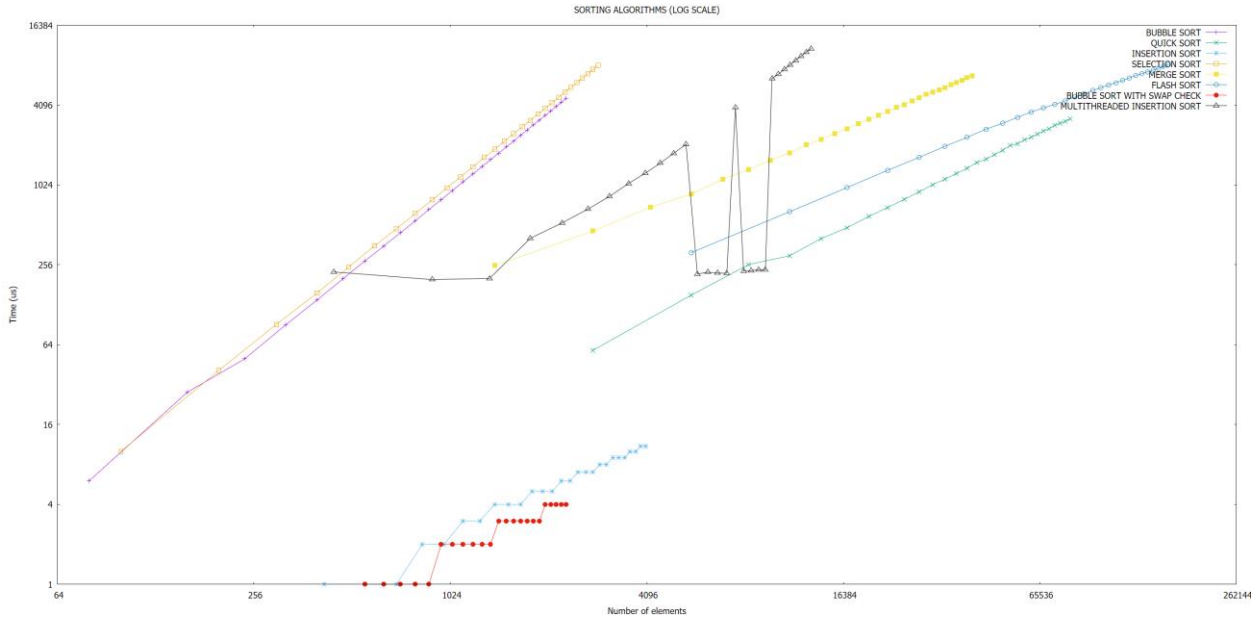
7. Быстрая

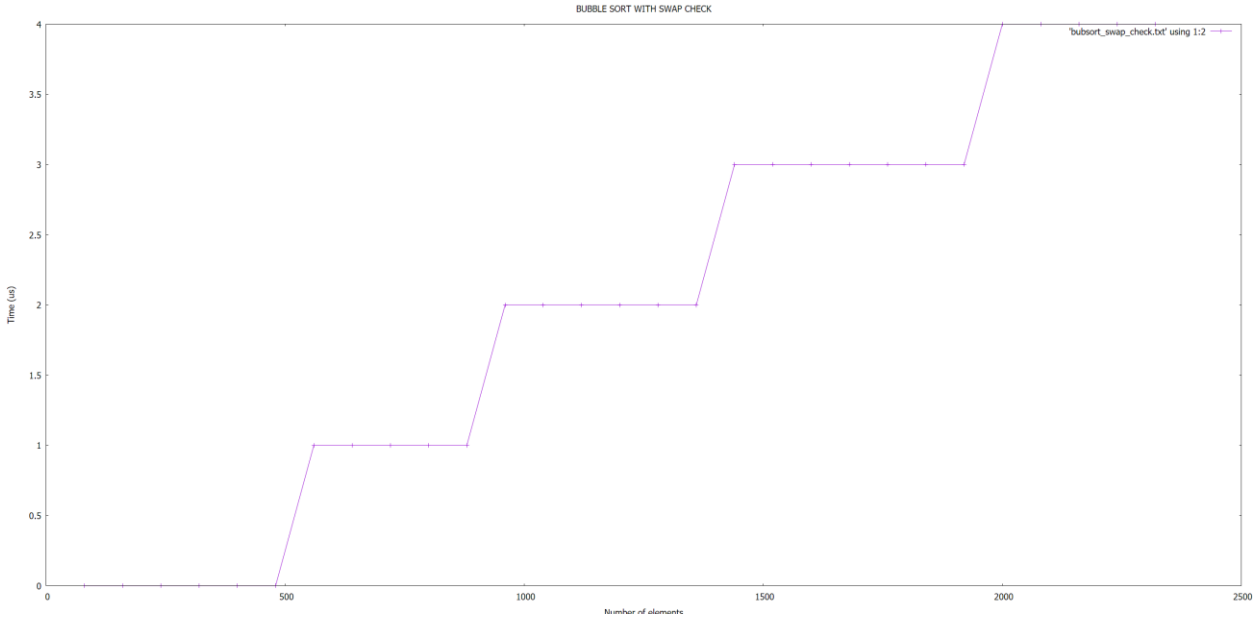
8. Flash sort

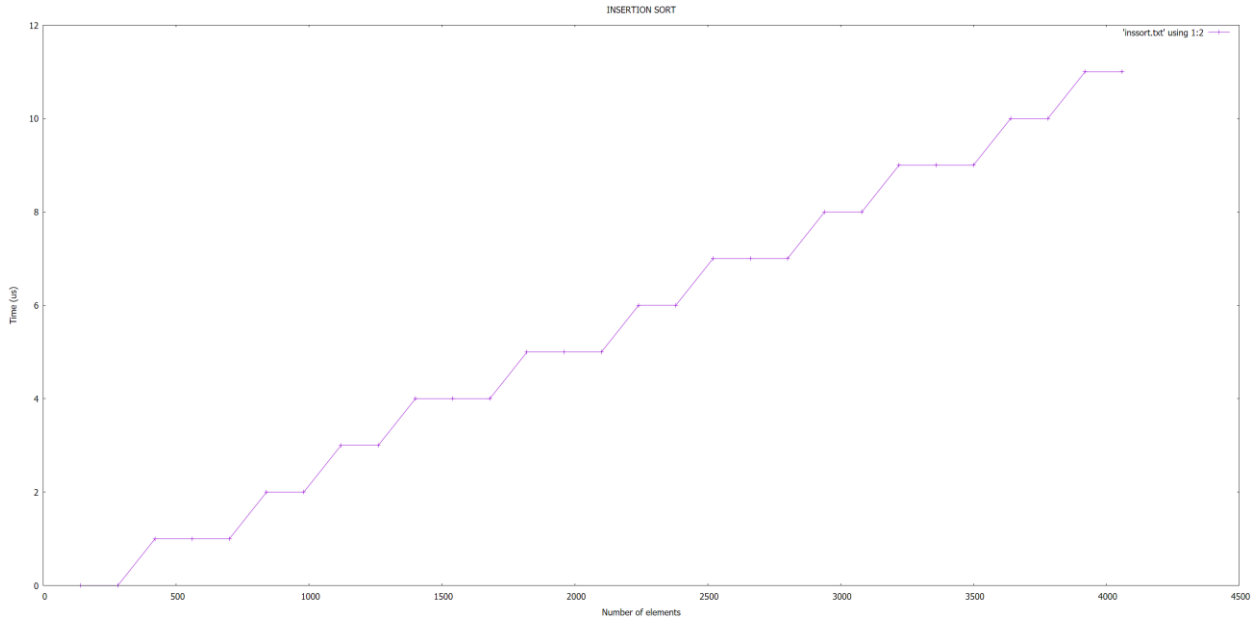
Дополнительно:

Графики времени работы на уже отсортированном массиве:









Т.е. сортировка вставками и пузырьковая с проверкой перестановки ввиду свойств алгоритма очень быстро обрабатывают уже отсортированный массив по сравнению со временем, которое они затратили бы на сортировку массива, заполненного случайными значениями.