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:

Кирилин М. С.

Николаев А. А.

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:**

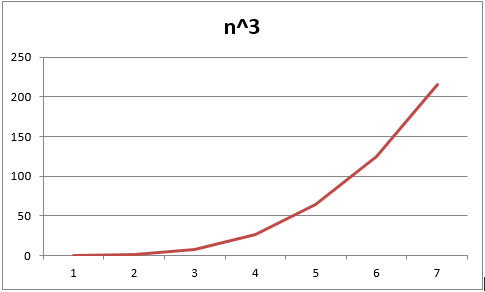
1. clock\_t - возвращается функцией clock(). Обычно определён как int или longint.

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

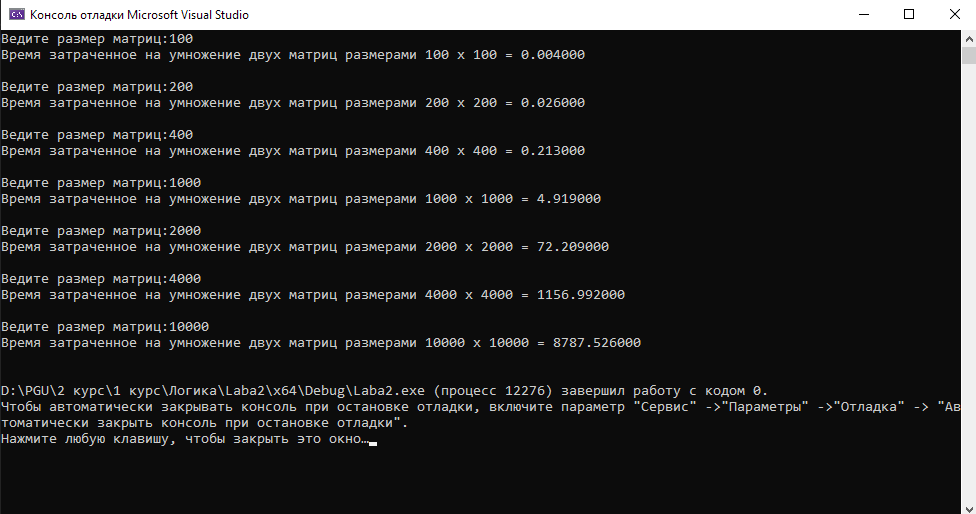
3. structtm - нелинейное, дискретное календарное представление времени.

**Задание 1:**

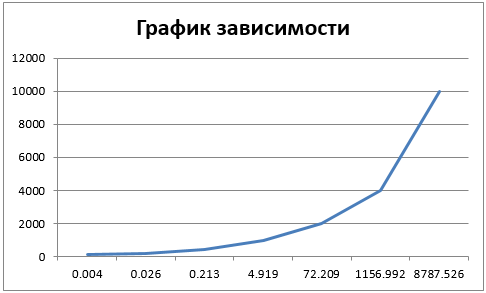
Посчитали сложность данного алгоритма, она равна n^3.

****

**Результаты работы программы:**



По результатам работы программы построили график зависимости времени от объёма данных.

****

**Вывод:** сложность данного алгоритма равно O(n^3)**. С**ложность определена верно т.к. графики примерно совпадают.

**Листинг:**

#define\_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<Windows.h>

voidmultiplications()

{

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

inti = 0, j = 0, r, size;

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

intelem\_c;

double Time = 0.0;

size = 0;

printf("Ведитеразмерматриц:");

scanf("%d", &size);

//Выделение памяти под матрицу a

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

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

{

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

}

if (a == NULL)

{

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

return;

}

//Выделение памяти под матрицу b

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

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

{

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

}

if (b == NULL)

{

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

return;

}

//Выделение памяти под матрицу c

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

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

{

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

}

if (c == NULL)

{

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

return;

}

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

while (i< size)

{

while (j < size)

{

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< size)

{

while (j <size)

{

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

j++;

}

i++;

}

start = 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();

