Отчет о выполнении задания 1  
«Алгоритмы сортировки»

**Мингазеев Роман Лутфиевич**

Группа ИВ-521

Direnol@yandex.ru

В задании 1 требовалось реализовать и исследовать эффективность трех алгоритмов сортировки: Counting Sort, Bubble Sort и Quick Sort. Все алгоритмы реализованы на языке C под операционной системой GNU/Linux.

# Описание алгоритмов

Алгоритм Counting Sort

Алгоритм Counting Sort (сортировка подсчетом) — алгоритм сортировки, в котором используется диапазон чисел сортируемого массива (списка) для подсчёта совпадающих элементов.

В работе реализована версия алгоритма с вычислительной сложностью в худшем случае *O*(*n* + k), где *n* — это количество ключей входного массива , а *k* — это количество ключей вспомогательного массива nmin ≤ k ≤ nmax. Сложность по памяти *O*(*n* + *k*). Обладает свойствами : устойчивость(реализованный), не на месте.

Псевдокод устойчивой версии алгоритма CountingSort:

**function CountingSort (A, n, k)**

**for i = 0 to k — 1 do**

**C[i] = 0;**

**for i = 0 to n — 1 do**

**C[A[i]] = C[A[i]] + 1;**

**for j = 1 to k — 1 do**

**C[j] = C[j] + C[j - 1];**

**for i = n - 1 to 0 do**

**C[A[i]] = C[A[i]] - 1;**

**B[C[A[i]]] = A[i];**

**end function**

Алгоритм Bubble Sort

Алгоритм Bubble Sort (“Пузырьковая” сортировка) - Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются *n* - 1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде, отсюда и название алгоритма).

Вычислительная сложность алгоритма в худшем случае — *O*(*n*2), где *n* — это количество ключей входного массива. Обладает свойствами : устойчивость, на месте.

Сложность по памяти — *O*(1).

Псевдокод алгоритма BubbleSort:

**function BubbleSort (v[0:n - 1], n)**

**swapped = True**

**while swapped do**

**swapped = False**

**for i = 1 to n - 1 do**

**if v[i - 1] > v[i] then**

**swap(v[i - 1], v[i])**

**swapped = True**

**end if**

**end for**

**end while**

**end function**

Алгоритм Quick Sort

Быстрая сортировка (quicksort) является одним из самых распространенных на практике алгоритмов внутренней сортировки. В среднем случае он работает за время *Θ*(? log ?) и требует порядка Θ(log ?) ячеек памяти на поддержание рекурсивных вызовов. Однако алгоритм не обладает свойством устойчивости и в худшем случае требует выполнения порядка *Θ*(?2) операций. Быстрая сортировка относится к рекурсивным алгоритмам сортировки сравнением и основана на методе декомпозиции (decomposition).

Обладает свойствами : неустойчивость.

Псевдокод алгоритма QuickSort:

