

Министерство науки и высшего образования РФ  
Пензенский государственный университет  
Кафедра «Вычислительная техника»

**Отчет**  
по лабораторной работе № 2  
по курсу «Арифметические и логические основы вычислительной  
техники»  
на тему «Оценка времени выполнения программ»

Выполнили  
студенты группы 24ВВВ2:

Макаров Я.С.

Бегичев Д.Д.

Родионов К.Е.

Приняли

Юрова О.В.

Деев М.В.

Пенза, 2025

## Цель работы

Провести исследование эффективности различных алгоритмов и операций (перемножения матриц и сортировки массивов) путем оценки их временной сложности с использованием теоретических моделей (O-символика), измерения реального времени выполнения на различных входных данных (матрицы и массивы разного размера и структуры), а также визуализации результатов для сопоставления теории с практикой

## Лабораторное задание

1. Проанализировать алгоритм перемножения матриц и определить его временную сложность в терминах O-символики.
2. Реализовать программу на языке C, которая:
  - Выделяет динамическую память под две входные матрицы и одну матрицу результата.
  - Инициализирует входные матрицы случайными числами.
  - Выполняет умножение матриц классическим алгоритмом.
  - Измеряет время выполнения умножения с помощью функций из библиотеки `time.h`.
3. Выполнить замеры времени работы программы для матриц размером 100, 200, 400, 1000, 2000, 4000, 10000.
4. Построить график зависимости времени выполнения умножения матриц от их размера.
5. Сравнить экспериментальные результаты с теоретической оценкой сложности  $O(N^3)$  и сделать выводы о соответствии времени выполнения теоретической модели.

## Описание метода решения

Для оценки временной сложности алгоритма анализируют количество операций в зависимости от размера входных данных  $N$ . Подсчитывают шаги во вложенных циклах, выделяют главный член с наибольшим ростом (например,  $N^3$  для тройного цикла) и обозначают сложность как  $O(N^3)$ , игнорируя константы и менее значимые слагаемые.

Потом измеряют реальное время выполнения программы с помощью функций из `time.h` для разных размеров данных и сравнивают экспериментальные результаты с теоретической оценкой.

**Выделение памяти** - операции `malloc` выполняются константное количество раз (3 раза), временная сложность —  $O(1)$ .

### Первый двойной цикл:

```
for (i = 0; i < N; i++) {  
    for (j = 0; j < N; j++) {  
        a[i * N + j] = rand() % 10;  
        b[i * N + j] = rand() % 10;  
        f[i * N + j] = 0;  
    }  
}
```

Здесь выполняется  $N \times N = N^2$  операций - инициализация трех массивов за каждую итерацию. Сложность —  $O(N^2)$ .

### Тройной вложенный цикл перемножения матриц:

```
for (i = 0; i < N; i++) {  
    for (j = 0; j < N; j++) {  
        c = 0;
```

```

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

    c += a[i * N + r] * b[r * N + j];

}

f[i * N + j] = c;

}

}

```

Этот участок кода - классический алгоритм перемножения матриц. Вложенные циклы  $i$ ,  $j$ ,  $r$  все идут от 0 до  $N$ , итого операций порядка  $N \times N \times N = N^3$ . Сложность —  $O(N^3)$ .

## Псевдокод

### 1-ое задание

дано

две матрицы  $a$  и  $b$  размера  $N \times N$

результат умножения в матрицу  $f$  того же размера

надо

- вычислить произведение матриц  $f = a \times b$

нач цел

матрицы  $a$ ,  $b$ ,  $f$  размером  $N \times N$

переменные  $i, j, r, c$ , время

инициализировать генератор случайных чисел

нц для  $i$  от 0 до  $N-1$

нц для  $j$  от 0 до  $N-1$

$a[i, j] :=$  случайное число от 0 до 9

$b[i, j] :=$  случайное число от 0 до 9

$f[i, j] := 0$

кц

кц

записать время начала вычислений

нц для  $i$  от 0 до  $N-1$

нц для  $j$  от 0 до  $N-1$

$c := 0$

```
нц для r от 0 до N-1
  c := c + a[i, r] * b[r, j]
кц
f[i, j] := c
```

кц

кц

записать время окончания вычислений

вычислить время выполнения как разницу времени конца и начала

вывести первые 10 строк и 10 столбцов матрицы f

вывести время выполнения

кон

2-ое задание

алг сравнение\_сортировок

дано

массивы inc (возрастающий), dec (убывающий), random (случайный),  
korn (угловатый) размером 100

