Министерство науки и высшего 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:

Уткин М. М.

Соколовский Е. В.

Саветкин Д. Д.

Приняли:

Акифьев О. В.

Юрова О. В.

Пенза 2023

**Цель работы:**

Изучение простых структур данных и массивов, получение навыков составления простейших алгоритмов.

**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.

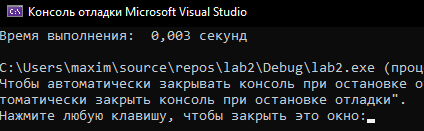
**Типы данных:**

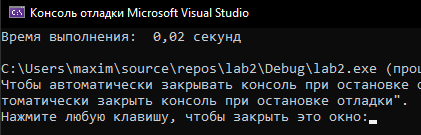
1. clock\_t - возвращается функцией clock(). Обычно определён как int или long int.

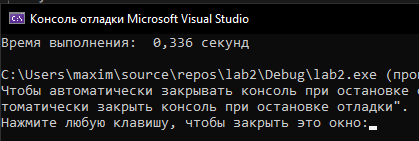
2. time\_t - возвращается функцией time(). Обычно определён как int или long int.

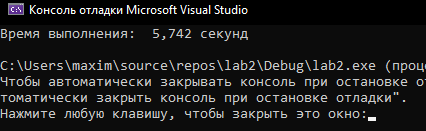
3. struct tm - нелинейное, дискретное календарное представление времени.

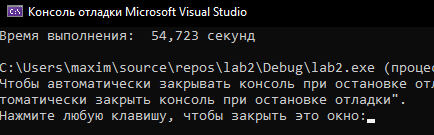
**Результаты работы программы:**

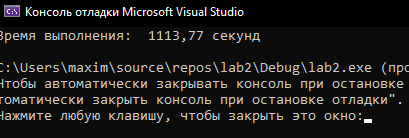


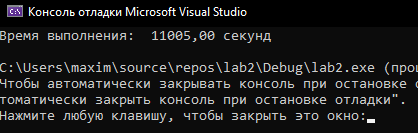




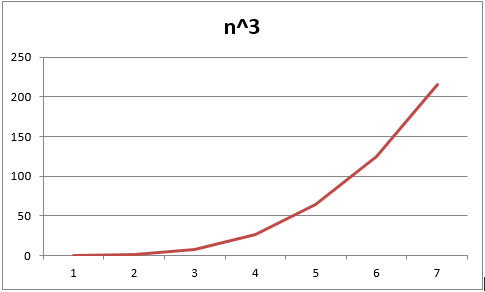




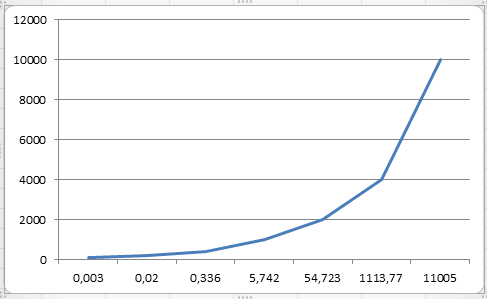




**График сложности:**

****

**График зависимости времени выполнения программы от размера матриц:**



**Вывод:** В ходе данной лабораторной работы мы определили, что сложность данного алгоритма равно O(n^3). Сложность определена верно т.к. построенные графики примерно совпадают.

**Листинг:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

double multiply\_matrices\_time(void) {

clock\_t start, end;

int i, j, r;

int size = 400;

// Выделение памяти для динамических массивов

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

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

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

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

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

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

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

}

srand(time(NULL));

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

for (j = 0; j < size; j++) {

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

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

}

}

start = clock();

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

for (j = 0; j < size; j++) {

int elem\_c = 0;

for (r = 0; r < size; r++) {

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

}

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

}

}

end = clock();

// Свободная динамически выделяемая память

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

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

return ((double)(end - start) / CLOCKS\_PER\_SEC);

}

int main(void) {

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

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

double time = multiply\_matrices\_time();

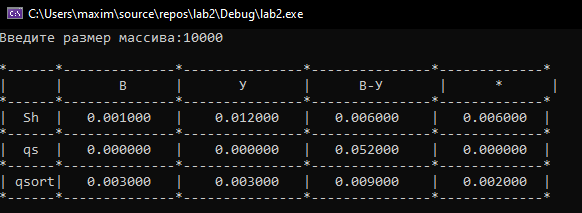
printf("Времявыполнения: %lf seconds\n", time);

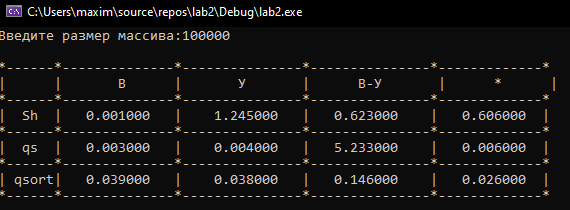
return 0;

