**Пензенский государственный университет**

**Кафедра вычислительной техники**

**Отчет**

по лабораторной работе № 2

по дисциплине: “Логика и алгоритмизация в инженерных задачах"

Выполнили студенты группы 22ВВП2:

Бормотов А.А.

Кузьмин Д.В.

Кочетков А.М.

Приняли:

Юрова О.В.

Акифьев И.В.

2023

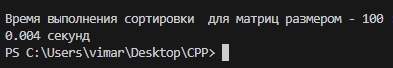
**Цель**:

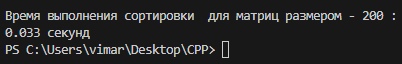
**Задание 1**

1. Определение О-символики программы



1. Время сортировки матриц размерами от 100 и до 2000



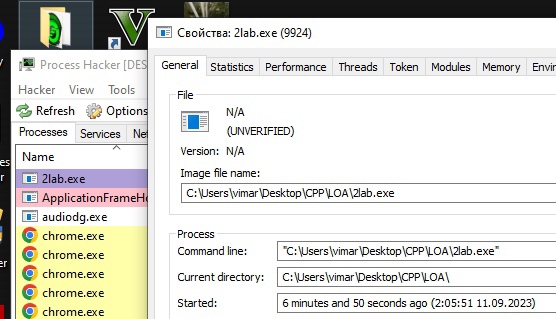


https://sun9-22.userapi.com/impg/lxGRV1Cy9ZtoegUvI_FZE8cMZWeBKHRimTniMw/82HpGmRW6eY.jpg?size=401x63&quality=96&sign=56c249d7c4ecc1399120e30e041a13e7&type=album

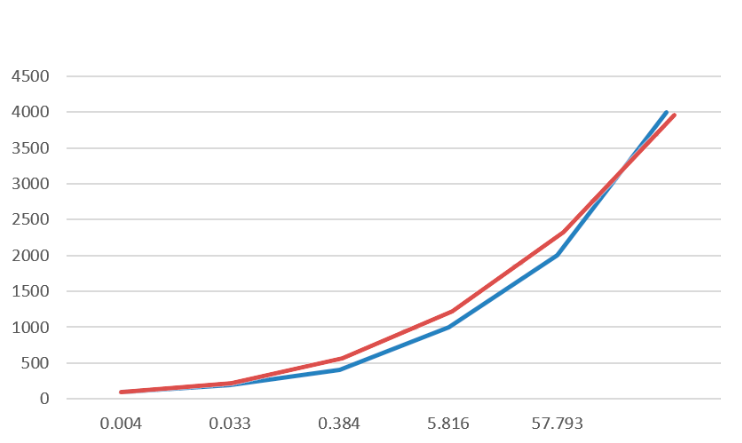
https://sun9-59.userapi.com/impg/SikMMlmv7ej94WBo7EZMWMLWGDUUuahvyE2gbg/CeNN_Xyfkj0.jpg?size=410x58&quality=96&sign=a1827771b83b0fd8065102377246e133&type=album

https://sun9-13.userapi.com/impg/y3qBJV9N04mcjgxB8Cw8L1C-IHh59sn6tbea1A/0YE-Iz62bAc.jpg?size=408x59&quality=96&sign=c32a09ff6e9a32a5962ff6ecbabc79d1&type=album

При дальнейшей оценке времени выполнения программы, при размере матриц больше 2000, время превысило 6 минут:

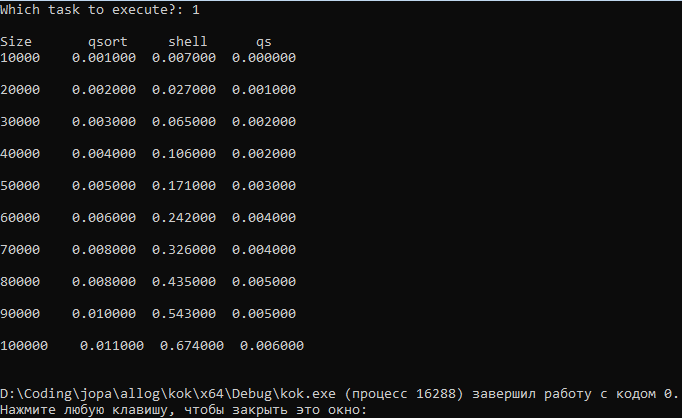


1. Красным отмечено теоретическое время выполнения, синим – действительное

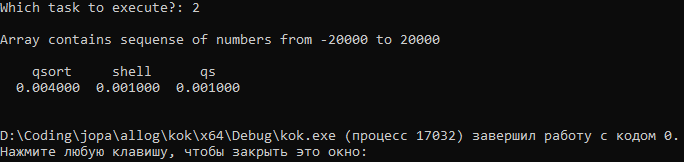


**Задание 2**

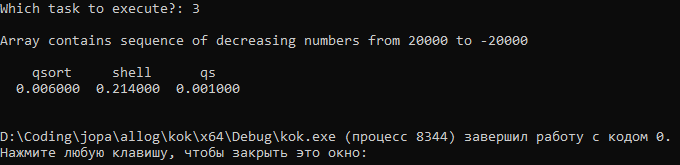
1. Время выполнения функций на случайных наборах данных