процедуры сортировок: Shell и Быстрая

надо измерить и вывести время работы каждой сортировки на каждом массиве

нач цел

массивы inc, dec, random, korn, temp

время timed

переменные i, size

// Копирование массива

процедура копировать(откуда, куда, размер)

нц для i от 0 до размер-1

куда[i] := откуда[i]

кц

конец процедуры

// Сортировка Шелла

функция Shell(массива arr, размер n)

измерить старт времени

нц для gap от n/2 до 1 с шагом деления gap на 2

нц для i от gap до n-1

temp := arr[i]

j := i

пока j ≥ gap и arr[j - gap] > temp

arr[j] := arr[j - gap]

```

        j := j - gap
    конец пока
    arr[j] := temp
кц
кц
измерить конец времени
вернуть время работы
конец функции

// Быстрая сортировка (QuickSort)
процедура quick_sort(массива arr, low, high)
если low ≥ high тогда вернуть
pivot := arr[high]
i := low - 1
нц для j от low до high-1
    если arr[j] ≤ pivot тогда
        i := i + 1
        поменять arr[i] и arr[j]
    конец если
кц
i := i + 1
поменять arr[i] и arr[high]
вызвать quick_sort(arr, low, i-1)
вызвать quick_sort(arr, i+1, high)
конец процедуры

функция quick(массива arr, low, high)
измерить старт времени
вызвать quick_sort(arr, low, high)
измерить конец времени
вернуть время работы
конец функции

// Основной блок
инициализировать генератор случайных чисел
заполнить массивы inc, dec, random, korn согласно условиям

для каждого массива из {random, inc, dec, korn} делать
    размер := длина массива
    копировать(исходный, temp, размер)
    timed := Shell(массив, размер)
    вывести "Время работы сортировки Шелла на массиве:", timed
    копировать(temp, массив, размер)
    timed := quick(массив, 0, размер-1)
    вывести "Время работы быстрой сортировки на массиве:", timed
конец для

```

КОН

## Листинг

### 1-ое задание

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


#define N 100 // Спросите почему одномерный? Потому что malloc
читает массив как одномерный двух мерного нет

int main() {

    int i, j, r, c;

    int *a, *b, *f;


    // выделение памяти под массив

    a = (int*)malloc(N * N * sizeof(int));
    b = (int*)malloc(N * N * sizeof(int));
    f = (int*)malloc(N * N * sizeof(int));


    srand(time(NULL));

    clock_t start = clock();


    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            a[i * N + j] = rand() % 10; // [i * N + J] - по
формеле как константа
            b[i * N + j] = rand() % 10;
            f[i * N + j] = 0;
        }
    }


    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
```

```

        c = 0;
        for (r = 0; r < N; r++) {
            c += a[i * N + r] * b[r * N + j];
        }
        f[i * N + j] = c;
    }

}

clock_t end = clock();

double time = (double)(end - start) / CLOCKS_PER_SEC;

printf("10 el:\n"); // матрица 10 на 10, в задании есть
10000 на 10000, считать будешь вечность

for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++) {
        printf("%d ", f[i * N + j]);
    }
    printf("\n");
}

printf("time: %.10f", time);

free(a);
free(b);
free(f);

return 0;
}

```



## 2-ое задание

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>

double Shell(int arr[], int n) {
    LARGE_INTEGER frequency, start, end;
    QueryPerformanceFrequency(&frequency);
    QueryPerformanceCounter(&start);
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -=
gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
    QueryPerformanceCounter(&end);
    double time_spent = (double)(end.QuadPart - start.QuadPart)
/ frequency.QuadPart;
    return time_spent;
}

void quick_sort(int arr[], int low, int high) {
    if (low >= high) return;

    int pivot = arr[high];
    int i = low - 1;
```

```

        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        i++;
        int temp = arr[i];
        arr[i] = arr[high];
        arr[high] = temp;

        quick_sort(arr, low, i - 1);
        quick_sort(arr, i + 1, high);
    }

double quick(int arr[], int low, int high) {
    LARGE_INTEGER frequency, start, end;
    QueryPerformanceFrequency(&frequency);
    QueryPerformanceCounter(&start);

    quick_sort(arr, low, high);

    QueryPerformanceCounter(&end);

    return (double)(end.QuadPart - start.QuadPart) /
frequency.QuadPart;
}

int copy(int arr[], int temp[]) {
    int n = sizeof(arr) / sizeof(arr[0]);

