Лабораторная работа 1. Алгоритмы сортировки

Постановка задачи

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

Таблица 1. Распределение заданий по вариантам

Вариант	Алгоритмы, не использующие операцию сравнения	Квадратичные алгоритмы сортировки	«Быстрые» алгоритмы сортировки
1	Counting Sort	Bubble Sort	Merge Sort
2	Radix Sort	Selection Sort	Heap Sort
3	Counting Sort	Insertion Sort	Quick Sort
4	Radix Sort	Odd-Even Sort	Merge Sort
5	Counting Sort	Bubble Sort	Heap Sort
6	Radix Sort	Selection Sort	Quick Sort
7	Counting Sort	Insertion Sort	Merge Sort
8	Radix Sort	Odd-Even Sort	Heap Sort
9	Counting Sort	Bubble Sort	Quick Sort
10	Radix Sort	Selection Sort	Merge Sort
11	Counting Sort	Insertion Sort	Heap Sort
12	Radix Sort	Odd-Even Sort	Quick Sort
13	Counting Sort	Bubble Sort	Merge Sort
14	Radix Sort	Selection Sort	Heap Sort
15	Counting Sort	Insertion Sort	Quick Sort
16	Radix Sort	Odd-Even Sort	Merge Sort
17	Counting Sort	Bubble Sort	Merge Sort
18	Radix Sort	Selection Sort	Heap Sort
19	Counting Sort	Insertion Sort	Quick Sort
20	Radix Sort	Odd-Even Sort	Merge Sort
21	Counting Sort	Bubble Sort	Heap Sort
22	Radix Sort	Selection Sort	Quick Sort
23	Counting Sort	Insertion Sort	Merge Sort
24	Radix Sort	Odd-Even Sort	Heap Sort
25	Counting Sort	Bubble Sort	Quick Sort

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]

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	

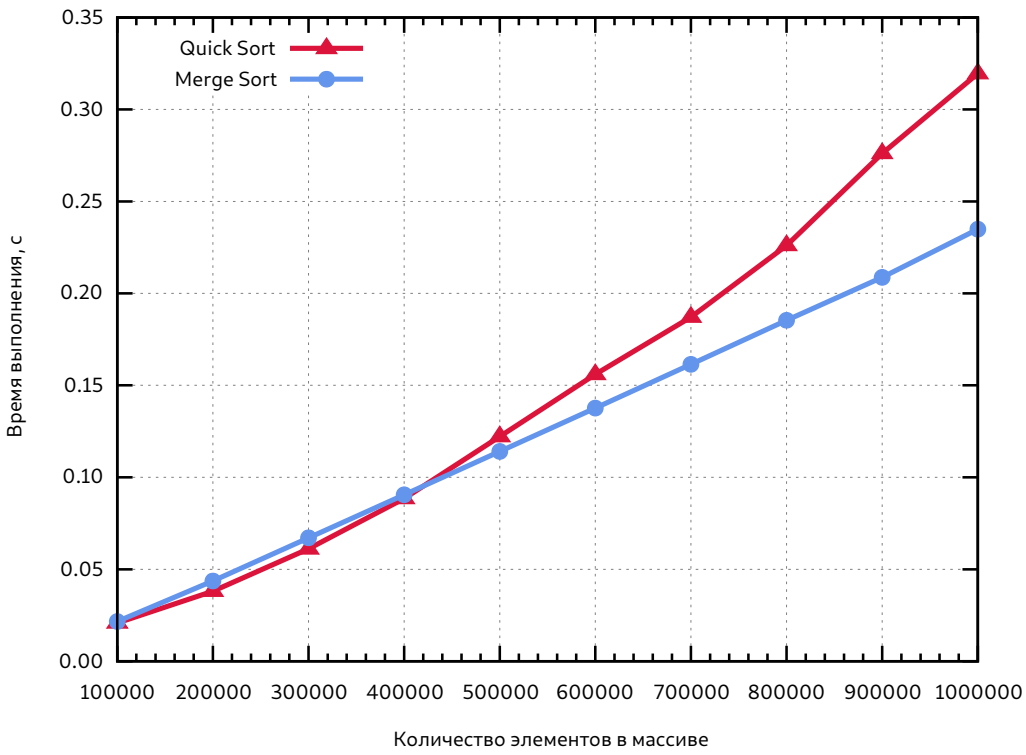

Рис. 1. Зависимость времени выполнения алгоритмов Quick Sort и Merge Sort от количества элементов в массиве

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

В приложении к данному файлу приведены примеры функций для измерения времени выполнения кода, пример работы с датчиком псевдослучайных чисел, а также пример скрипта для построения графиков на языке gnuplot.

Контрольные вопросы

- Что такое вычислительная сложность алгоритма?
- 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 реализованных вами алгоритмов?
- 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?

Приложение

Измерение времени выполнения кода

Для использования функции `wtime()` достаточно скопировать её в свою программу и подключить необходимый заголовочный файл. Функция `wtime()` возвращает текущее время в секундах.

```
#include <sys/time.h>

double wtime()
{
    struct timeval t;
    gettimeofday(&t, NULL);
    return (double)t.tv_sec + (double)t.tv_usec * 1E-6;
}
```

Генерация псевдослучайных чисел

Для использования функции `getrand()` достаточно скопировать её в свою программу и подключить необходимый заголовочный файл. Функция `getrand(int min, int max)` возвращает равномерно распределённое псевдослучайное число из интервала `[min, max)`.

```
#include <stdlib.h>

int getrand(int min, int max)
{
    return (double)rand() / (RAND_MAX + 1.0) * (max - min) + min;
}
```

Построение графиков в gnuplot

Экспериментальные данные (результаты замеров времени выполнения алгоритмов) записываются в текстовый файл. Пример файла `sort.dat` с экспериментальными данными:

```
100000      0.020796    0.021578
200000      0.038083    0.043593
300000      0.061112    0.067131
400000      0.088463    0.090547
500000      0.122182    0.114106
600000      0.155982    0.137603
700000      0.187101    0.161421
800000      0.226070    0.185390
900000      0.276111    0.208726
1000000     0.319532    0.234848
```

Для генерации svg-файла с графиком необходимо подготовить скрипт с командами на языке gnuplot. Пример скрипта:

```
#!/usr/bin/gnuplot

set termoption enhanced
set terminal svg size 800,600 font "Arial, 16"
set output "plot.svg"

set style line 1 lc rgb "0xDC143C" lt 1 lw 4 pt 9 ps 1
set style line 2 lc rgb "0x6495ED" lt 1 lw 4 pt 7 ps 1

set border lw 2
set grid
set key top left

set xlabel "Количество элементов в массиве"
set ylabel "Время выполнения, с" rotate by 90
set xtics 100000
set mxtics
set format x "%6.0f"
set format y "%.2f"

plot "sort.dat" using 1:2 title "Quick Sort" with linespoints ls 1, \
"sort.dat" using 1:3 title "Merge Sort" with linespoints ls 2
```