Исследование асимптотики алгоритмов сортировки

Фахруидинов Алексей

Апрель 2021

1 Цель работы

Исследовать экспериментальным путем асимптотику алгоритмов сортировки, графически представить полученные результаты.

2 Описание алгоритмов

Реализовано 4 алгоритма сортировки: сортировки пузырьком, выбором, вставками и сортировка Шелла. Для сортировки Шелла использовалась заранее вычисленная последовательность чисел(характеризующая алгоритм) вида $2^i \cdot 3^j$.

3 Сбор экспериментальных данных

```
void test_sorts (void(*bubble_sort)(int*, const int), void(*selection_sort)(int*, const int),
                 void(*insertion_sort)(int*, const int), void(*shell_sort)(int*, const int)) {
    std::mt19937 engine(23);
    std::uniform_int_distribution<int> int_dist(0, 10000);

    for (int i = 0; i < count_of_lenth; ++i) {
        int cur_length = 400 + i*10;
        for (int j = 0; j < count_of_test; ++j) {
            int* b = new int[cur_length];
            int* sel = new int[cur_length];
            int* ins = new int[cur_length];
            int* sh = new int[cur_length];
            for (int k = 0; k < cur_length; ++k){
                int num = int_dist(engine);
                b[k] = num;
                sel[k] = num;
                ins[k] = num;
                sh[k] = num;
            }
            auto start = std::chrono::high_resolution_clock::now();
            shell_sort(sh, cur_length);
            auto end = std::chrono::high_resolution_clock::now();
            auto nsec = end - start;
            sh_sort[i][j] = nsec.count() ;

            start = std::chrono::high_resolution_clock::now();
            selection_sort(sel, cur_length);
            end = std::chrono::high_resolution_clock::now();
            nsec = end - start;
            sel_sort[i][j] = nsec.count();

            start = std::chrono::high_resolution_clock::now();
            insertion_sort(ins, cur_length);
            end = std::chrono::high_resolution_clock::now();
            nsec = end - start;
            ins_sort[i][j] = nsec.count();

            start = std::chrono::high_resolution_clock::now();
            bubble_sort(b, cur_length);
            end = std::chrono::high_resolution_clock::now();
            nsec = end - start;
            b_sort[i][j] = nsec.count();

            delete[] b;
            delete[] sel;
            delete[] ins;
            delete[] sh;
        }
    }
}
```

Выше приведена функция, реализующая замер времени сортировки для разных длин и повторений на одной длине. Следующим образом определены необходимые константы и массивы обрабатываемых данных.

```
const int count_of_test = 100, count_of_lenth = 200;

const int shell[100] {1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, 48, 54, 64, 72, 81, 96, 108, 128, 144, 162, 192,
                      216, 243, 256, 288, 324, 384, 432, 486, 512, 576, 648, 729, 768, 864, 972, 1152, 1296, 1458, 1536,
                      1728, 1944, 2187, 2304, 2592, 2916, 3456, 3888, 4374, 4608, 5184, 5832, 6561, 6912, 7776, 8748,
                      10368, 11664, 13122, 13824, 15552, 17496, 19683, 20736, 23328, 26244, 31104, 34992, 39366, 41472,
                      46656, 52488, 62208, 69984, 78732, 93312, 104976, 124416, 139968, 157464, 186624, 209952, 279936,
                      314928, 373248, 419904, 559872, 629856, 839808, 1119744, 1259712, 1679616, 2519424, 3359232,
                      5038848, 10077696};

double b_sort[count_of_lenth][count_of_test], sel_sort[count_of_lenth][count_of_test],
    ins_sort[count_of_lenth][count_of_test], sh_sort[count_of_lenth][count_of_test];
```

В результате мы получаем усредненное время сортировки для каждой длины массива(результаты в репозитории).

4 Обработка результатов

Для обработки использовался программный пакет OriginLab. Аппроксимируя наши зависимости функциями $c \cdot n^2$ для медленных сортировок и $a \cdot x \cdot (log(b \cdot x))^2$ для сортировки Шелла, получаем следующий график:

