Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в ИЗ»

на тему «Оценка времени выполнения программ»

Выполнил:

студент групп 22ВВВ2

Изосин М.А.

Приняли:

д.т.н. профессор Митрохин М.А.

к.э.н. доцент Акифьев И.В.

Пенза 2023

**Название**

Оценка времени выполнения программ

**Цель работы**

Сравнить время сортировок Шелла, быстрой сортировки и сортировки qsort

**Лабораторное задание**

**Задание 1:**

1. Вычислить порядок сложности программы (*О*-символику).
2. Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000.
3. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

**Задание 2**:

1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.
2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.
3. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.
4. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.
5. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

**Листинг**

1)

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <malloc.h>

#define m 860 // строковая константа "число столбцов"

#define n 860// строковая константа "число строк"

int main(void)

{

setvbuf(stdin, NULL, \_IONBF, 0);

setvbuf(stdout, NULL, \_IONBF, 0);

clock\_t start, end; // объявляем переменные для определения времени выполнения

int i = 0, j = 0, r;

int elem\_c;

int \*\*one = (int\*\*)malloc(n \* sizeof(int\*)); // выделяем память под динамический двумерный массив a[n][m]

for(int i = 0; i < n; i++)

{

one[i] = (int\*)malloc(m \* sizeof(int));

}

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

while(i < n)

{

while(j < m)

{

one[i][j] = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

int \*\*two = (int\*\*)malloc(n \* sizeof(int\*)); // выделяем память под динамический двумерный массив b[n][m]

for(int i = 0; i < n; i++)

{

two[i] = (int\*)malloc(m \* sizeof(int));

}

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

i = 0; j = 0;

while(i < n)

{

while(j < m)

{

two[i][j] = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

int \*\*tree = (int\*\*)malloc(n \* sizeof(int\*)); // выделяем память под динамический двумерный массив c[n][m]

for(int i = 0; i < n; i++)

{

tree[i] = (int\*)malloc(m \* sizeof(int));

}

start = clock(); // в переменную start записываем время от начала запуска программы

for(i=0;i < n;i++)

{

for(j=0;j < m;j++)

{

elem\_c=0;

for(r=0;r < n;r++)

{

elem\_c=elem\_c+one[i][r] \* two[r][j];

tree[i][j]=elem\_c;

}

}

}

end = clock(); // в переменную end записываем время от начала запуска программы

double time = (end - start) / CLOCKS\_PER\_SEC; // вычисляем разность(т.е. время, затраченное на умножение матриц)

printf("%lf s\n", time); // выводим результат работы программы в секундах

for(int i = 0; i < n; i++) // очищаем память динамического двумерного массива a[n][m]

{

free(one[i]);

}

free(one);

for(int i = 0; i < n; i++) // очищаем память динамического двумерного массива b[n][m]

{

free(two[i]);

}

free(two);

for(int i = 0; i < n; i++) // очищаем память динамического двумерного массива c[n][m]

{

free(tree[i]);

}

free(tree);

return(0);

}

2)

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <malloc.h>

#include <locale.h>

void shell(int \*items, int count)

{

int i, j, gap, k;

int x, a[5];

a[0]=9; a[1]=5; a[2]=3; a[3]=2; a[4]=1;

for(k=0; k < 5; k++) {

gap = a[k];

for(i=gap; i < count; ++i) {

x = items[i];

for(j=i-gap; (x < items[j]) && (j >= 0); j=j-gap)

items[j+gap] = items[j];

items[j+gap] = x;

}

}

}

void qs(int \*items, int left, int right) //вызов функции: qs(items, 0, count-1);

{

int i, j;

int x, y;

i = left; j = right;

/\* выбор компаранда \*/

x = items[(left+right)/2];

do {

while((items[i] < x) && (i < right)) i++;

while((x < items[j]) && (j > left)) j--;

if(i <= j) {

y = items[i];

items[i] = items[j];

items[j] = y;

i++; j--;

}

} while(i <= j);

if(left < j) qs(items, left, j);

if(i < right) qs(items, i, right);