**function QuickSort (massiv, begin, end)**

**pivot = (massiv[low] + massiv[high]) / 2**

**low = begin**

**high = end**

**repeat**

**while (massiv[low] < pivot)**

**low = low + 1**

**while (massiv[high]) > pivot)**

**high = high - 1**

**if (low <= high)**

**swap (massiv[low], massiv[high])**

**low = low +1**

**high = high -1**

**until (low <= high)**

**if (begin < high)**

**QuickSort (massiv, begin, high)**

**if (low < end)**

**QuickSort (massiv, begin, high)**

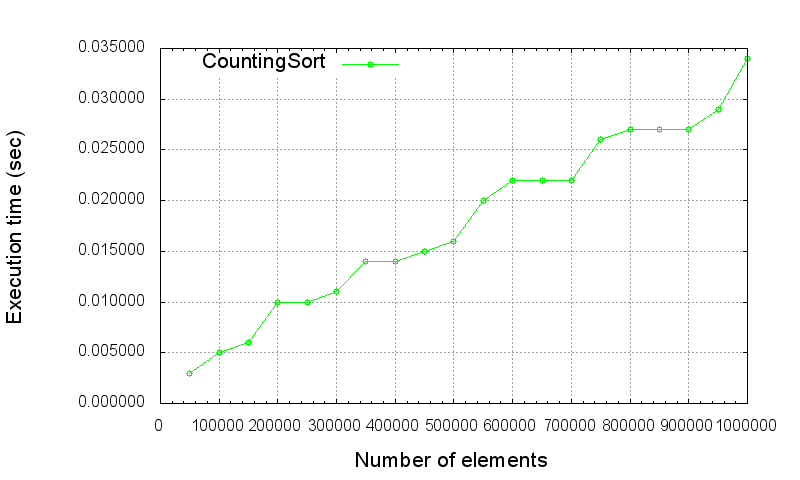
**end function**

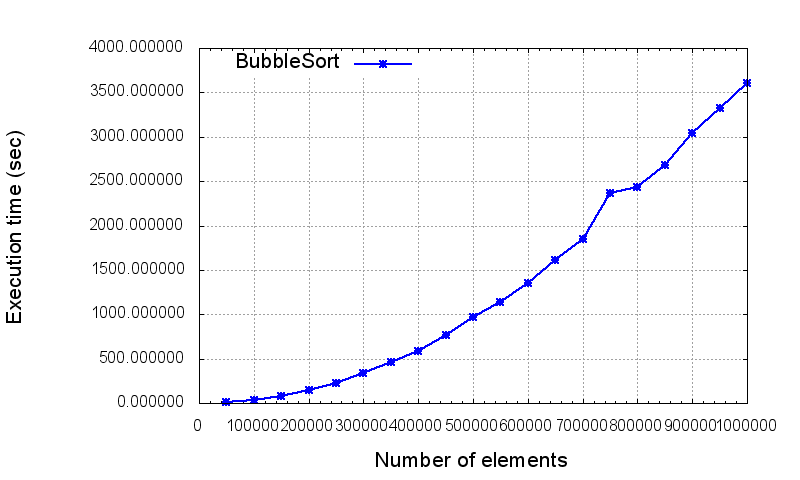
# Организация экспериментов

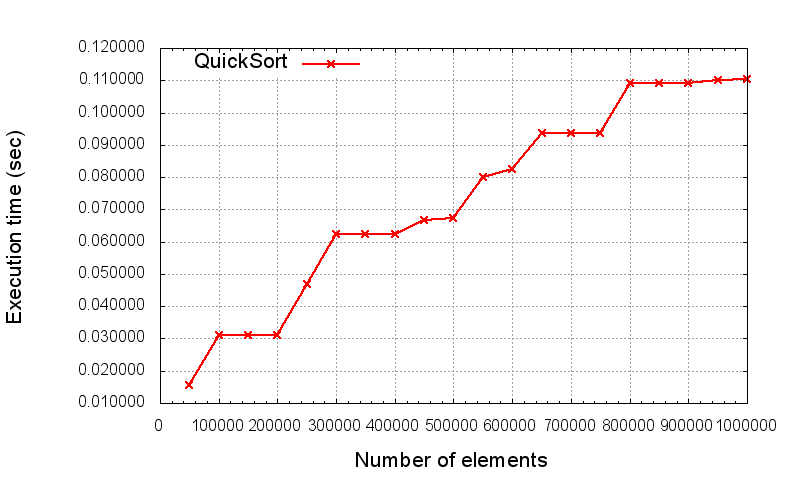
* Эксперименты проводились на ПК DNS   
  (CPU: QuadCore Intel Core i5-3550, 3500 MHz (35 x 100), RAM: 8GB, Hitachi HDS721010CLA330 ATA Device (931 ГБ, IDE))
* Операционная система Ubuntu 15.10 x64 (компилятор gcc 5.2.1)
* Ключи компиляции программы: -Wall -o

# Результаты экспериментов

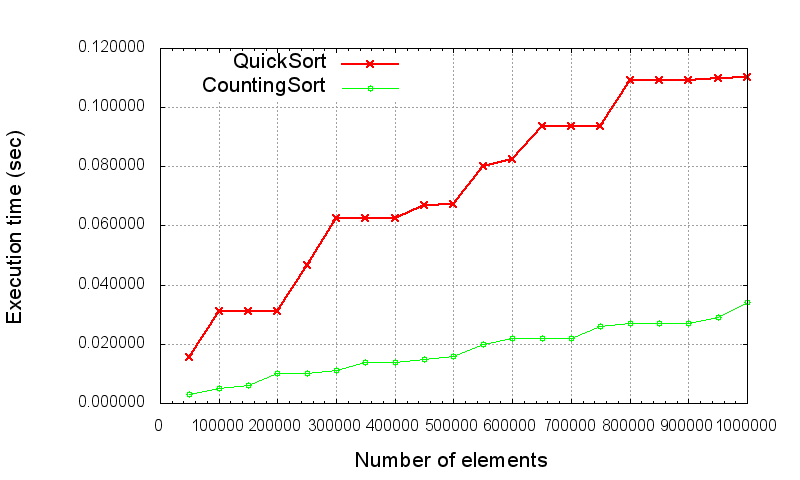
На рисунках ниже показаны графики зависимости времени выполнения алгоритмов от количества элементов в массиве.