```

```

        for (int i = 0; i < n; i++) {
            temp[i] = arr[i];
        }
    }

int main() {
    system("chcp 1251");
    system("cls");
    srand(time(NULL));

    int inc[100], dec[100], random[100], korn[100], temp[100];
    double timed;

    for (int i = 0; i < 100; i++) {
        inc[i] = i + 1;
        dec[i] = 100 - i;
        random[i] = rand() % 100;
        if (i < 48) {
            korn[i] = i + 1;
        }
        else {
            korn[i] = 100 - i;
        }
    }

    int size1 = sizeof(inc) / sizeof(inc[0]);
    int size2 = sizeof(dec) / sizeof(dec[0]);
    int size3 = sizeof(random) / sizeof(random[0]);
    int size4 = sizeof(korn) / sizeof(korn[0]);
    int size5 = sizeof(temp) / sizeof(temp[0]);

    copy(random, temp);

    timed = Shell(random, size3);

    printf("Время работы сортировки Шелла на случайном массиве:
    %lf\n", timed);

    copy(temp, random);

    timed = quick(random, 0, size3 - 1);

```

```
    printf("Время работы быстрой сортировки на случайном массиве: %lf\n", timed);

    copy(inc, random);

    timed = Shell(inc, size1);

    printf("Время работы сортировки Шелла на возрастающем массиве: %lf\n", timed);

    copy(temp, inc);

    timed = quick(inc, 0, size1 - 1);

    printf("Время работы быстрой сортировки на возрастающем массиве: %lf\n", timed);

    copy(dec, temp);

    timed = Shell(dec, size2);

    printf("Время работы сортировки Шелла на убывающем массиве: %lf\n", timed);

    copy(temp, dec);

    timed = quick(dec, 0, size2 - 1);

    printf("Время работы быстрой сортировки на убывающем массиве: %lf\n", timed);

    copy(korn, temp);

    timed = Shell(korn, size4);

    printf("Время работы сортировки Шелла на угловатом массиве: %lf\n", timed);

    copy(temp, korn);

    timed = quick(korn, 0, size4 - 1);

    printf("Время работы быстрой сортировки на угловатом массиве: %lf\n", timed);

}
```

# Результат работы программы

## 1-ое задание

100

```
Консоль отладки Microsoft Visual Studio
10 el:
1947 2066 1936 1899 1999 2033 2219 2057 2053 1942
1900 1979 1875 1908 1969 1844 1894 1851 1994 1520
1973 2053 1841 2090 2138 2135 2038 2210 2088 1891
2099 2118 1899 2130 2232 2261 2087 1947 2175 1994
1862 1912 1782 1968 1766 1983 1873 1951 1809 1638
1694 1748 1726 1843 1728 1953 1739 1675 1757 1526
1837 2087 1881 1821 1950 2000 1987 1846 1982 1626
1818 1900 1786 1812 1908 2015 1937 1868 1926 1620
1963 2187 1904 2037 1952 2050 1970 1911 2041 1878
2083 2167 1946 2022 1930 2129 1970 1975 2075 1703
time: 0.010000000
C:\Users\test\source\repos\ConsoleApplication4\x64\Debug\ConsoleApplication4.exe (процесс 2700) завершил работу с кодом
0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

200

```
Консоль отладки Microsoft Visual Studio
10 el:
3927 3673 4495 4300 4005 4029 4000 3926 3986 4483
3885 3661 4361 4113 4291 4138 4064 3950 4286 4429
3851 3706 4397 4388 4492 4016 4081 3993 4334 4275
3441 3204 3792 3706 3688 3559 3617 3500 3876 3824
3416 3550 4159 3981 4213 3645 3492 3656 4054 3885
3591 3441 3953 3787 4097 3566 3822 3722 3688 4052
3628 3545 4148 4143 4137 3737 3929 3992 3959 4056
3827 3825 4648 4363 4371 4271 4194 4201 4339 4515
3888 3622 4195 3944 3885 4148 3868 4073 4243 4144
3671 3720 4116 4031 4137 3845 3548 3745 3941 4131
time: 0.019000000
C:\Users\test\source\repos\ConsoleApplication4\x64\Debug\ConsoleApplication4.exe (процесс 3148) завершил работу с кодом
0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

