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 22ВВП2:

Перкин П.О.

Широкова И.Д.

Приняли

Акифьев И.В.

Юрова О.В.

Пенза 2023

**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**

**Задание 1**:

#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;

clock\_t start1;

start = clock();

int size = 200;

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

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

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

if (a == NULL || b == NULL || c == NULL)

{

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

return 1;

}

srand(time(NULL));

for (int 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));

}

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

{

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

{

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

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

}

}

start1 = clock();

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

{

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

{

int elem\_c = 0;

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

{

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

}

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

}

}

end = clock();

printf("razmer matric:%d\n", size);

printf("program time: %f sec\n", (difftime(end, start)) / CLOCKS\_PER\_SEC);

printf("algoritm time: %f sec\n", (difftime(end, start1)) / CLOCKS\_PER\_SEC);

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

{

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

return 0;

}

**Задание 2**:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

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

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

}

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)

{

int i, j;

int x, y;

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)

qs(items, left, j);

if (i < right)

qs(items, i, right);

}

void generateRandomArray(int\* arr, int size)

{

srand(time(NULL));

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

{

arr[i] = rand() % 1000;

}

}

void generateIncreasingArray(int\* arr, int size)

{

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

{

arr[i] = i;

}

}

void generateDecreasingArray(int\* arr, int size)

{

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

{

arr[i] = size - i - 1;

}

}

void generateIncreasingDecreasingArray(int\* arr, int size)

{

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

{

arr[i] = i;

}

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

{

arr[i] = size - i - 1;

}

}

int main()

{

setlocale(LC\_ALL, "Rus");

int n;

clock\_t start\_time;

clock\_t end\_time;

printf("Введите размер массива: ");

scanf("%d", &n);

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

printf("Случайный массив\n\n");

generateRandomArray(arr, n);

start\_time = clock();

shell(arr, n);

end\_time = clock();

printf(" Shell Sort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

generateRandomArray(arr, n);

start\_time = clock();

qs(arr, 0, n - 1);

end\_time = clock();

printf(" Quick Sort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

generateRandomArray(arr, n);

start\_time = clock();

qsort(arr, n, sizeof(int), compare);

end\_time = clock();

printf(" qsort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

printf("\nВозрастающий массив\n");

generateIncreasingArray(arr, n);

start\_time = clock();

shell(arr, n);

end\_time = clock();

printf(" Shell Sort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

generateIncreasingArray(arr, n);

start\_time = clock();

qs(arr, 0, n - 1);

end\_time = clock();

printf(" Quick Sort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

generateIncreasingArray(arr, n);

start\_time = clock();

qsort(arr, n, sizeof(int), compare);

end\_time = clock();

printf(" qsort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

printf("\nУбывающий массив\n");

generateDecreasingArray(arr, n);

start\_time = clock();

shell(arr, n);

end\_time = clock();

printf(" Shell Sort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

generateDecreasingArray(arr, n);

start\_time = clock();

qs(arr, 0, n - 1);

end\_time = clock();

printf(" Quick Sort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

generateDecreasingArray(arr, n);

start\_time = clock();

qsort(arr, n, sizeof(int), compare);

end\_time = clock();

printf(" qsort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

printf("\nМассив с первой половиной возрастающей и второй половиной убывающей\n");

generateIncreasingDecreasingArray(arr, n);

start\_time = clock();

shell(arr, n);

end\_time = clock();

printf(" Shell Sort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

generateIncreasingDecreasingArray(arr, n);

start\_time = clock();

qsort(arr, n, sizeof(int), compare);

end\_time = clock();

printf(" qsort: %f секунд\n", (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC);

generateIncreasingDecreasingArray(arr, n);

start\_time = clock();

qs(arr, 0, n - 1);

end\_time = clock();

double elapsed\_time = difftime(end\_time, start\_time) / CLOCKS\_PER\_SEC;

printf(" Quick Sort: %f секунд\n", elapsed\_time);

return 0;

}

**Задание 1**

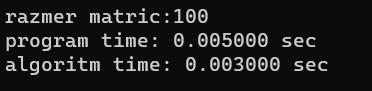
1. Сложность О(N^3)

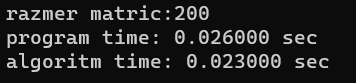
|  |  |
| --- | --- |
| Размер матрицы | Время алгоритма |
| 100 | 0,003 |
| 200 | 0,02 |
| 400 | 0,176 |
| 1000 | 4,02 |
| 2000 | 37,450 |
| 4000 | 340,23 |
| 10000 | 3195,208 |

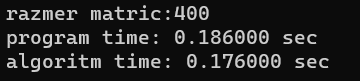


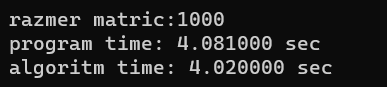
Сравнение результатов с теоретической оценкой сложности показывает, что временная сложность программы при увеличении размера матрицы size действительно соответствует теоретической оценке O(size^3). При увеличении размера матрицы в 10 раз (например, от 100 до 1000 элементов), время выполнения увеличивается примерно в 1000 раз, что соответствует кубической зависимости .

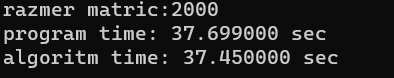
**Результаты работы программы**





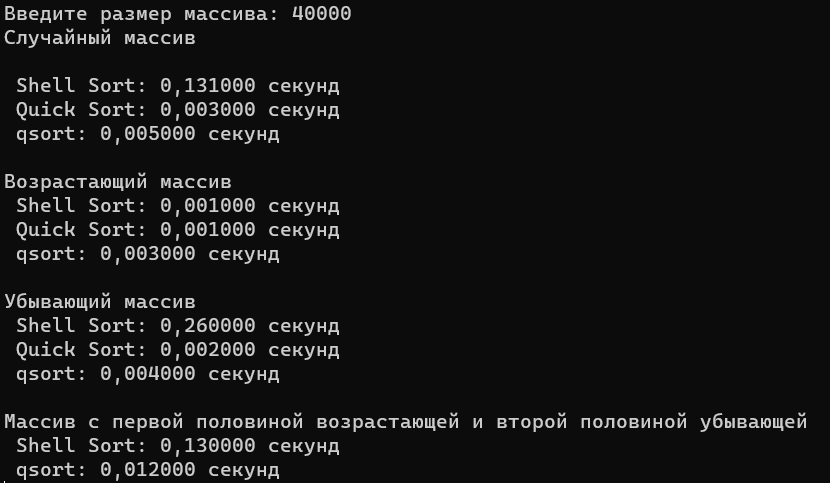






**Задание 2**

**Результаты работы программы**



**Вывод**

В рамках лабораторной работы была проведена оценка времени выполнения программы с использованием алгоритмов сортировки и перемножения матриц.

В ходе выполнения первой части лабораторной работы было проведено исследование по оценке времени выполнения программы. Был выполнен анализ порядка сложности порядка и проведено сравнение полученных результатов с теоретической оценкой.

Во второй части работы было проведено исследование производительности алгоритмов сортировки (Shell Sort, Quick Sort, qsort) на различных типах входных данных. Результаты экспериментов позволили сделать выводы о преимуществах и недостатках каждого из алгоритмов на разных типах данных и размерах массивов.

Итоговый анализ данных позволяет сделать вывод о том, что выбор алгоритма сортировки и оценка времени выполнения программы важны для оптимизации производительности программ.