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

printf("Время затраченное на умножение двух матриц размерами %d x %d = %f\n\n", size, size, Time);

free(a);

free(b);

free(c);

}

intmain()

{

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

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

SetConsoleCP(1251);//Установкаязыка

SetConsoleOutputCP(1251);

multiplications();

multiplications();

multiplications();

multiplications();

multiplications();

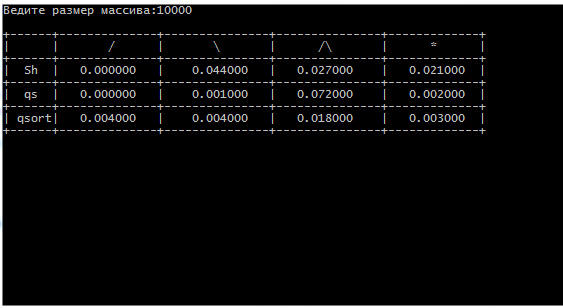
multiplications();

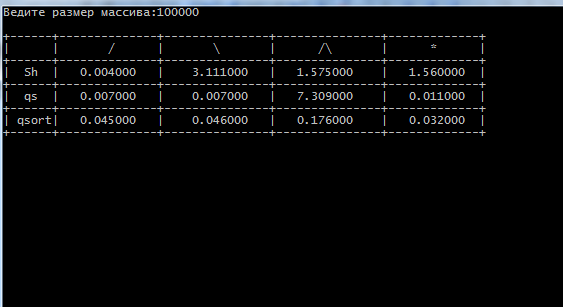
multiplications();

}

**Задание 2:**

**Результаты работы программы:**

****

****

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

**Листинг:**

**Файл Library.h:**

//#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <Windows.h>

#include <conio.h>

void MemoryAllocation(int\*\* arr, int size);

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

void fillings1(int size);

void fillings2(int size);

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

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

void fillings3(int size);

**Файл main.cpp:**

#include "Library.h"

//Функция для выделения памяти под массив

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

{

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

if (\*arr == NULL) {

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

return;

}

}

//Основная функция программы

void main()

{

SetConsoleCP(1251);//Установка языка

SetConsoleOutputCP(1251);

srand(time(NULL));//установка времени

int size;

printf("Ведите размер массива:");

scanf("%d", &size);

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

printf("| | / | \\ | /\\ | \* |\n");

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

fillings1(size);

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

fillings2(size);

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

fillings3(size);

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

getch();

}

**Файл qs.cpp:**

#include "Library.h"

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

{

// Создаем стек

int stack[100];

int top = -1;

int x, y;

int j, i;

// Добавляем начальные значения в стек

stack[++top] = left;

stack[++top] = right;

// Пока стек не пуст

while (top >= 0) {

// Извлекаем пару индексов из стека

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);

// Добавляем новые пары индексов в стек

if (left < j) {

stack[++top] = left;

stack[++top] = j;

}

if (i < right) {

stack[++top] = i;

stack[++top] = right;

}

}

}

void fillings2(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

MemoryAllocation(&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

MemoryAllocation(&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

MemoryAllocation(&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

MemoryAllocation(&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);

}

**Файл qsort.cpp:**

#include "Library.h"

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

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

}

void fillings3(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\* qsort1, \* qsort2, \* qsort3, \* qsort4;

//Последовательно

MemoryAllocation(&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

MemoryAllocation(&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

MemoryAllocation(&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

MemoryAllocation(&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);

}

**Файл Shell.cpp:**

#include "Library.h"

//Сортировка Шеллла

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 fillings1(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\* Shell1, \* Shell2, \* Shell3, \* Shell4;

//Последовательное

MemoryAllocation(&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

MemoryAllocation(&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

MemoryAllocation(&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

MemoryAllocation(&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);

}

**Вывод**

В ходе работы изучили способ оценки времени выполнения программ, познакомились с библиотекой **time.h**иизучили навыки составления алгоритмов с помощью функций clock\_t, time\_t и structtm.