}

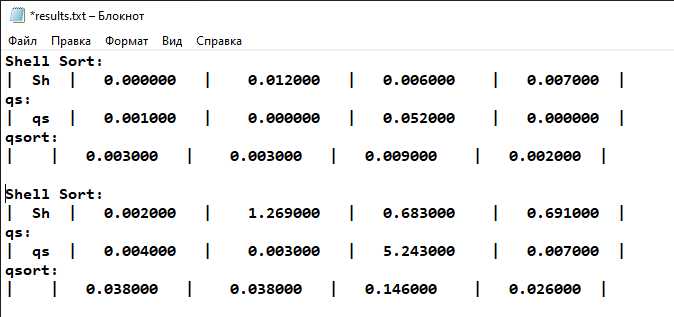
**Задание 2:**

**Результаты работы программы:**





**Реализация записи в файл результата:**



**Вывод:** В ходе работы программы мы сделали вывод, что самая быстрая функция сортировки для всех наборов данных функция qsort.

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#include <conio.h>

void VideleniePamyati(int\*\* arr, int size)

{

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

if (\*arr == NULL) {

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

return;

}

}

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 zapolnenieshell(int size)// заполнение под сортировку shell

{

int\* a = NULL;

int i;

double T1 = 0.0;

double T2 = 0.0;

double T3 = 0.0;

double T4 = 0.0;

clock\_t start, end;

FILE\* Shell1, \* Shell2, \* Shell3, \* Shell4;

//Последовательное

VideleniePamyati(&a, size);

Shell1 = fopen("ShellUp.txt", "w");

fprintf(Shell1, "До сортировки\n");

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

{

a[i] = i;

fprintf(Shell1, "%d\t", a[i]);

}

start = clock();

shell(a, size);

end = clock();

T1 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(Shell1, "\n\nПосле сортировки\n");

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

{

fprintf(Shell1, "%d\t", a[i]);

}

fclose(Shell1);

free(a);

//Убывающее

VideleniePamyati(&a, size);

Shell2 = fopen("ShellDown.txt", "w");

fprintf(Shell2, "До сортировки\n");

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

{

a[i] = size - i - 1;

fprintf(Shell2, "%d\t", a[i]);

}

start = clock();

shell(a, size);

end = clock();

T2 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(Shell2, "\n\nПосле сортировки\n");

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

{

fprintf(Shell2, "%d\t", a[i]);

}

fclose(Shell2);

free(a);

//Половинное

VideleniePamyati(&a, size);

Shell3 = fopen("ShellHalf.txt", "w");

fprintf(Shell3, "До сортировки\n");

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

{

a[i] = i;

}

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

{

a[i] = size - i - 1;

}

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

{

fprintf(Shell3, "%d\t", a[i]);

}

start = clock();

shell(a, size);

end = clock();

T3 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(Shell3, "\n\nПосле сортировки\n");

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

{

fprintf(Shell3, "%d\t", a[i]);

}

fclose(Shell3);

free(a);

//Случайное

VideleniePamyati(&a, size);

Shell4 = fopen("ShellRand.txt", "w");

fprintf(Shell4, "До сортировки\n");

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

{

a[i] = rand() % 100;

fprintf(Shell4, "%d\t", a[i]);

}

start = clock();

shell(a, size);

end = clock();

T4 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(Shell4, "\n\nПосле сортировки\n");

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

{

fprintf(Shell4, "%d\t", a[i]);

}

fclose(Shell4);

free(a);

printf("| Sh | %f | %f | %f | %f |\n", T1, T2, T3, T4);

FILE\* resultFile = fopen("results.txt", "a"); // Открываем файл для добавления (режим "a")

if (resultFile == NULL) {

printf("Не удалось открыть файл для записи результатов!\n");

return;

}

fprintf(resultFile, "Shell Sort:\n");

fprintf(resultFile, "| Sh | %f | %f | %f | %f |\n", T1, T2, T3, T4);

fclose(resultFile);

}

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

{

int stack[100];

int top = -1;

int x, y;

int j, i;

stack[++top] = left;

stack[++top] = right;

while (top >= 0) {

right = stack[top--];

left = stack[top--];

i = left; j = right;

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) {

stack[++top] = left;

stack[++top] = j;

}

if (i < right) {

stack[++top] = i;

stack[++top] = right;

}

}

}

int compare(const void\* a, const void\* b) {

return (\*(int\*)a - \*(int\*)b);

}

void zapolneniequick(int size)

{

int\* a = NULL;

int i;

double T1 = 0.0;

double T2 = 0.0;

double T3 = 0.0;

double T4 = 0.0;

clock\_t start, end;

FILE\* qs1, \* qs2, \* qs3, \* qs4;

//Последовательное

VideleniePamyati(&a, size);

qs1 = fopen("qsUp.txt", "w");

fprintf(qs1, "До сортировки\n");

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

{

a[i] = i;

fprintf(qs1, "%d\t", a[i]);

}

start = clock();

qs(a, 0, size - 1);

end = clock();

T1 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(qs1, "\n\nПосле сортировки\n");

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

{

fprintf(qs1, "%d\t", a[i]);

}

fclose(qs1);

free(a);

