Пензенский государственный университет

Факультет вычислительной техники

Кафедра «Вычислительная техника»

**Отчет**

по лабораторной работе №2

по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»

на тему: «Оценка времени выполнения программ»

**Выполнил студент группы 23ВВВ1:**

Евстифеев Д.А

**Приняли:**

Митрохин М. А.

Юрова О. В.

**Пенза 2024**

**Название**

Оценка времени выполнения программ

**Цель работы**

Выполнить поставленные задачи в соответствии с требованиями

**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**

**Задание 1**:   
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.

**Листинг А:**

#include <stdio.h>

#include <limits.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

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, j, r;

int n = 100; // размер матрицы

// динамическое выделение памяти для матриц

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

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

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

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

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

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

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

}

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

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

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

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

}

}

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

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

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

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

}

}

start = clock(); // начало измерения времени

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

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

int elem\_c = 0;

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

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

}

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

}

}

end = clock(); // конец измерения времени

float t = (float)(end - start) / (float)(CLOCKS\_PER\_SEC);

printf("time: %f\n", t);

// освобождение выделенной памяти

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

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

getchar();

return 0;

}

Результат выполнения:

При n = 200:

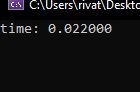


Рис. 1 ­— время выполнения при n = 200

При n = 400:

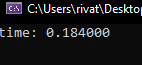


Рис. 2 ­— время выполнения при n = 400

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

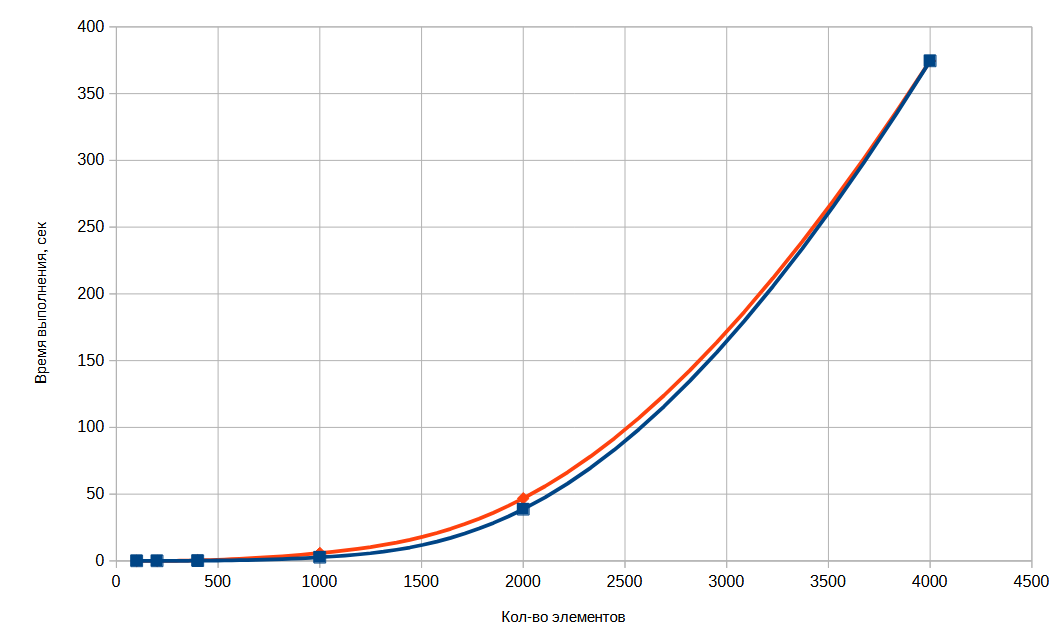


Рис. 3 ­— таблица для задания 1

Cиний цвет - полученные данные при тестировании программы.

Красный - ветка, отображающая теоретические расчёты сложности алгоритма при введённых исходных данных.

