Министерство образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили

студенты группы 22ВВВ1:

Ганин И.Р.

Курушин Я.С.

Приняли:

К.т.н, доцент Акифьев И. В.

К.т.н, доцент Юрова О. В.

**Цель работы**

Научиться оценивать время работы программы, пользоваться алгоритмами быстрой сортировки и сортировки Шелла, научиться оценивать сложность работы программы.

Для оценки времени выполнения программ языка Си или их частей могут использоваться средства, предоставляемые библиотекой **time.h**. Данная библиотека содержит описания типов и прототипы функций для работы с датой и временем.

**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**

**Теоретическая часть**

Типы данных:

1. clock\_t - возвращается функцией clock(). Обычно определён как int или long int.

2. time\_t - возвращается функцией time(). Обычно определён как int или long int.

3. struct tm - нелинейное, дискретное календарное представление времени.

Основные функции:

1.  clock\_t clock(void) - возвращает время, измеряемое процессором в тактах от начала выполнения программы, или −1, если оно не известно. Пересчет этого времени в секунды выполняется по формуле:

clock() / CLOCKS\_PER\_SEC

где CLOCKS\_PER\_SEC – константа, определяющая количество тактов системных часов в секунду.

2. time\_t time(time\_t \*tp)

    Возвращает текущее календарное время или −1, если это время не известно. Если указатель tp не равен NULL, то возвращаемое значение записывается также и в \*tp.

3. double difftime(time\_t time2,time\_t time1)

    Возвращает разность time2-time1, выраженную в секундах.

**Практическая часть**

Дана программа, вычисляющая произведение двух матриц:

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <time.h>

**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** a[200][200], b[200][200], c[200][200], elem\_c;

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

**while**(i<200)

{

**while**(j<200)

{

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

j++;

}

i++;

}

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

i=0; j=0;

**while**(i<200)

{

**while**(j<200)

{

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

j++;

}

i++;

}

**for**(i=0;i<200;i++)

{

**for**(j=0;j<200;j++)

{

elem\_c=0;

**for**(r=0;r<200;r++)

{

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

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

}

}

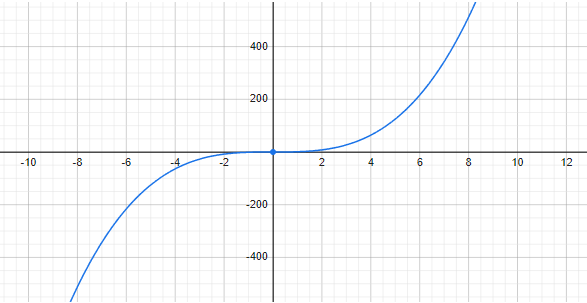
}

**return**(0);

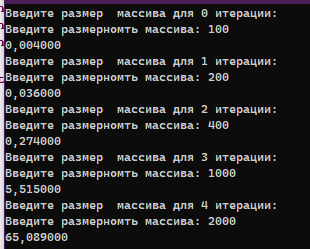
}

**Задание 1:**

1. Вычислить порядок сложности программы (*О*-символику).

Сложность инициализации и заполнения массива a составляет O(1)  
  
Сложность инициализации и заполнения массива b составляет O(1)  
  
Сложность умножения матриц составляет O(n3)  
  
Таким образом, итоговая сложность программы будет определяться умножением матриц и составляет O(n3) и иметь график 

1. 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.



Результат работы программы

После кол-ва чисел = 4000 - время работы программы становится очень большим.

1. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

Даны реализации алгоритмов сортировки Шелла и быстрой сортировки:

**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);

}

**Задание 2**:

1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.

void ran(int\* arr, int size) {

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

arr[i] = rand() % 100;

}

}

1. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.

void voz(int\* arr, int size) {

arr[0] = rand() % 100;

for (int i = 1; i < size; i++) {

arr[i] = arr[i - 1] + rand() % 100;

}

}

1. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

void yb(int\* arr, int size) {

arr[size - 1] = rand() % 100;

for (int i = size - 2; i >= 0; i--) {

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

}

}

1. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

void vozyb(int\* arr, int size\_2, int size) {

arr[0] = rand() % 100;

size\_2 = size / 2;

for (int i = 1; i < size\_2; i++) {

arr[i] = arr[i - 1] + rand() % 100;

}

arr[size - 1] = rand() % 100;

for (int i = size - 2; i >= size\_2; i--) {

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

}

}

1. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных. (Кол-во данных = 20000.)

