Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №2  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Оценка времени выполнения программ»

Выполнили студенты группы 22ВВВ2:  
Хоссейни Нежад С. А. С. М.

Приняли:  
Акифьев И. В.  
Митрохин М. А.

Пенза 2023

**Название**

Оценка времени выполнения программ

**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.

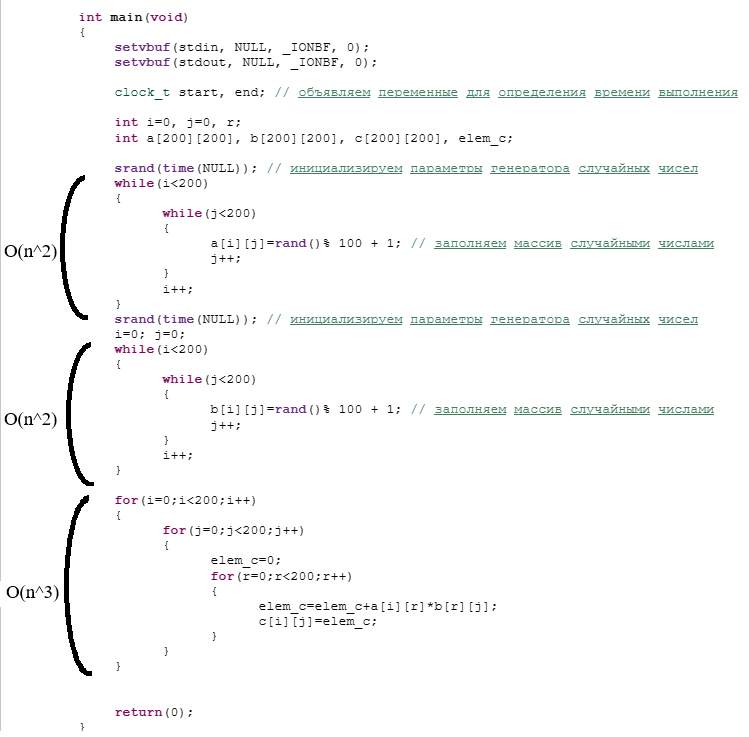
Задание 2:

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.

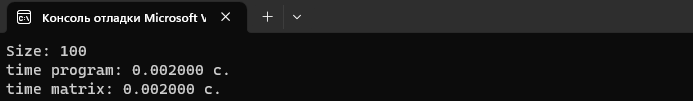
**Ход работы**

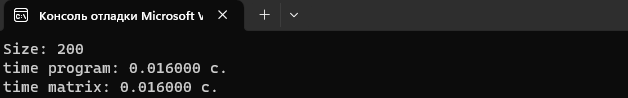
**1 задание**

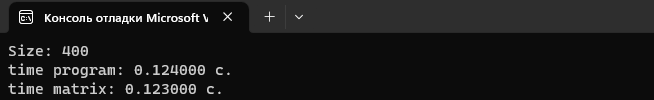
1) Сложность программы O(n^3)

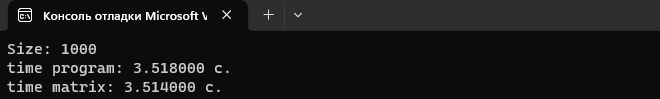


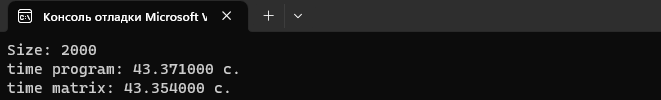
2) Время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000.

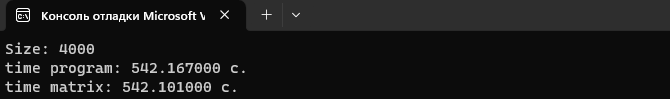


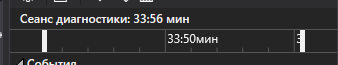












3) График

**Вывод:** по графику видно, что программа выполняется намного быстрее, чем мы высчитали аналитическим способом.

**2 задание**

Размер массива: 0

Случайные числа

Время shell:0.000000 с.

Время qs:0.000000 с.

Время стандартной qsort: 0.000000 с.

Возрастающая последовательность

Время shell:0.000000 с.