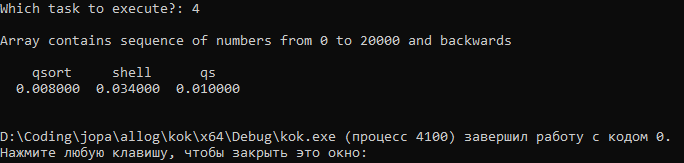
1. Время выполнения функций на возрастающей последовательности чисел



1. Время выполнения функций на убывающей последовательности чисел



1. Время выполнения функций на возрастающей, а потом убывающей последовательности чисел



**Листинг программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

void shell(int\* items, int count)

{

clock\_t start\_s, end\_s;

int i, j, gap, k;

int x, a[5];

a[0] = 9; a[1] = 5; a[2] = 3; a[3] = 2; a[4] = 1;

start\_s = clock(); // Начало отсчета времени работы функции shell

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;

}

}

end\_s = clock() - start\_s; // Конец отсчета времени работы функции shell

printf("%lf", (double)end\_s / CLOCKS\_PER\_SEC); // Вывод времени работы функции shell

}

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);

}

// function for making qsort work

int Comp(const void\* i1, const void\* i2)

{

return (\*(int\*)i1 - \*(int\*)i2);

}

// file processing function

void File\_Process(char file\_name[14])

{

int\* items; // pointer

int count = 0;

clock\_t start\_qs, end\_qs; // qs time

// saving file length

FILE\* curr\_file = fopen(file\_name, "r");

fseek(curr\_file, 0, SEEK\_SET);

while (true)

{

int val;

if (fscanf(curr\_file, "%d", &val) == 1)

{

count++;

}

if (feof(curr\_file))

{

break;

}

}

fclose(curr\_file);

items = (int\*)malloc(count \* sizeof(int)); // memory expansion

// filling array from file

curr\_file = fopen(file\_name, "r");

fseek(curr\_file, 0, SEEK\_SET);

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

{

fscanf(curr\_file, "%d", &(\*(items + i)));

}

fclose(curr\_file);

clock\_t start = clock(), end; // start of qsort func

qsort(items, count, sizeof(int), Comp); // qsort call

end = clock() - start; // end of qsort func

printf(" %lf ", (double)end / CLOCKS\_PER\_SEC); // time consumed by qsort

curr\_file = fopen(file\_name, "r");

fseek(curr\_file, 0, SEEK\_SET);

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

{

fscanf(curr\_file, "%d", &(\*(items + i)));

}

fclose(curr\_file);

shell(items, count); // shell call

curr\_file = fopen(file\_name, "r");

fseek(curr\_file, 0, SEEK\_SET);

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

{

fscanf(curr\_file, "%d", &(\*(items + i)));

}

fclose(curr\_file);

start\_qs = clock(); // start of qs func

qs(items, 1, count - 1); // qs call

end\_qs = clock() - start\_qs; // end of qs func

printf(" %lf\n\n", (double)end\_qs / CLOCKS\_PER\_SEC); // time consumed by qs

free(items); // freeing memory

}

int main(clock\_t end\_s)

{

srand(time(NULL)); // rng

clock\_t start\_qs, end\_qs; // qs time

int TaskType;

printf("Which task to execute?: ");

scanf("%d", &TaskType);

switch (TaskType)

{

case 1:

{

int\* items;

printf("\nSize\t qsort shell qs\n");

for (int count = 10000; count <= 100000; count += 10000) // array size variants from 10000 to 100000 with step of 10000

{

printf("%d ", count);

items = (int\*)malloc(count \* sizeof(int)); // memory expansion

// filling array with rng

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

{

\*(items + i) = rand() % 100;

}

clock\_t start = clock(), end; // start of qsort func

qsort(items, count, sizeof(int), Comp); // qsort call

end = clock() - start; // end of qsort func

printf(" %lf ", (double)end / CLOCKS\_PER\_SEC); // time consumed by qsort

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

{

\*(items + i) = rand() % 100;

}

shell(items, count); // shell call

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

{

\*(items + i) = rand() % 100;

}

start\_qs = clock(); // start of qs func

qs(items, 0, count - 1); // qs call

end\_qs = clock() - start\_qs; // end of qs func

printf(" %lf\n\n", (double)end\_qs / CLOCKS\_PER\_SEC); // time consumed by qs

}

free(items); // freeing memory

break;

}

case 2:

{

printf("\nArray contains sequense of numbers from -20000 to 20000\n\n qsort shell qs\n");

char file\_name[14] = "ArrInc.txt";

File\_Process(file\_name);

break;

}

case 3:

{

printf("\nArray contains sequence of decreasing numbers from 20000 to -20000\n\n qsort shell qs\n");

char file\_name[14] = "ArrDec.txt";

File\_Process(file\_name);

break;

}

case 4:

{

printf("\nArray contains sequence of numbers from 0 to 20000 and backwards\n\n qsort shell qs\n");

char file\_name[14] = "ArrIncDec.txt";

File\_Process(file\_name);

break;

}

default:

{

return 0;

}

}

}

**Вывод:**