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--;

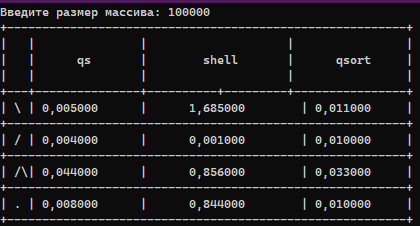
}

} while (i <= j);

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

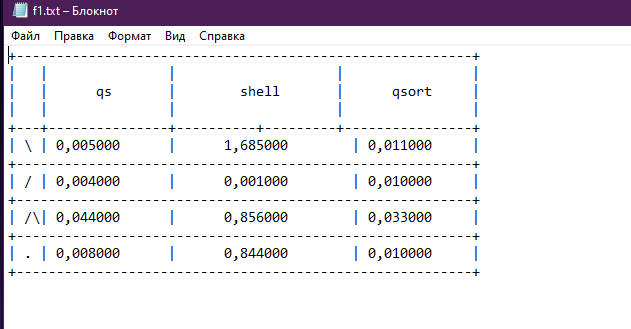
if (i < right) qs(items, i, right);

}

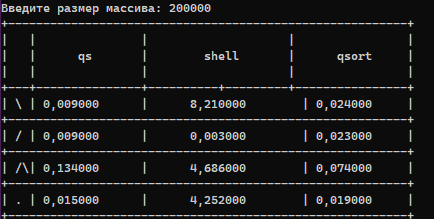


Результат работы в программы на 100000 элементах

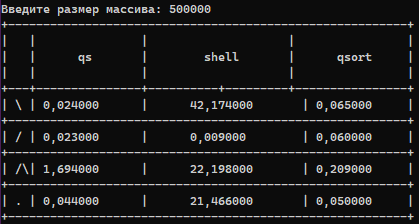
Как можно заметить по первой таблице – Shell является самым медленным алгоритмом из всех предложенных.



Запись данных в файл



Результат работы в программы на 200000 элементах



Результат работы в программы на 500000 элементах

**Вывод:**

В данной лабораторной работе, мы научились как вычислять сложность прогрммы. Вспомнили такие алгоритмы сортировки, как Shell, Qs, Встроенный Qsort. Вспомнили, как работать с таймером и замерять время работы определенных алгоритмов.

**Приложение А  
Листинг**

**Файл source.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

int main() {

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

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

setlocale(LC\_ALL, "RUS");

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

int\*\* arr = 0;

int\*\* brr = 0;

int\*\* crr = 0;

int i = 0, j = 0, r, rows, cols, p;

int elem\_c;

double time\_spent = 0.0;

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

for (p = 0; p < 7; p++) {

printf("Введите размер массива для %d итерации:\n", p);

printf("Введите размерномть массива: ");

scanf("%d", &rows);

arr = (int\*\*)malloc(rows \* sizeof(int\*));

brr = (int\*\*)malloc(rows \* sizeof(int\*));

crr = (int\*\*)malloc(rows \* sizeof(int\*));

if (arr == NULL) {

printf("Не удалось выделить память!\n");

return (0);

}

for (i = 0; i < rows; i++) {

arr[i] = (int\*)malloc(rows \* sizeof(int));

}if (brr == NULL) {

printf("Не удалось выделить память!\n");

return (0);

}

for (i = 0; i < rows; i++) {

brr[i] = (int\*)malloc(rows \* sizeof(int));

}if (crr == NULL) {

printf("Не удалось выделить память!\n");

return (0);

}

for (i = 0; i < rows; i++) {

crr[i] = (int\*)malloc(rows \* sizeof(int));

}

i = 0; j = 0;

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

{

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

{

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

}

}

i = 0; j = 0;

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

{

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

{

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

}

}

start = clock();

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

{

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

{

elem\_c = 0;

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

{

elem\_c = elem\_c + arr[i][r] \* brr[r][j];

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

}

}

}

end = clock();

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

printf("%f\n", time\_spent);

}

return (0);

}

**Файл source.cpp2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <time.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);

}

void voz(int\* arr, int size) {

arr[0] = rand() % 100;

for (int i = 1; i < size; i++) {

arr[i] = arr[i - 1] + rand() % 100;

}

}

void yb(int\* arr, int size) {

arr[size - 1] = rand() % 100;

for (int i = size - 2; i >= 0; i--) {

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

}

}

void vozyb(int\* arr, int size\_2, int size) {

arr[0] = rand() % 100;

size\_2 = size / 2;

for (int i = 1; i < size\_2; i++) {

arr[i] = arr[i - 1] + rand() % 100;

}

arr[size - 1] = rand() % 100;

for (int i = size - 2; i >= size\_2; i--) {

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

}

}

void ran(int\* arr, int size) {

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

arr[i] = rand() % 100;

}

}

int comp(const void\* x, const void\* y) {

return (\*(int\*)x - \*(int\*)y);