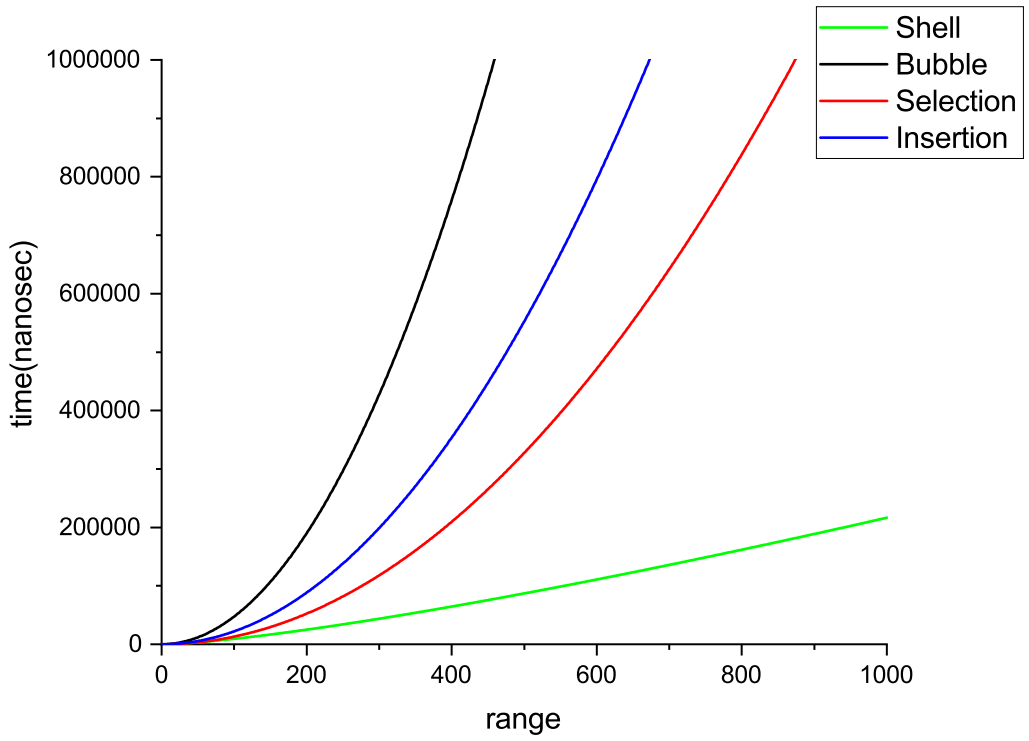


Рис. 1: Сравнение сортировок при маленьких длинах массива

То, насколько быстро растет время медленных сортировок в сравнении с сортировкой шелла, демонстрирует график:

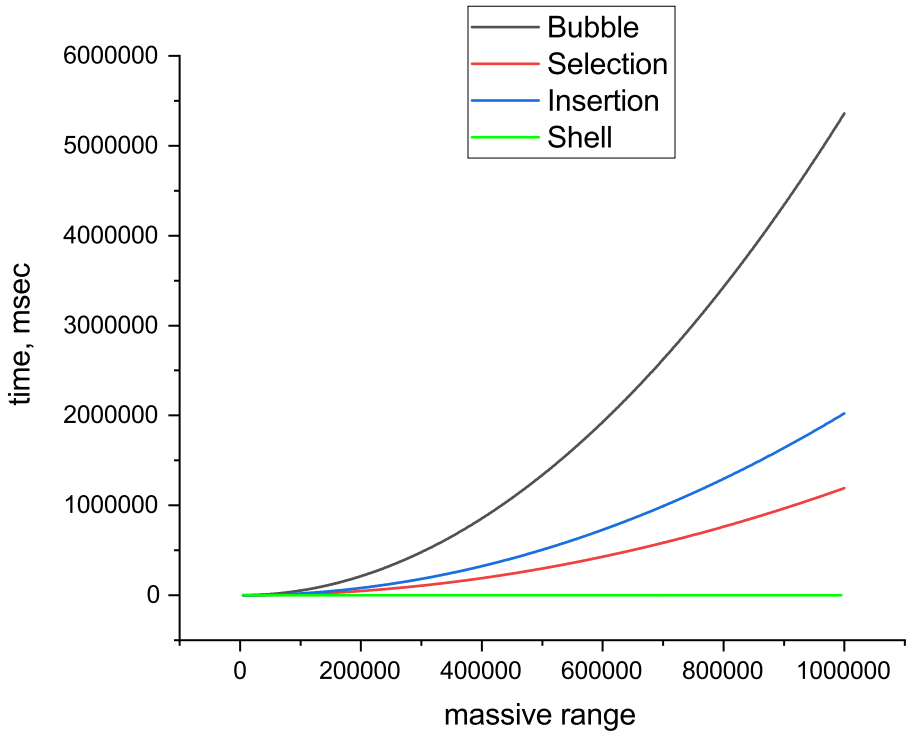
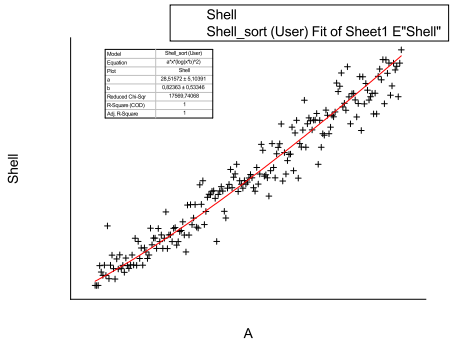
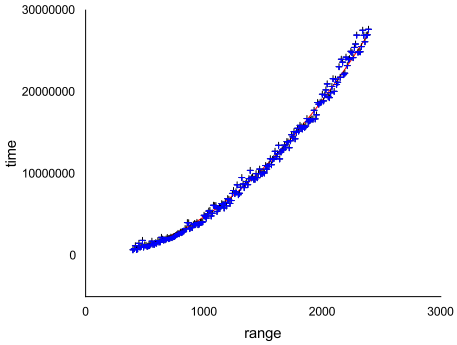


Рис. 2: Сравнение сортировок при больших длинах массива

5 Точность определения зависимостей

Поскольку при достаточно больших длинах массива медленные сортировки работают за довольно большое время, зависимость определялась по временам сортировок маленьких массивов. Была проведена "выборочная"проверка корректности определения зависимости следующим образом: измерено время сортировки массива для длин в интервале от 10^4 до 10^6 чисел, они достаточно точно укладываются в полученные зависимости. Как зависимость "покрывает"экспериментальные данные иллюстрируют данные графики:





6 Вывод

Полученные зависимости отвечают теоретическим оценкам сложностей алгоритмов. Страница github с исходным кодом