400

```
Консоль отладки Microsoft Visual Studio
10 el:
8144 8274 7880 7712 7340 7184 7931 8015 7630 7841
8461 8227 8496 8126 8147 7521 8643 8372 8145 8260
7862 8159 8128 7921 7583 7732 8145 7913 7970 7738
7818 7976 8109 7802 7276 7406 8059 7824 7786 7530
8683 8648 8811 8276 7983 8019 8836 8718 8660 8523
9182 8815 8983 8515 9045 7979 8745 8893 8741 8873
8223 7765 7929 8022 7488 7448 7888 8166 7987 7660
8666 8659 8675 8503 8195 8357 8979 8557 8384 8673
8639 8387 8640 8480 8082 7946 8711 8651 8526 8378
7812 7848 7905 7515 7142 7238 8165 7787 7605 7802
time: 0.168000000
C:\Users\test\source\repos\ConsoleApplication4\x64\Debug\ConsoleApplication4.exe (процесс 9380) завершил работу с кодом
0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

1000

```
Консоль отладки Microsoft Visual Studio
10 el:
19157 19594 19768 20478 19602 20191 19505 20092 19249 19363
19286 19108 19643 19523 19379 20433 19513 19170 18796 19151
18586 19207 19520 18974 18467 19506 18967 19629 18668 18691
19397 21052 20571 19847 19877 21221 20025 20435 19874 19587
19824 20360 20127 19983 20024 20998 19963 20205 19991 19871
19992 20023 19964 20165 19439 21175 19341 20159 19600 19873
20053 20451 20183 20095 19868 20893 19973 20680 19735 20410
19943 20262 20528 20178 19485 20529 19390 20609 19403 19377
19079 19382 19752 19575 19172 20266 19368 19695 18818 19389
18759 20212 19849 19778 18818 20558 19626 20319 19435 19880
time: 3.416000000
C:\Users\test\source\repos\ConsoleApplication4\x64\Debug\ConsoleApplication4.exe (процесс 14216) завершил работу с кодом
0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