}

void main() {

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

int size, i, choice, size\_2;

clock\_t start, end;

double time\_spent\_1 = 0.0, time\_spent\_2 = 0.0, time\_spent\_3 = 0.0, time\_spent\_4 = 0.0, time\_spent\_5 = 0.0,

time\_spent\_6 = 0.0, time\_spent\_7 = 0.0, time\_spent\_8 = 0.0, time\_spent\_9 = 0.0, time\_spent\_10 = 0.0,

time\_spent\_11 = 0.0, time\_spent\_12 = 0.0;

int\* arr = 0;

printf("Введите размер массива: ");

scanf("%d", &size);

arr = (int\*)malloc(size \* sizeof(int));

if (arr == NULL) {

printf("Не удалось выделить память!\n");

return;

}

size\_2 = size / 2;

yb(arr, size);

start = clock();

qs(arr, 0, size - 1);

end = clock();

time\_spent\_1 = (double)(end - start) / CLOCKS\_PER\_SEC;

yb(arr, size);

start = clock();

shell(arr, size);

end = clock();

time\_spent\_2 = (double)(end - start) / CLOCKS\_PER\_SEC;

yb(arr, size);

start = clock();

qsort(arr, size, sizeof(int), comp);

end = clock();

time\_spent\_3 = (double)(end - start) / CLOCKS\_PER\_SEC;

voz(arr, size);

start = clock();

qs(arr, 0, size - 1);

end = clock();

time\_spent\_4 = (double)(end - start) / CLOCKS\_PER\_SEC;

voz(arr, size);

start = clock();

shell(arr, size);

end = clock();

time\_spent\_5 = (double)(end - start) / CLOCKS\_PER\_SEC;

voz(arr, size);

start = clock();

qsort(arr, size, sizeof(int), comp);

end = clock();

time\_spent\_6 = (double)(end - start) / CLOCKS\_PER\_SEC;

vozyb(arr, size\_2, size);

start = clock();

qs(arr, 0, size - 1);

end = clock();

time\_spent\_7 = (double)(end - start) / CLOCKS\_PER\_SEC;

vozyb(arr, size\_2, size);

start = clock();

shell(arr, size);

end = clock();

time\_spent\_8 = (double)(end - start) / CLOCKS\_PER\_SEC;

vozyb(arr, size\_2, size);

start = clock();

qsort(arr, size, sizeof(int), comp);

end = clock();

time\_spent\_9 = (double)(end - start) / CLOCKS\_PER\_SEC;

ran(arr, size);

start = clock();

qs(arr, 0, size - 1);

end = clock();

time\_spent\_10 = (double)(end - start) / CLOCKS\_PER\_SEC;

ran(arr, size);

start = clock();

shell(arr, size);

end = clock();

time\_spent\_11 = (double)(end - start) / CLOCKS\_PER\_SEC;

ran(arr, size);

start = clock();

qsort(arr, size, sizeof(int), comp);

end = clock();

time\_spent\_12 = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("+---------------------------------------------------------+\n"

"| | | | |\n"

"| | qs | shell | qsort |\n"

"| | | | |\n"

"+---+---------------+----------+---------+----------------+\n");

printf("| \\ | %-14f| %-16f| %-13f|\n", time\_spent\_1, time\_spent\_2, time\_spent\_3);

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

printf("| / | %-14f| %-16f| %-13f|\n", time\_spent\_4, time\_spent\_5, time\_spent\_6);

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

printf("| /\\| %-14f| %-16f| %-13f|\n", time\_spent\_7, time\_spent\_8, time\_spent\_9);

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

printf("| . | %-14f| %-16f| %-13f|\n", time\_spent\_10, time\_spent\_11, time\_spent\_12);

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

char n\_file[50];

FILE\* file;

printf("Введите имя файла для загрузки данных в него: ");

scanf("%s", n\_file);

if (sizeof(n\_file) > 50) {

printf("Имя файла превышает доступное значение (49)!\n");

return;

}

if ((file = fopen(n\_file, "w")) == NULL)

{

printf("\nНевозможно открыть для записи файл: %s\n", n\_file);

\_getch();

return;

}

fprintf(file, "+---------------------------------------------------------+\n"

"| | | | |\n"

"| | qs | shell | qsort |\n"

"| | | | |\n"

"+---+---------------+----------+---------+----------------+\n");

fprintf(file, "| \\ | %-14f| %-16f| %-13f|\n", time\_spent\_1, time\_spent\_2, time\_spent\_3);

fprintf(file, "+---------------------------------------------------------+\n");

fprintf(file, "| / | %-14f| %-16f| %-13f|\n", time\_spent\_4, time\_spent\_5, time\_spent\_6);

fprintf(file, "+---------------------------------------------------------+\n");

fprintf(file, "| /\\| %-14f| %-16f| %-13f|\n", time\_spent\_7, time\_spent\_8, time\_spent\_9);

fprintf(file, "+---------------------------------------------------------+\n");

fprintf(file, "| . | %-14f| %-16f| %-13f|\n", time\_spent\_10, time\_spent\_11, time\_spent\_12);

fprintf(file, "+---------------------------------------------------------+\n");

fclose(file);

printf("Данные успешно записаны!\n");

}