*Рис. 1 Counting Sort*

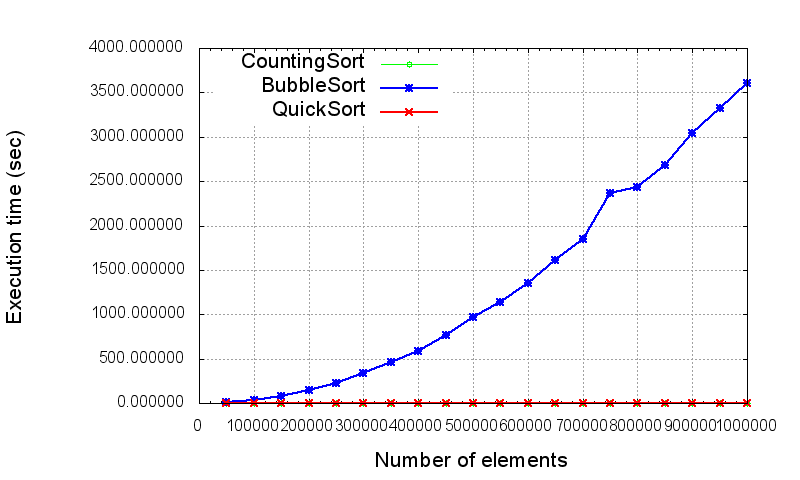


*Рис. 2 Bubble Sort*

*Рис. 3 Quick Sort*



*Рис. 4 Сравнение алгоритмов Quick Sort и Counting Sort*

*Рис. 5 Сравнение алгоритмов Quick Sort, Bubble Sort, Counting Sort*

**Вывод**

Из результатов экспериментов видно, что алгоритм Counting Sort работает быстрее алгоритмов Quick Sort и Bubble Sort. Это объясняется тем, что алгоритмы Quick Sort и Bubble Sort работают на основе сравнения элементов массива. Quick Sort работает быстрее алгоритма Bubble Sort — это объяснятеся тем, что вычислительная сложность Bubble Sort для выполнения всей сортировки должен массив из  *n* элементов *n* раз наибольший элемент переместить в конец массива. А Quick Sort разбивает массив из *n* элементов относительной опорного элемента на два меньших подмассива. Получается, что *T*bubbleSort(*n*) = *n*2 и *T*quickSort(*n*) = *n*2, но *T*quickSort(*n*) = *n*2 встречается редко, все зависит от выбора опорного элемента, поэтому для данных графиков верно, что *T*quickSort(*n*) = *n* log *n*. Из чего сдледует, что Quick Sort в большенстве случаев работает быстрее Bubble Sort.

*Таблица. Результаты экспериментов*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Количество элементов в массиве** | **Время выполнения алгоритма Counting Sort, с** | **Время выполнения алгоритма**  **Quick Sort, с** | **Время выполнения алгоритма**  **Bubble Sort, с** |
| 1 | 50 000 | 0.003001 | 0.15600 | 8.924216 |
| 2 | 100 000 | 0.005001 | 0.31200 | 35.887266 |
| 3 | 150 000 | 0.006000 | 0.031200 | 80.985415 |
| 4 | 200 000 | 0.010000 | 0.031200 | 147.322474 |
| 5 | 250 000 | 0.010001 | 0.046800 | 228.990098 |
| 6 | 300 000 | 0.011000 | 0.062400 | 339.886440 |
| 7 | 350 000 | 0.014001 | 0.062400 | 457.485768 |
| 8 | 400 000 | 0.014001 | 0.062400 | 589.844158 |
| 9 | 450 000 | 0.15000 | 0.066801 | 767.591940 |
| 10 | 500 000 | 0.016001 | 0.067400 | 968.373454 |
| 11 | 550 000 | 0.020001 | 0.080000 | 1132.960498 |
| 12 | 600 000 | 0.022001 | 0.082400 | 1352.867945 |
| 13 | 650 000 | 0.022000 | 0.093600 | 1608.581921 |
| 14 | 700 000 | 0.022002 | 0.093600 | 1845.808671 |
| 15 | 750 000 | 0.026001 | 0.093601 | 2368.154450 |
| 16 | 800 000 | 0.027001 | 0.109200 | 2434.932608 |
| 17 | 850 000 | 0.027001 | 0.109200 | 2681.308559 |
| 18 | 900 000 | 0.027001 | 0.109200 | 3046.220124 |
| 19 | 950 000 | 0.029001 | 0.110001 | 3318.313772 |
| 20 | 1 000 000 | 0.034002 | 0.110301 | 3600.631846 |

# Ссылки

# <https://ru.wikipedia.org/wiki/Сортировка_подсчетом>

1. <https://ru.wikipedia.org/wiki/Сортировка_пузырьком>
2. [https://ru.wikipedia.org/wiki/Быстрая\_Сортировка](https://ru.wikipedia.org/wiki/Быстрая_сортировка)