**Задание 2**:

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

**Листинг Б:**

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

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);

}

int main() {

int count = 10000; // размер массива

int\* items = (int\*)malloc(count \* sizeof(int));

int i;

// генерация случайного массива

srand(time(NULL));

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

items[i] = rand();

}

// копирование массива для сортировки Шелла

int\* items\_shell = (int\*)malloc(count \* sizeof(int));

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

items\_shell[i] = items[i];

}

// копирование массива для быстрой сортировки

int\* items\_qs = (int\*)malloc(count \* sizeof(int));

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

items\_qs[i] = items[i];

}

// измерение времени для сортировки Шелла

clock\_t start\_shell = clock();

shell(items\_shell, count);

clock\_t end\_shell = clock();

float time\_shell = (float)(end\_shell - start\_shell) / (float)(CLOCKS\_PER\_SEC);

// измерение времени для быстрой сортировки

clock\_t start\_qs = clock();

qs(items\_qs, 0, count - 1);

clock\_t end\_qs = clock();

float time\_qs = (float)(end\_qs - start\_qs) / (float)(CLOCKS\_PER\_SEC);

// Вывод результатов

printf("time\_shell\_sort: %f\n", time\_shell);

printf("time\_quick\_sort: %f\n", time\_qs);

// освобождение памяти

free(items);

free(items\_shell);

free(items\_qs);

getchar();

return 0;

}

Результат выполнения:

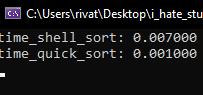


Рис. 4 ­— время выполнения при 10000

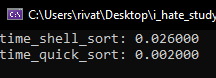


Рис. 5 ­— время выполнения при 20000

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

**Листинг В:**

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

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);

}

int main() {

int count = 100000; // размер массива

int\* items = (int\*)malloc(count \* sizeof(int));

int i;

// генерация возрастающего массива

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

items[i] = i;

}

// копирование массива для сортировки Шелла

int\* items\_shell = (int\*)malloc(count \* sizeof(int));

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

items\_shell[i] = items[i];

}

// копирование массива для быстрой сортировки

int\* items\_qs = (int\*)malloc(count \* sizeof(int));

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

items\_qs[i] = items[i];

}

// измерение времени для сортировки Шелла

clock\_t start\_shell = clock();

shell(items\_shell, count);

clock\_t end\_shell = clock();

float time\_shell = (float)(end\_shell - start\_shell) / (float)(CLOCKS\_PER\_SEC);

// измерение времени для быстрой сортировки

clock\_t start\_qs = clock();

qs(items\_qs, 0, count - 1);

clock\_t end\_qs = clock();

float time\_qs = (float)(end\_qs - start\_qs) / (float)(CLOCKS\_PER\_SEC);

// Вывод результатов

printf("time\_shell\_sort: %f\n", time\_shell);

printf("time\_quick\_sort: %f\n", time\_qs);

// освобождение памяти

free(items);

free(items\_shell);

free(items\_qs);

getchar();

return 0;

}

Результат выполнения:

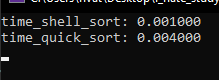


Рис. 6 ­— время выполнения при 100000

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

**Листинг Г:**

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

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);

}

int main() {

int count = 100000; // размер массива

int\* items = (int\*)malloc(count \* sizeof(int));

int i;

// генерация убывающего массива

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

items[i] = count - i - 1;

}

// копирование массива для сортировки Шелла

int\* items\_shell = (int\*)malloc(count \* sizeof(int));

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

items\_shell[i] = items[i];

}

// копирование массива для быстрой сортировки

int\* items\_qs = (int\*)malloc(count \* sizeof(int));

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

items\_qs[i] = items[i];

}

// измерение времени для сортировки Шелла

clock\_t start\_shell = clock();

shell(items\_shell, count);

clock\_t end\_shell = clock();

float time\_shell = (float)(end\_shell - start\_shell) / (float)(CLOCKS\_PER\_SEC);

