# ЛАБОРАТОРНАЯ РАБОТА №7

# АНАЛИЗ АЛГОРИТМОВ СОРТИРОВКИ

Цель работы

Экспериментально сравнить между собой алгоритмы сортировки по критериям сложности и количества операций. Получить навыки экспериментальной оценки сложности и количества операций для алгоритмов сортировки.

Задания

1. Написать программы реализации соответствующих алгоритмов сортировки согласно варианту задания (Таблица 1);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Простейший  алгоритм 1 | Простейший  алгоритм 2 | Простейший  алгоритм 3 | Структура и критерий сортировки |
| 12 | Сортировка прямым выбором | Метод «пузырька» | «Быстрая» сортировка | Структура Продукт, содержит название и стоимость, сортировка по возрастанию стоимости. |

Таблица 1 – Вариант задания

1. Дополнить алгоритмы операторами контроля числа операций, выполняемыми в ходе сортировки. Пользуясь средствами операционной системы определить объёмы программного кода, соответствующие исследуемым алгоритмам;
2. Выполнить вычислительные эксперименты. Сравнить производительность различных алгоритмов;

Текст программы

#include <iostream>

#include <ctime>

#include <string>

#include <windows.h>

using namespace std;

const int n = 100; // длина массива

// Структура для продукта

struct product

{

string name; // название

int price; // цена

};

// Меняет переменные местами

void swap(product &a, product &b)

{

product temp = a;

a = b;

b = temp;

return;

}

// Сортировка прямым выбором

int selectSort(product\* a, int n)

{

int counter = 2; // счётчик операций

for (int i = n; i > 1; i--) // заполнение i-й ячейки

{

counter += 11;

int imax = 0;

for (int j = 0; j < i; j++) // поиск максимального элемента

{

counter += 5;

if (a[j].price > a[imax].price)

{

counter++;

imax = j;

}

}

swap(a[i - 1], a[imax]); // вставка максимального элемента на своё место

}

return counter;

}

// "Быстрая" сортировка

int quickSort(product\* a, int n)

{

int counter = 5; // счётчик операций

int i = 0, j = n - 1; // поставить указатели на исходные места

product p = a[n / 2]; // центральный элемент

do // рекурсивные вызовы, если есть, что сортировать

{

counter += 6;

while (a[i].price < p.price)

{

counter += 3;

i++;

}

while (a[j].price > p.price)

{

counter += 3;

j--;

}

if (i <= j)

{

counter += 7;

swap(a[i], a[j]);

i++;

j--;

}

}

while (i <= j);

counter += 2;

if (j > 0) counter += quickSort(a, j + 1) + 1;

if (n > i) counter += quickSort(&a[i], n - i) + 2;

return counter;

}

// Сортировка методом пузырька

int bubbleSort(product\* a, int n)

{

int counter = 2; // счётчик операций

for (int i = n; i > 1; i--)

{

counter += 5;

for (int j = 0; j < i - 1; j++) // выталкивается самый тяжелый

{

counter += 7;

if (a[j].price > a[j + 1].price) // если он больше своего соседа

{

counter += 6;

swap(a[j], a[j + 1]);

}

}

}

return counter;

}

// Заполнение массива и измерение эффективности его сортировки

int main()

{

srand(time(0));

product array[n];

for (int i = 0; i < n; i++) // заполнение массива

{

array[i].name = to\_string(i + 1);

array[i].price = rand() % 1000 + 1;

}

LARGE\_INTEGER freq, start\_time, end\_time; // подсчёт времени

QueryPerformanceFrequency(&freq);

QueryPerformanceCounter(&start\_time);

int counter = quickSort(array, n); // сортировка и счётчик операций

QueryPerformanceCounter(&end\_time);

long long el\_time\_ns = (end\_time.QuadPart - start\_time.QuadPart) \* 1000000000LL / freq.QuadPart;

cout << "Operations performed - " << counter << endl;

cout << "Time - " << el\_time\_ns << " ns" << endl;

for (int i = 0; i < n; i++) // вывод массива

{

cout << array[i].name << ' ' << array[i].price << endl;

}

return 0;

}

Ход работы

Написанная программа была запущена для трёх видов сортировок при N=10 (длина массива). Все три функции сортировки верно выполнили свою задачу (Рисунок 1). Метод пузырьком заняла 2500 нс. при 476 операциях, быстрая сортировка заняла 4400 нс. при 313 операциях, сортировка прямым выбором заняла 2300 нс. при 375 операциях.

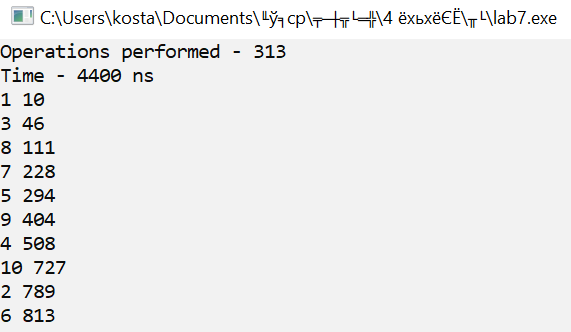


Рисунок 1 – Быстрая сортировка при N=10

Далее программа была запущена для N=100. Результаты оказали таковы:

* Пузырёк: 185,4 мкс., 50183 операции;
* Прямой выбор: 40,8 мкс., 26690 операций;
* Быстрая сортировка: 25,8 мкс., 5003 операции;

Программа была запущена для N=1000. Результаты оказали таковы:

* Пузырёк: 19109,7 мкс., 4988681 операции;
* Прямой выбор: 1506,7 мкс., 2518354 операций;
* Быстрая сортировка: 338,8 мкс., 72959 операции;

Также была зафиксирована память, занимаемая кодом для алгоритмов на языке C++: метод пузырька − 360 Б., быстрая сортировка − 646 Б., метод прямого выбора − 454 Б.

Вывод

В ходе работы были проведены экспериментальные сравнения алгоритмов сортировки (метод пузырька, метод прямого выбора, быстрая сортировка) по критериям сложности и количества операций.

В результате выяснилось, что при малых длинах массивов методы пузырька и выбора оказываются гораздо проще, быстрее и эффективнее быстрой сортировки. Однако при многократной увеличении длины быстрая сортировка значительно обходит остальные два алгоритма по всем параметрам, за исключением сложности восприятия и занимаемой памяти.