}

int comp(const void \* x1, const void \* x2)

{

return ( (int)x1 - (int)x2 );

}

int main2()

{

setlocale(LC\_ALL, "rus");

clock\_t start, end; // объявляем переменные для определения времени выполнения

int \*mas\_qs, \*mas\_sh, \*mas\_qsort; // создаем указатели на массивы для сортировок

int k; // определяет размер динамических массивов

printf("Введите размер массива > 1\n");

scanf("%d", &k);

while (k <= 1)

{

printf("Введите размер массива еще раз:\n");

scanf("%d", &k);

}

mas\_qs = (int\*)malloc(k \* sizeof(k)); // выделяем память под динамический массив

mas\_sh = (int\*)malloc(k \* sizeof(k)); // выделяем память под динамический массив

mas\_qsort = (int\*)malloc(k \* sizeof(k)); // выделяем память под динамический массив

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

for (int i = 0; i < k; i++)

{

mas\_qs[i] = rand() % 100 + 1; // заполняем массив случайными элементами

mas\_sh[i] = mas\_qs[i]; // копируем этот массив в другой

mas\_qsort[i] = mas\_qs[i]; // копируем этот массив в другой

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

start = clock(); // в переменную start записываем время от начала запуска программы

shell(mas\_sh, k); // вызываем сортировку Шелла

end = clock(); // в переменную end записываем время от начала запуска программы

long time\_shell = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на сортировку Шелла)

start = clock(); // в переменную start записываем время от начала запуска программы

qs(mas\_qs, 0, k - 1); // вызываем быструю сортировку

end = clock(); // в переменную end записываем время от начала запуска программы

long time\_qs = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на быструю сортировку)

start = clock(); // в переменную start записываем время от начала запуска программы

qsort(mas\_qsort, k, sizeof(int), comp); // вызываем быструю сортировку

end = clock(); // в переменную end записываем время от начала запуска программы

long time\_qsort = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на быструю сортировку)

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("| |Сортировка Шелла(ms)|Быстрая сортировка(ms)|Сортировка qsort(ms)|\n");

printf("|-----------|--------------------|----------------------|--------------------|\n");

printf("|Случ. набор|%20d|%22d|%20d|\n", time\_shell, time\_qs, time\_qsort); // выводим результат работы быстрой сортировки в миллисекундах

printf("|-----------|--------------------|----------------------|--------------------|\n");

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for (int i = 0; i < k; i++)

{

mas\_qs[i] = i; // копируем в этот массив, возрастающий

mas\_sh[i] = i; // копируем в этот массив, возрастающий

mas\_qsort[i] = i; // копируем в этот массив, возрастающий

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

start = clock(); // в переменную start записываем время от начала запуска программы

shell(mas\_sh, k); // вызываем сортировку Шелла

end = clock(); // в переменную end записываем время от начала запуска программы

time\_shell = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на сортировку Шелла)

start = clock(); // в переменную start записываем время от начала запуска программы

qs(mas\_qs, 0, k - 1); // вызываем быструю сортировку

end = clock(); // в переменную end записываем время от начала запуска программы

time\_qs = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на быструю сортировку)

start = clock(); // в переменную start записываем время от начала запуска программы

qsort(mas\_qsort, k, sizeof(int), comp); // вызываем быструю сортировку

end = clock(); // в переменную end записываем время от начала запуска программы

time\_qsort = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на быструю сортировку)

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

printf("|Прямая п. |%20d|%22d|%20d|\n", time\_shell, time\_qs, time\_qsort); // выводим результат работы быстрой сортировки в миллисекундах

printf("|-----------|--------------------|----------------------|--------------------|\n");

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for (int i = 0; i < k; i++)