// измерение времени для быстрой сортировки

clock\_t start\_qs = clock();

qs(items\_qs, 0, count - 1);

clock\_t end\_qs = clock();

float time\_qs = (float)(end\_qs - start\_qs) / (float)(CLOCKS\_PER\_SEC);

// Вывод результатов

printf("time\_shell\_sort: %f\n", time\_shell);

printf("time\_quick\_sort: %f\n", time\_qs);

// освобождение памяти

free(items);

free(items\_shell);

free(items\_qs);

getchar();

return 0;

}

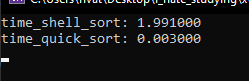


Рис. 6 ­— время выполнения при 100000

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

**Листинг Г:**

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

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;

}

}

}

int medianOfThree(int\* items, int left, int right) {

int mid = (left + right) / 2;

if (items[left] > items[mid]) {

int temp = items[left];

items[left] = items[mid];

items[mid] = temp;

}

if (items[left] > items[right]) {

int temp = items[left];

items[left] = items[right];

items[right] = temp;

}

if (items[mid] > items[right]) {

int temp = items[mid];

items[mid] = items[right];

items[right] = temp;

}

return mid;

}

void qs(int\* items, int left, int right) //вызов функции: qs(items, 0, count-1);

{

int i, j;

int x, y;

if (left < right) {

int pivotIndex = medianOfThree(items, left, right);

x = items[pivotIndex];

i = left; j = right;

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);

}

}

int main() {

int count = 100000; // размер массива

int\* items = (int\*)malloc(count \* sizeof(int));

int i;

// генерация массива, одна половина которого возрастающая, а другая убывающая

for (i = 0; i < count / 2; i++) {

items[i] = i;

}

for (i = count / 2; i < count; i++) {

items[i] = count - i - 1;

}

// копирование массива для сортировки Шелла

int\* items\_shell = (int\*)malloc(count \* sizeof(int));

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

items\_shell[i] = items[i];

}

// копирование массива для быстрой сортировки

int\* items\_qs = (int\*)malloc(count \* sizeof(int));

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

items\_qs[i] = items[i];

}

// измерение времени для сортировки Шелла

clock\_t start\_shell = clock();

shell(items\_shell, count);

clock\_t end\_shell = clock();

float time\_shell = (float)(end\_shell - start\_shell) / (float)(CLOCKS\_PER\_SEC);

// измерение времени для быстрой сортировки

clock\_t start\_qs = clock();

qs(items\_qs, 0, count - 1);

clock\_t end\_qs = clock();

float time\_qs = (float)(end\_qs - start\_qs) / (float)(CLOCKS\_PER\_SEC);

// Вывод результатов

printf("time\_shell\_sort: %f\n", time\_shell);

printf("time\_quick\_sort: %f\n", time\_qs);

// освобождение памяти

free(items);

free(items\_shell);

free(items\_qs);

getchar();

return 0;

}



Рис. 6 ­— время выполнения при 100000

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

**Листинг Д:**

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

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;

}

}

}

int medianOfThree(int\* items, int left, int right) {

int mid = (left + right) / 2;

if (items[left] > items[mid]) {

int temp = items[left];

items[left] = items[mid];

items[mid] = temp;

}

if (items[left] > items[right]) {

int temp = items[left];

items[left] = items[right];

items[right] = temp;

}

if (items[mid] > items[right]) {

int temp = items[mid];

items[mid] = items[right];

items[right] = temp;

}

return mid;

}

void qs(int\* items, int left, int right) //вызов функции: qs(items, 0, count-1);

{

int i, j;

int x, y;

if (left < right) {

int pivotIndex = medianOfThree(items, left, right);

x = items[pivotIndex];

i = left; j = right;

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);

}

}

int main() {

int count = 100000; // размер массива

int\* items = (int\*)malloc(count \* sizeof(int));

int i;

// генерация массива, одна половина которого возрастающая, а другая убывающая

for (i = 0; i < count / 2; i++) {

items[i] = i;