Время qs:0.000000 с.

Время стандартной qsort: 0.000000 с.

Убывающая последовательность

Время shell:0.000000 с.

Время qs:0.000000 с.

Время стандартной qsort: 0.000000 с.

Половина массива возрастающая, половина убывающая

Время shell:0.000000 с.

Время qs:0.000000 с.

Время стандартной qsort: 0.000000 с.

Размер массива: 20000

Случайные числа

Время shell:0.015000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.004000 с.

Возрастающая последовательность

Время shell:0.000000 с.

Время qs:0.000000 с.

Время стандартной qsort: 0.006000 с.

Убывающая последовательность

Время shell:0.030000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.007000 с.

Половина массива возрастающая, половина убывающая

Время shell:0.007000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.007000 с.

Размер массива: 40000

Случайные числа

Время shell: 0.065000 с.

Время qs:0.002000 с.

Время стандартной qsort: 0.009000 с.

Возрастающая последовательность

Время shell: 0.001000 с.

Время qs:0.000000 с.

Время стандартной qsort: 0.012000 с.

Убывающая последовательность

Время shell: 0.129000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.014000 с.

Половина массива возрастающая, половина убывающая

Время shell: 0.031000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.013000 с.

Размер массива: 60000

Случайные числа

Время shell: 0.136000 с.

Время qs:0.003000 с.

Время стандартной qsort: 0.014000 с.

Возрастающая последовательность

Время shell: 0.001000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.021000 с.

Убывающая последовательность

Время shell: 0.280000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.021000 с.

Половина массива возрастающая, половина убывающая

Время shell: 0.066000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.022000 с.

Размер массива: 80000

Случайные числа

Время shell: 0.235000 с.

Время qs:0.004000 с.

Время стандартной qsort: 0.018000 с.

Возрастающая последовательность

Время shell: 0.001000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.026000 с.

Убывающая последовательность

Время shell: 0.467000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.028000 с.

Половина массива возрастающая, половина убывающая

Время shell: 0.116000 с.

Время qs:0.001000 с.

Время стандартной qsort: 0.026000 с.

Размер массива: 100000

Случайные числа

Время shell: 0.357000 с.

Время qs:0.006000 с.

Время стандартной qsort: 0.023000 с.

Возрастающая последовательность

Время shell: 0.001000 с.

Время qs:0.002000 с.

Время стандартной qsort: 0.032000 с.

Убывающая последовательность

Время shell: 0.708000 с.

Время qs:0.002000 с.

Время стандартной qsort: 0.036000 с.

Половина массива возрастающая, половина убывающая

Время shell: 0.186000 с.

Время qs:0.002000 с.

Время стандартной qsort: 0.033000 с.

С случайными числами лучше всего себя показывает быстрая сортировка, а хуже всего сортировка Шелла. С возрастающей последовательностью сортировка Шелла показывает результат тем лучше, чем больше массив, а стандартная сортировка показывает наихудший результат. С убывающей последовательностью сортировка показывает самое медленное время работы, в то время как быстрая сортировка показывает очень хороший результат. С половиной возрастающим и убывающим массивом быстрее всех работает быстрая сортировка, а медленее все сортировка Шелла.

**Вывод:** Быстрее всех работает в основном быстрая сортировка.

**Листинг**

**1 задание**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 10000

int main(void)

{

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

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

clock\_t start, end, startCycle;

start = clock();

int i = 0, j = 0, r;

int elem\_c;

int\*\* a,\*\* b,\*\* c;

a = (int\*\*)malloc(SIZE \* sizeof(int\*));

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

{

a[i] = (int\*)malloc(SIZE \* sizeof(int));

}

b = (int\*\*)malloc(SIZE \* sizeof(int\*));

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

{

b[i] = (int\*)malloc(SIZE \* sizeof(int));

}

c = (int\*\*)malloc(SIZE \* sizeof(int\*));

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

{

c[i] = (int\*)malloc(SIZE \* sizeof(int));

}

srand(time(NULL));

while (i < SIZE)

{

while (j < SIZE)

{

a[i][j] = rand() % 100 + 1;

j++;

}

i++;

}

srand(time(NULL));

i = 0; j = 0;

while (i < SIZE)

{

while (j < SIZE)

