Задание 1. Алгоритмы сортировки

Постановка задачи

В соответствии со своим вариантом (номер в журнале успеваемости) необходимо реализовать 3 алгоритма сортировки и провести экспериментальное исследование их эффективности. Распределение алгоритмов по вариантам приведено в табл. 1.

Таблица 1. Распределение заданий по вариантам

Вариант	**Алгоритмы, не использующие операцию сравнения**	**Квадратичные алгоритмы сортировки**	**«Быстрые» алгоритмы сортировки**
1	Counting Sort	Bubble Sort	MergeSort
2	Radix Sort	Selection Sort	HeapSort
3	Counting Sort	Insertion Sort	QuickSort
4	Radix Sort	Odd-Even Sort	MergeSort
5	Counting Sort	Bubble Sort	HeapSort
6	Radix Sort	Selection Sort	QuickSort
7	Counting Sort	Insertion Sort	MergeSort
8	Radix Sort	Odd-Even Sort	HeapSort
9	Counting Sort	Bubble Sort	QuickSort
10	Radix Sort	Selection Sort	MergeSort
11	Counting Sort	Insertion Sort	HeapSort
12	Radix Sort	Odd-Even Sort	QuickSort
13	Counting Sort	Bubble Sort	MergeSort
14	Radix Sort	Selection Sort	HeapSort
15	Counting Sort	Insertion Sort	QuickSort
16	Radix Sort	Odd-Even Sort	MergeSort
17	Counting Sort	Bubble Sort	HeapSort

Экспериментальное исследование

- Необходимо измерить время работы каждого алгоритма при различном количестве элементов в массиве — **заполните таблицу 2 для каждого алгоритма**
- По заполненной таблице 2 **постройте для каждого алгоритма график** зависимости времени его выполнения от числа элементов в массиве. Для построения графиков удобно использовать gnuplot, R, Scilab, LibreOffice Calc
- 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] — можно использовать функцию `getrand()` из примеров на сайте

На сайте приведены примеры функций для измерения времени выполнения кода (функция `wtime()`), а также пример работы с датчиком псевдослучайных чисел (функция `getrand()`).

В отчете должны быть следующие разделы:

- **Описание алгоритмов** — общая характеристика алгоритмов (желательно привести псевдокод), их свойства (in place, stable), вычислительная сложность и сложность по памяти
- **Организация экспериментов** — в этом разделе следует указать конфигурацию машины, на которой вы проводили эксперименты (модель процессора, объем памяти; версию операционной системы и ключи компиляции программы)
- **Результаты экспериментов** — таблицы, графики и выводы о эффективности алгоритмов

Таблица 2. Результаты экспериментов

#	Количество элементов в массиве	Время выполнения алгоритма, с
1	50 000	
2	100 000	
3	150 000	
…	…	
20	1 000 000	

Оси графиков должны быть подписаны — указаны название показателя и единицы его измерения. Например, «Время выполнения алгоритмы, с», «Количество элементов в массиве». Под графиком размещается подрисуночная подпись с пояснением зависимость какой величины от какого параметра на нем показана. Например, «Зависимость времени выполнения алгоритма Selection sort от размера массива» (см. рис. 1).

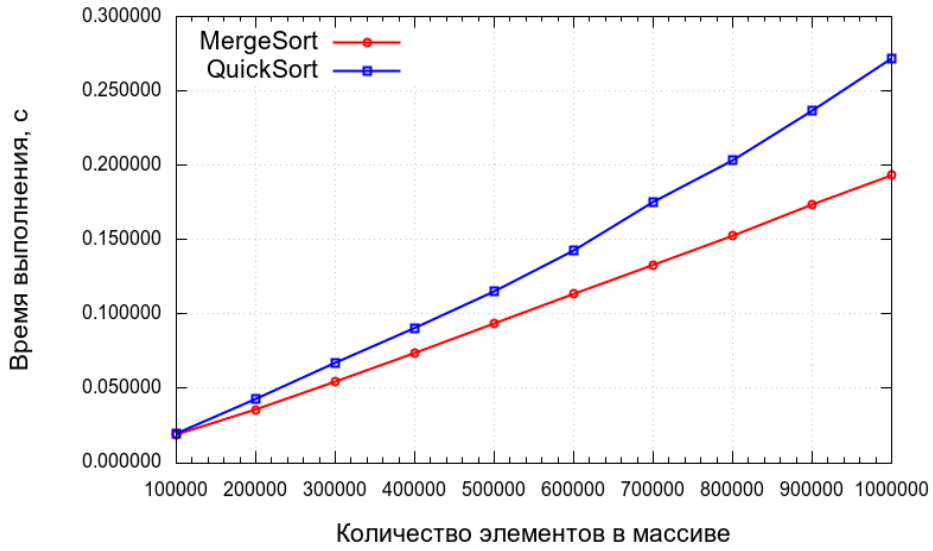

Рис. 1. Зависимость времени выполнения алгоритмов MergeSort и QuickSort от количества элементов в массиве

Контрольные вопросы

На защите отчета вы должны ответить на следующие вопросы:

- Что такое вычислительная сложность алгоритма (computational complexity)?
- Что означают записи $f(n) = O(g(n))$, $f(n) = \Theta(g(n))$, $f(n) = \Omega(g(n))$?
- Какой алгоритм сортировки называется устойчивым (stable)?
- Какой алгоритм сортировки называется сортировкой «на месте» (in place)?
- Какая вычислительная сложность в худшем случае у алгоритмов, которые Вы реализовали?
- Объясните поведение кривых на графиках, которые вы построили. Согласуются ли экспериментальные результаты с оценкой вычислительной сложности алгоритмов?
- Какие алгоритмы сортировки с вычислительной сложностью $O(n\log n)$ для худшего случая вам известны?
- Известны ли вам алгоритмы сортировки, работающие быстрее $O(n\log n)$ для худшего случая?