}

for (i = count / 2; i < count; i++) {

items[i] = count - i - 1;

}

// копирование массива для сортировки Шелла

int\* items\_shell = (int\*)malloc(count \* sizeof(int));

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

items\_shell[i] = items[i];

}

// копирование массива для быстрой сортировки

int\* items\_qs = (int\*)malloc(count \* sizeof(int));

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

items\_qs[i] = items[i];

}

// измерение времени для сортировки Шелла

clock\_t start\_shell = clock();

shell(items\_shell, count);

clock\_t end\_shell = clock();

float time\_shell = (float)(end\_shell - start\_shell) / (float)(CLOCKS\_PER\_SEC);

// измерение времени для быстрой сортировки

clock\_t start\_qs = clock();

qs(items\_qs, 0, count - 1);

clock\_t end\_qs = clock();

float time\_qs = (float)(end\_qs - start\_qs) / (float)(CLOCKS\_PER\_SEC);

// Вывод результатов

printf("time\_shell\_sort: %f\n", time\_shell);

printf("time\_quick\_sort: %f\n", time\_qs);

// освобождение памяти

free(items);

free(items\_shell);

free(items\_qs);

getchar();

return 0;

}

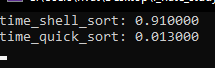


Рис. 6 ­— время выполнения при 100000

5. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

**Листинг Е:**

#include <stdio.h>

#include <windows.h>

#include <time.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

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

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

}

void test\_qsort(int\* items, int count) {

// копирование массива для стандартной функции qsort

int\* items\_qsort = (int\*)malloc(count \* sizeof(int));

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

items\_qsort[i] = items[i];

}

// измерение времени для стандартной функции qsort

clock\_t start\_qsort = clock();

qsort(items\_qsort, count, sizeof(int), compare);

clock\_t end\_qsort = clock();

float time\_qsort = (float)(end\_qsort - start\_qsort) / (float)(CLOCKS\_PER\_SEC);

// Вывод результатов

printf("time\_qsort: %f\n", time\_qsort);

// освобождение памяти

free(items\_qsort);

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int count = 100000; // размер массива

int\* items = (int\*)malloc(count \* sizeof(int));

int i;

// случайный массив

srand(time(NULL));

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

items[i] = rand();

}

printf("случайный массив:\n");

test\_qsort(items, count);

// возрастающий массив

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

items[i] = i;

}

printf("\nвозрастающий массив:\n");

test\_qsort(items, count);

// убывающий массив

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

items[i] = count - i - 1;

}

printf("\nубывающий массив:\n");

test\_qsort(items, count);

// массив, одна половина которого возрастающая, а другая убывающая

for (i = 0; i < count / 2; i++) {

items[i] = i;

}

for (i = count / 2; i < count; i++) {

items[i] = count - i - 1;

}

printf("\nмассив, одна половина которого возрастающая, а другая убывающая:\n");

test\_qsort(items, count);

// освобождение памяти

free(items);

getchar();

return 0;

}

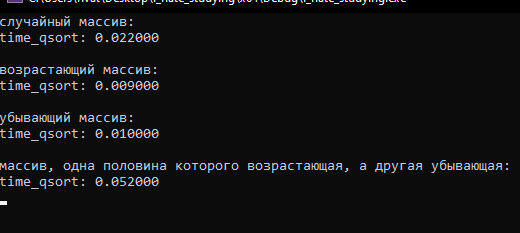


Рис. 7 ­— время выполнения при 100000

**Вывод:** в ходе работы было установлено, что время выполнения операций с матрицами значительно возрастает с увеличением их размера, что свидетельствует о росте вычислительных затрат при работе с большими данными. В экспериментах по сортировке массивов различные алгоритмы проявили себя по-разному в зависимости от структуры исходных данных. На случайных и частично упорядоченных массивах эффективнее показала себя быстрая сортировка, тогда как встроенный алгоритм qsort продемонстрировал стабильность для различных видов массивов.