2000

```
Консоль отладки Microsoft Visual Studio
10 el:
39747 39619 39697 39913 40748 40155 40585 40834 40546 41006
40146 39305 39448 40011 40484 39897 41118 40725 40522 40809
40076 40630 41030 40839 41748 40317 41744 41117 41413 41392
40810 39833 41545 40917 42114 41021 41708 41627 41789 41488
40433 39882 40139 39970 41705 40623 41062 40814 40771 41227
40147 39638 40061 39406 40758 40604 40581 40782 40904 41432
40321 39924 40273 39970 41420 40220 41610 40829 41184 40797
39222 38793 39816 39676 41101 39687 40833 40163 40001 40510
39630 39115 39410 40441 40775 39225 40757 40594 39735 40344
39619 39633 40393 38970 40990 39697 40558 40028 40220 40559
time: 32.090000000
C:\Users\test\source\repos\ConsoleApplication4\x64\Debug\ConsoleApplication4.exe (процесс 7056) завершил работу с кодом
0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

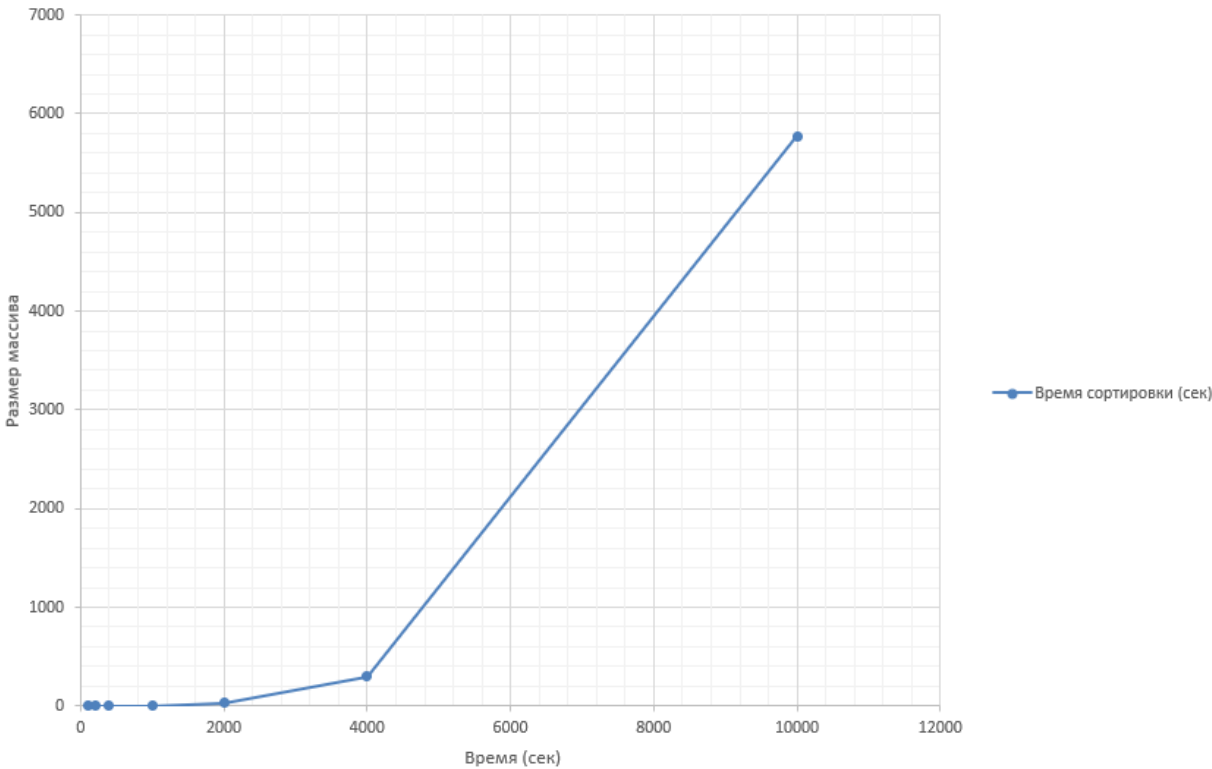
4000

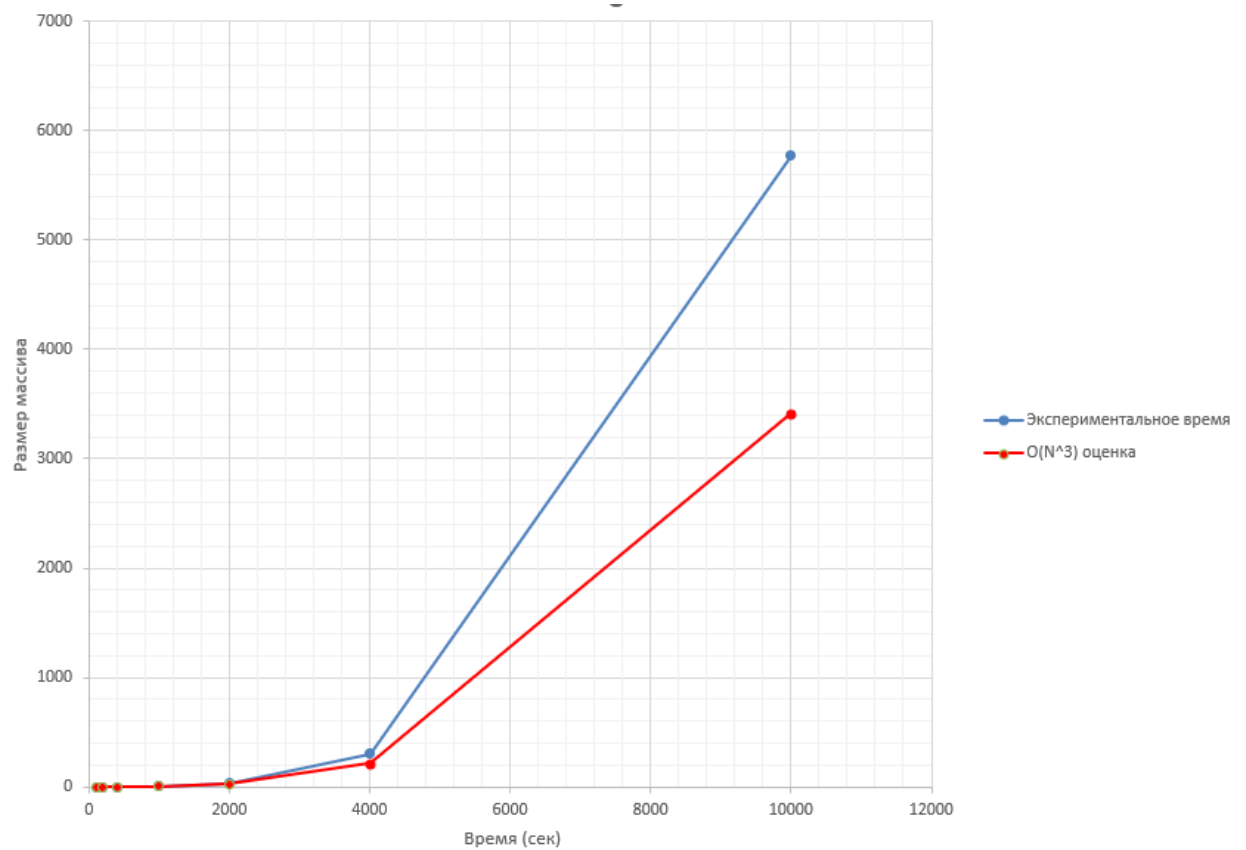
```
Консоль отладки Microsoft Visual Studio
10 el:
81741 82646 83392 79572 82060 82433 81369 81624 82483 82524
82705 83666 84385 80934 82774 84026 81866 82669 82244 82892
82565 82379 85183 80898 82460 83526 81154 82468 82911 83366
80376 80297 81993 79068 81379 81343 80298 80520 80688 81729
80824 81546 81555 79419 80168 80981 79082 81217 80739 80319
80932 80654 81911 80017 82353 81811 81801 81041 81354 82077
81113 80109 80833 78308 80071 81191 80301 79006 80611 80233
80606 80754 81676 79034 80607 80385 78646 80265 80086 80672
81977 81294 83550 79139 81440 80525 81609 80535 80958 80902
80434 80166 80493 77550 79798 80752 80813 78957 80622 80861
time: 317.477000000
C:\Users\amaka\source\repos\ConsoleApplication1\x64\Debug\ConsoleApplication1.exe (процесс 1368) завершил работу с кодом
0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

