Лабораторная работа 2. Алгоритмы сортировки.

Постановка задачи

Реализовать алгоритмы сортировок в соответствии с вариантом (таблица 1), провести экспериментальное исследование скорости работы алгоритмов, построить график для демонстрации результатов.

Вариант определяется по номеру в таблице:

```
(номер_в_таблице - 1) % 24 + 1
```

* Дополнительно реализуйте еще одну сортировку из столбца 3 (линейно-логарифмические), находящуюся на 1 строку ниже сортировки по варианту.

Таблица 1. Распределение заданий по вариантам

Вариант	Не использующие операцию сравнения	Линейно-логарифмические	Квадратичные
1	Counting Sort	Quick Sort	Bubble Sort
2	Radix Sort	Merge Sort	Selection Sort
3	Counting Sort	Heap Sort	Insertion Sort
4	Radix Sort	Quick Sort	Odd-Even Sort
5	Counting Sort	Heap Sort	Bubble Sort
6	Radix Sort	Merge Sort	Odd-Even Sort
7	Counting Sort	Quick Sort	Insertion Sort
8	Radix Sort	Heap Sort	Selection Sort
9	Counting Sort	Merge Sort	Bubble Sort
10	Radix Sort	Quick Sort	Bubble Sort
11	Counting Sort	Merge Sort	Selection Sort
12	Radix Sort	Heap Sort	Odd-Even Sort
13	Counting Sort	Quick Sort	Selection Sort
14	Radix Sort	Merge Sort	Bubble Sort
15	Counting Sort	Heap Sort	Selection Sort
16	Radix Sort	Quick Sort	Insertion Sort
17	Counting Sort	Merge Sort	Insertion Sort
18	Radix Sort	Heap Sort	Bubble Sort
19	Counting Sort	Merge Sort	Odd-Even Sort
20	Radix Sort	Quick Sort	Selection Sort

21	Counting Sort	Heap Sort	Odd-Even Sort
22	Radix Sort	Merge Sort	Insertion Sort
23	Counting Sort	Quick Sort	Odd-Even Sort
24	Radix Sort	Heap Sort	Insertion Sort

Экспериментальное исследование

- Необходимо измерить время работы каждого алгоритма при различном количестве элементов в массиве — заполните таблицу 2 для каждого алгоритма, постройте график по полученной таблице (пример на рисунке 1)
- Если один из алгоритмов работает значительно дольше других, используйте логарифмическую шкалу при построении графика или постройте дополнительно отдельный график без результата этой сортировки.
- По результатам экспериментов определите, какой алгоритм работает быстрее и почему
- В экспериментах используйте массивы с целочисленными элементами типа `uint32_t` (подключите заголовочный файл `inttypes.h`)
- Массивы заполняйте псевдослучайными числами с равномерным распределением из интервала [0, 100000]
- Память выделяется функцией *malloc*:

```
int* arr = (int*)malloc(sizeof(int) * n);
```

Таблица 2. Результаты экспериментов

#	Количество элементов в массиве	Время выполнения алгоритма 1, с	Время выполнения алгоритма 2, с	Время выполнения алгоритма 3, с
1	50000			
2	100000			
3	150000			
...	...			
20	1000000			

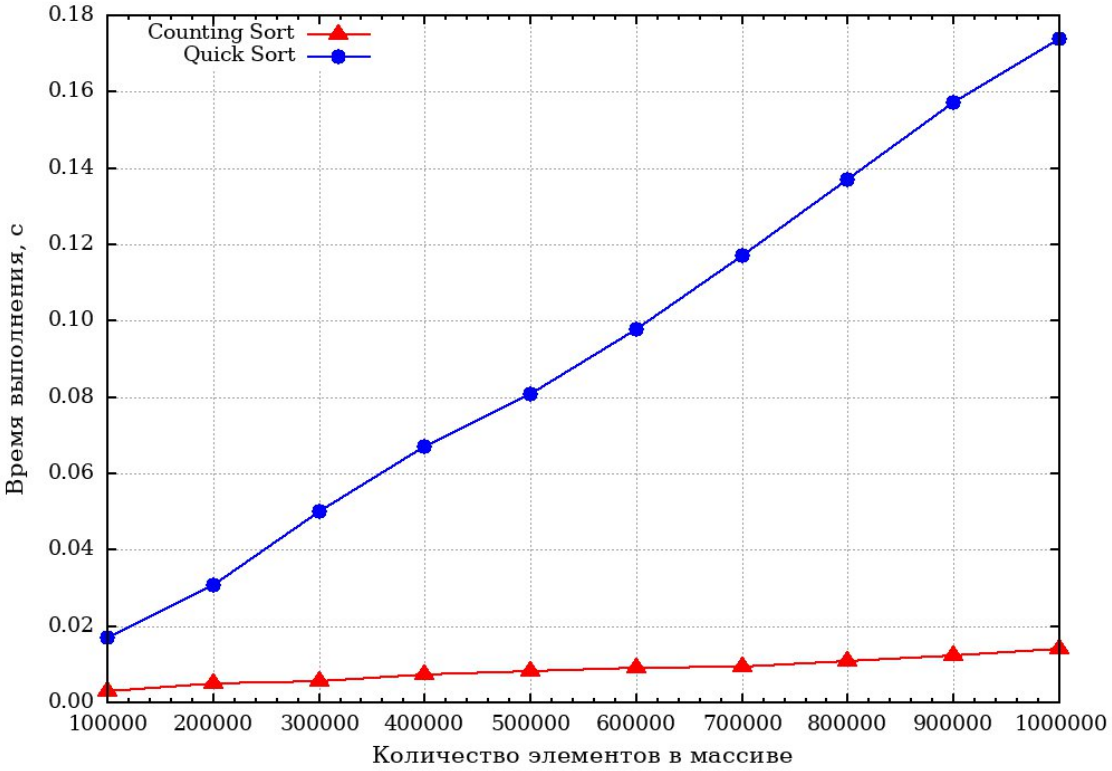

Рисунок 1. Зависимость времени выполнения сортировки подсчетом и быстрой сортировки от количества элементов в массиве

Контрольные вопросы

- Что означают записи $f(n) = O(g(n))$, $f(n) = \Theta(g(n))$, $f(n) = \Omega(g(n))$?
- Вычислительная сложность реализованных алгоритмов сортировок в среднем и худшем случае, причины возникновения худшего случая или его отсутствия.
- Пространственная сложность («сложность по памяти») реализованных алгоритмов сортировок.
- Что такое «устойчивая» сортировка?
- Что такое «сортировка на месте»?
- Свойства сортировок по варианту.
- Определение вычислительной сложности сортировки по исходному коду.
- Алгоритмы сортировок с вычислительной сложностью $n\log n$ для худшего случая.
- Алгоритмы сортировок, работающие быстрее $n\log n$ для худшего случая.
- Объяснение поведения кривых на графиках.