{

mas\_qs[i] = k - i; // копируем в этот массив, убывающий

mas\_sh[i] = k - i; // копируем в этот массив, убывающий

mas\_qsort[i] = k - i; // копируем в этот массив, убывающий

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

start = clock(); // в переменную start записываем время от начала запуска программы

shell(mas\_sh, k); // вызываем сортировку Шелла

end = clock(); // в переменную end записываем время от начала запуска программы

time\_shell = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на сортировку Шелла)

start = clock(); // в переменную start записываем время от начала запуска программы

qs(mas\_qs, 0, k - 1); // вызываем быструю сортировку

end = clock(); // в переменную end записываем время от начала запуска программы

time\_qs = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на быструю сортировку)

start = clock(); // в переменную start записываем время от начала запуска программы

qsort(mas\_qsort, k, sizeof(int), comp); // вызываем быструю сортировку

end = clock(); // в переменную end записываем время от начала запуска программы

time\_qsort = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на быструю сортировку)

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

printf("|Обратная п.|%20d|%22d|%20d|\n", time\_shell, time\_qs, time\_qsort); // выводим результат работы быстрой сортировки в миллисекундах

printf("|-----------|--------------------|----------------------|--------------------|\n");

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for (int i = 0; i < k; i++)

{

if (i < (k / 2))

{

mas\_qs[i] = i; // копируем в этот массив, возрастающий

mas\_sh[i] = i; // копируем в этот массив, возрастающий

mas\_qsort[i] = i; // копируем в этот массив, у возрастающий

}

else

{

mas\_qs[i] = k - 1 - i; // копируем в этот массив, убывающий

mas\_sh[i] = k - 1 - i; // копируем в этот массив, убывающий

mas\_qsort[i] = k - 1 - i; // копируем в этот массив, убывающий

}

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

start = clock(); // в переменную start записываем время от начала запуска программы

shell(mas\_sh, k); // вызываем сортировку Шелла

end = clock(); // в переменную end записываем время от начала запуска программы

time\_shell = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на сортировку Шелла)

start = clock(); // в переменную start записываем время от начала запуска программы

qs(mas\_qs, 0, k - 1); // вызываем быструю сортировку

end = clock(); // в переменную end записываем время от начала запуска программы

time\_qs = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на быструю сортировку)

start = clock(); // в переменную start записываем время от начала запуска программы

qsort(mas\_qsort, k, sizeof(int), comp); // вызываем быструю сортировку

end = clock(); // в переменную end записываем время от начала запуска программы

time\_qsort = (end - start) \* 1000 / CLOCKS\_PER\_SEC ; // вычисляем разность(т.е. время, затраченное на быструю сортировку)

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

printf("|Прям-обр п.|%20d|%22d|%20d|\n", time\_shell, time\_qs, time\_qsort); // выводим результат работы быстрой сортировки в миллисекундах

printf("|-----------|--------------------|----------------------|--------------------|\n");

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

free(mas\_qs); // очищаем память

free(mas\_sh); // очищаем память

free(mas\_qsort); // очищаем память

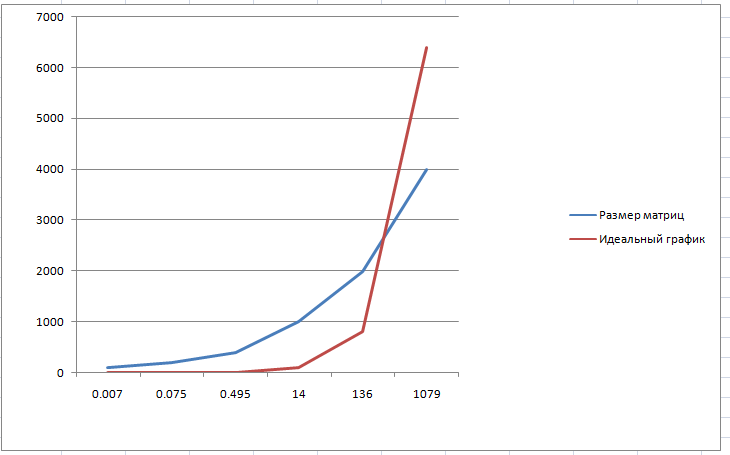
return 0;

}

**Сложность программы**

Сложность программы - O(n3), т.к. программа имеет 3 вложенных цикла.

**График**

****

На графике зависимости размера матриц от времени выполнения программы показаны 2 кривые. Синяя - практическая кривая, полученная в результате замера времени сортировки. Красна - теоретичекий график сложности O(n3).