10000

```
Консоль отладки Microsoft Visual Studio
10 el:
203205 203891 205272 206017 206983 205658 207125 206811 206180 205378
201620 201871 202024 203611 204403 203771 202133 202623 204438 201570
199402 202940 201712 202828 206061 203604 203014 204076 203101 201947
198302 199565 200250 201966 202707 202775 203444 202078 203062 201914
203247 202857 203733 204987 206973 203748 206279 204819 204338 203906
199870 201154 201656 200531 203463 203864 201557 202443 203611 201588
201940 202178 201534 205415 205114 204428 204907 204543 206434 203609
197638 199645 201005 201090 201833 201472 202558 200769 203033 199229
197901 198130 200784 201412 201069 203161 202009 202402 202801 201032
201089 204090 202636 204803 204956 204256 205552 205469 204259 201318
time: 5773.017000000
C:\Users\amaka\source\repos\ConsoleApplication1\x64\Debug\ConsoleApplication1.exe (процесс 2204) завершил работу с кодом
0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Ав
томатически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно: _
```

Размер массива (N)	Время сортировки (сек)
100	0.01
200	0.019
400	0.168
1000	3.416
2000	32.09
4000	300.672
10000	5773.017





## 2-ое задание

```
Консоль отладки Microsoft Visual Studio
Время работы сортировки Шелла на случайном массиве: 0.000006
Время работы быстрой сортировки на случайном массиве: 0.000054
Время работы сортировки Шелла на возрастающем массиве: 0.000002
Время работы быстрой сортировки на возрастающем массиве: 0.000018
Время работы сортировки Шелла на убывающем массиве: 0.000003
Время работы быстрой сортировки на убывающем массиве: 0.000026
Время работы сортировки Шелла на угловатом массиве: 0.000004
Время работы быстрой сортировки на угловатом массиве: 0.000018
```



## **Вывод**

### **1-ое задание**

График показывает, что время сортировки растёт очень быстро с увеличением размера массива, примерно, как кубический рост. Это значит, что алгоритм работает медленно на больших объёмах данных, и время его работы растёт очень сильно. Для маленьких массивов разница между теорией и практикой может быть заметна из-за разных технических деталей, но в целом поведение совпадает с ожиданиями. Такой алгоритм не подходит для работы с очень большими массивами, лучше использовать более быстрые методы.

В сравнении с теоретической оценкой времени по сложности  $O(N^3)$  экспериментальные данные показывают примерно такой же быстрый рост времени с увеличением размера массива, что подтверждает высокую вычислительную сложность алгоритма.

### **2-ое задание**

Разница времени работы сортировки Шелла и быстроты сортировки (quicksort) на разных типах массивов (случайный, возрастающий, убывающий, угловой) очень мала. Это говорит о высокой эффективности обеих сортировок на небольших тестовых массивах: быстрая сортировка и сортировка Шелла выполняются фактически за доли миллисекунды независимо от начального состояния данных.