//Убывающее

VideleniePamyati(&a, size);

qs2 = fopen("qsDown.txt", "w");

fprintf(qs2, "До сортировки\n");

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

{

a[i] = size - i - 1;

fprintf(qs2, "%d\t", a[i]);

}

start = clock();

qs(a, 0, size - 1);

end = clock();

T2 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(qs2, "\n\nПосле сортировки\n");

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

{

fprintf(qs2, "%d\t", a[i]);

}

fclose(qs2);

free(a);

//Половинное

VideleniePamyati(&a, size);

qs3 = fopen("qsHalf.txt", "w");

fprintf(qs3, "До сортировки\n");

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

{

a[i] = i;

}

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

{

a[i] = size - i - 1;

}

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

{

fprintf(qs3, "%d\t", a[i]);

}

start = clock();

qs(a, 0, size - 1);

end = clock();

T3 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(qs3, "\n\nПосле сортировки\n");

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

{

fprintf(qs3, "%d\t", a[i]);

}

fclose(qs3);

free(a);

//Случайное

VideleniePamyati(&a, size);

qs4 = fopen("qsRand.txt", "w");

fprintf(qs4, "До сортировки\n");

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

{

a[i] = rand() % 100;

fprintf(qs4, "%d\t", a[i]);

}

start = clock();

qs(a, 0, size - 1);

end = clock();

T4 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(qs4, "\n\nПосле сортировки\n");

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

{

fprintf(qs4, "%d\t", a[i]);

}

fclose(qs4);

free(a);

printf("| qs | %f | %f | %f | %f |\n", T1, T2, T3, T4);

FILE\* resultFile = fopen("results.txt", "a"); // Открываем файл для добавления (режим "a")

if (resultFile == NULL) {

printf("Не удалось открыть файл для записи результатов!\n");

return;

}

fprintf(resultFile, "qs:\n");

fprintf(resultFile, "| qs | %f | %f | %f | %f |\n", T1, T2, T3, T4);

fclose(resultFile);

}

void zapolenieqsort(int size)//заполнение qsort

{

int\* a = NULL;

int i;

double T1 = 0.0;

double T2 = 0.0;

double T3 = 0.0;

double T4 = 0.0;

clock\_t start, end;

FILE\* qsort1, \* qsort2, \* qsort3, \* qsort4;

//Последовательное

VideleniePamyati(&a, size);

qsort1 = fopen("qsortUp.txt", "w");

fprintf(qsort1, "До сортировки\n");

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

{

a[i] = i;

fprintf(qsort1, "%d\t", a[i]);

}

start = clock();

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

end = clock();

T1 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(qsort1, "\n\nПосле сортировки\n");

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

{

fprintf(qsort1, "%d\t", a[i]);

}

fclose(qsort1);

free(a);

//Убывающее

VideleniePamyati(&a, size);

qsort2 = fopen("qsortDown.txt", "w");

fprintf(qsort2, "До сортировки\n");

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

{

a[i] = size - i - 1;

fprintf(qsort2, "%d\t", a[i]);

}

start = clock();

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

end = clock();

T2 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(qsort2, "\n\nПосле сортировки\n");

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

{

fprintf(qsort2, "%d\t", a[i]);

}

fclose(qsort2);

free(a);

//Половинное

VideleniePamyati(&a, size);

qsort3 = fopen("qsortHalf.txt", "w");

fprintf(qsort3, "До сортировки\n");

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

{

a[i] = i;

}

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

{

a[i] = size - i - 1;

}

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

{

fprintf(qsort3, "%d\t", a[i]);

}

start = clock();

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

end = clock();

T3 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(qsort3, "\n\nПосле сортировки\n");

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

{

fprintf(qsort3, "%d\t", a[i]);

}

fclose(qsort3);

free(a);

//Случайное

VideleniePamyati(&a, size);

qsort4 = fopen("qsortRand.txt", "w");

fprintf(qsort4, "До сортировки\n");

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

{

a[i] = rand() % 100;

fprintf(qsort4, "%d\t", a[i]);

}

start = clock();

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

end = clock();

T4 = ((double)(end - start)) / CLOCKS\_PER\_SEC;

fprintf(qsort4, "\n\nПосле сортировки\n");

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

{

fprintf(qsort4, "%d\t", a[i]);

}

fclose(qsort4);

free(a);

printf("| qsort| %f | %f | %f | %f |\n", T1, T2, T3, T4);

FILE\* resultFile = fopen("results.txt", "a"); // Открываем файл для добавления (режим "a")

if (resultFile == NULL) {

printf("Не удалось открыть файл для записи результатов!\n");

return;

}

fprintf(resultFile, "qsort:\n");

fprintf(resultFile, "| | %f | %f | %f | %f |\n", T1, T2, T3, T4);

fclose(resultFile);

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int size;

printf("Введите размер массива:");

scanf("%d", &size);

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

printf("| | В | У\ | В-У\ | \* |\n");

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

zapolnenieshell(size);

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

zapolneniequick(size);

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

zapolenieqsort(size);

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

getchar();

return 0;

}