{

b[i][j] = rand() % 100 + 1;

j++;

}

i++;

}

startCycle = clock();

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

{

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

{

elem\_c = 0;

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

{

elem\_c = elem\_c + a[i][r] \* b[r][j];

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

}

}

}

end = clock();

printf("Size: %d", SIZE);

double time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime program: %f c.", time\_spent);

time\_spent = (double)(end - startCycle) / CLOCKS\_PER\_SEC;

printf("\ntime matrix: %f c.", time\_spent);

return(0);

}

**2 задание**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void shell(int\* items, int count);

void qs(int\* items, int left, int right);

void fillArray(int\* Arr, int size, int direction);

void fillArrayHalf(int\* Arr, int size);

int compare(const void\* x1, const void\* x2);

void exercise1(int\* Arr, int size);

void exercise2(int\* Arr, int size);

void exercise3(int\* Arr, int size);

void exercise4(int\* Arr, int size);

int main()

{

srand(time(NULL));

for (int i = 0; i < 120000; i += 20000)

{

printf("Size: %d", i);

int \*Arr = (int\*)malloc(i \* sizeof(int));

exercise1(Arr, i);

exercise2(Arr, i);

exercise3(Arr, i);

exercise4(Arr, i);

free(Arr);

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

}

return 0;

}

void exercise1(int\* Arr, int size)

{

printf("\n\nRandom numbers");

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

{

Arr[i] = rand() % 100;

}

clock\_t start = clock();

shell(Arr, size);

clock\_t end = clock();

double time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime shell: %f s.", time\_spent);

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

{

Arr[i] = rand() % 100;

}

start = clock();

qs(Arr, 0, size - 1);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime qs: %f s.", time\_spent);

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

{

Arr[i] = rand() % 100;

}

start = clock();

qsort(Arr, size, sizeof(int), compare);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime standart qsort: %f s.", time\_spent);

}

void exercise2(int\* Arr, int size)

{

printf("\n\nIncreasing sequence");

fillArray(Arr, size, 0);

clock\_t start = clock();

shell(Arr, size);

clock\_t end = clock();

double time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime shell: %f s.", time\_spent);

fillArray(Arr, size, 0);

start = clock();

qs(Arr, 0, size - 1);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime qs: %f s.", time\_spent);

fillArray(Arr, size, 0);

start = clock();

qsort(Arr, size, sizeof(int), compare);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime standart qsort: %f s.", time\_spent);

}

void exercise3(int\* Arr, int size)

{

printf("\n\nDescending sequencee");

fillArray(Arr, size, 1);

clock\_t start = clock();

shell(Arr, size);

clock\_t end = clock();

double time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime shell: %f s.", time\_spent);

fillArray(Arr, size, 1);

start = clock();

qs(Arr, 0, size - 1);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime qs: %f s.", time\_spent);

fillArray(Arr, size, 1);

start = clock();

qsort(Arr, size, sizeof(int), compare);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime standart qsort: %f s.", time\_spent);

}

void exercise4(int\* Arr, int size)

{

printf("\n\nHalf the numbers are increasing, half are decreasing");

fillArrayHalf(Arr, size);

clock\_t start = clock();

shell(Arr, size);

clock\_t end = clock();

double time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime shell: %f s.", time\_spent);

fillArrayHalf(Arr, size);

start = clock();

qs(Arr, 0, size - 1);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime qs: %f s.", time\_spent);

fillArrayHalf(Arr, size);

start = clock();

qsort(Arr, size, sizeof(int), compare);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("\ntime standart qsort: %f s.", time\_spent);

}

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);

}

void fillArray(int\* Arr, int size, int direction)

{

if(direction == 1)

{

for (int i = size-1; i > 0; i--)

{

Arr[size - 1 - i] = i;

}

}

else

{

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

{

Arr[i] = i;

}

}

}

void fillArrayHalf(int\* Arr, int size)

{

for (int i = 0; i < size/2; i++)

{

Arr[i] = i;

}

for (int i = size - 1; i > size/2 - 1; i--)

{

Arr[size - 1 - i + size/2] = i;

}

}

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

{

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

}

**Вывод**

В результате выполнения лабораторной работы я научился оценивать время работы программы.