**Результат работы программы**

Результат работы программы показан на рисунках.

1)

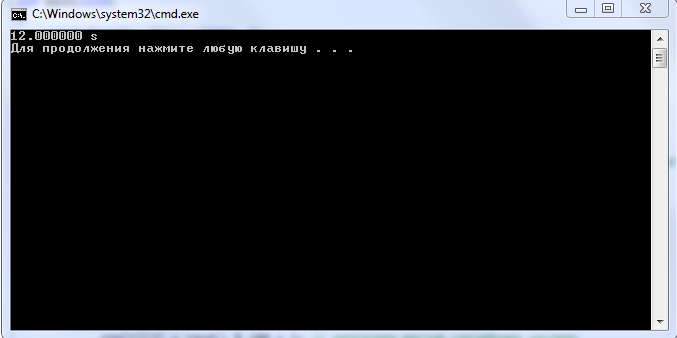


Рис. 1

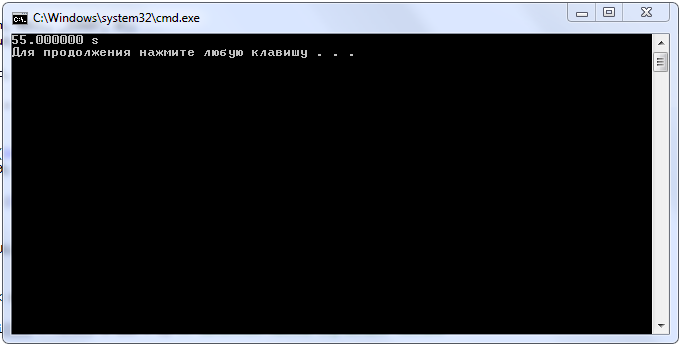


Рис. 2

2)

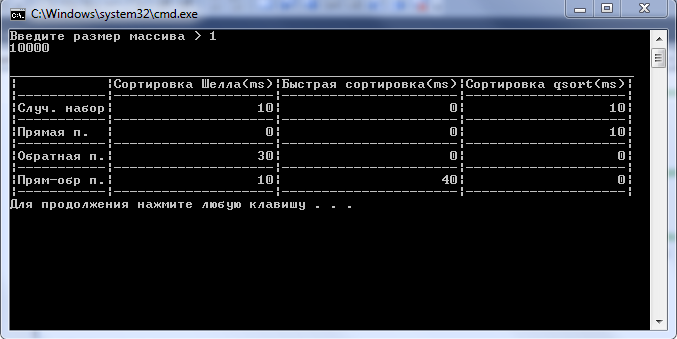


Рис. 3

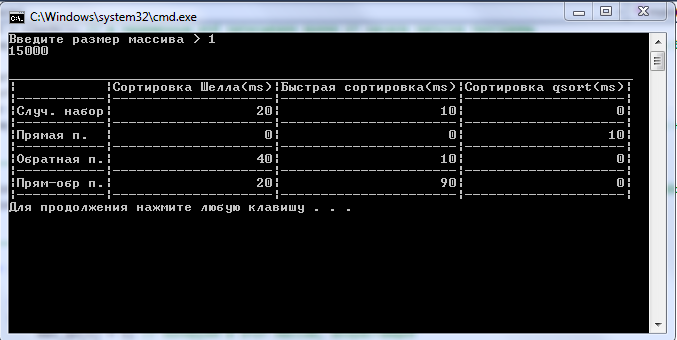


Рис. 4

**Вывод:**

В ходе выполнения лабораторной работы мы оценили время, затраченное на сортировку Шелла, быструю и qsort на разных наборах массива. Создали график зависимости времени выполнения программы от размера матриц и сравнили его с теоретическим графиком сложности O(n3).

Ссылка на репозиторий: [https://github.com/megawalnut/Lab2-log.git](https://vk.com/away.php?to=https%3A%2F%2Fgithub.com%2Fmegawalnut%2FLab2-log.git&cc_key=)