МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнил:

Студент группы 23ВВВ2

Монин Иван

Приняли:

доцент Юрова О.В.

доцент Митрохин М.А.

Пенза 2024

**Цель работы** – научиться оценивать время выполнения программ.

**Лабораторное задание:**

**Задание 1**

- Вычислить порядок сложности программы (О-символику).

- 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.

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

**Задание 2**

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

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

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

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

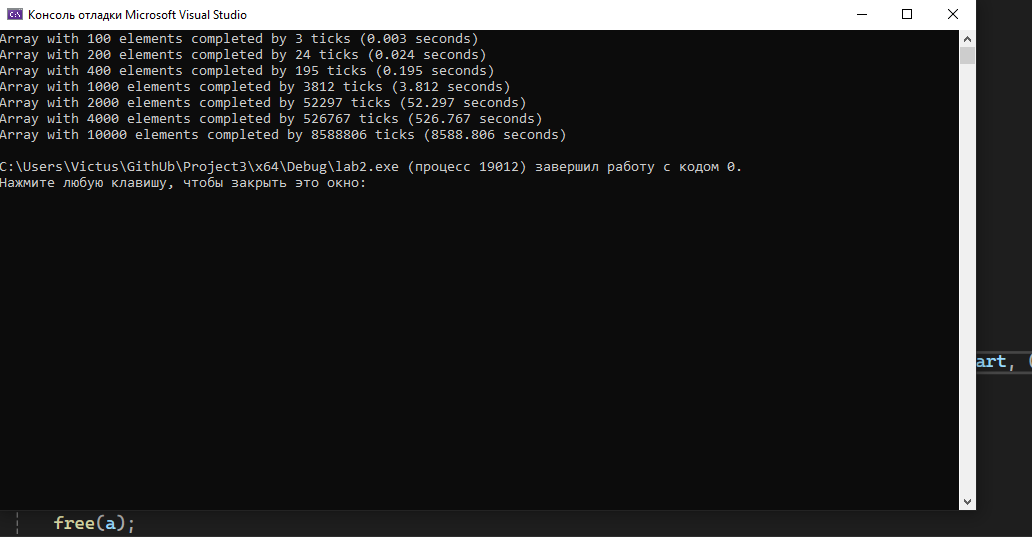
- Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

\

**Задание 1:**

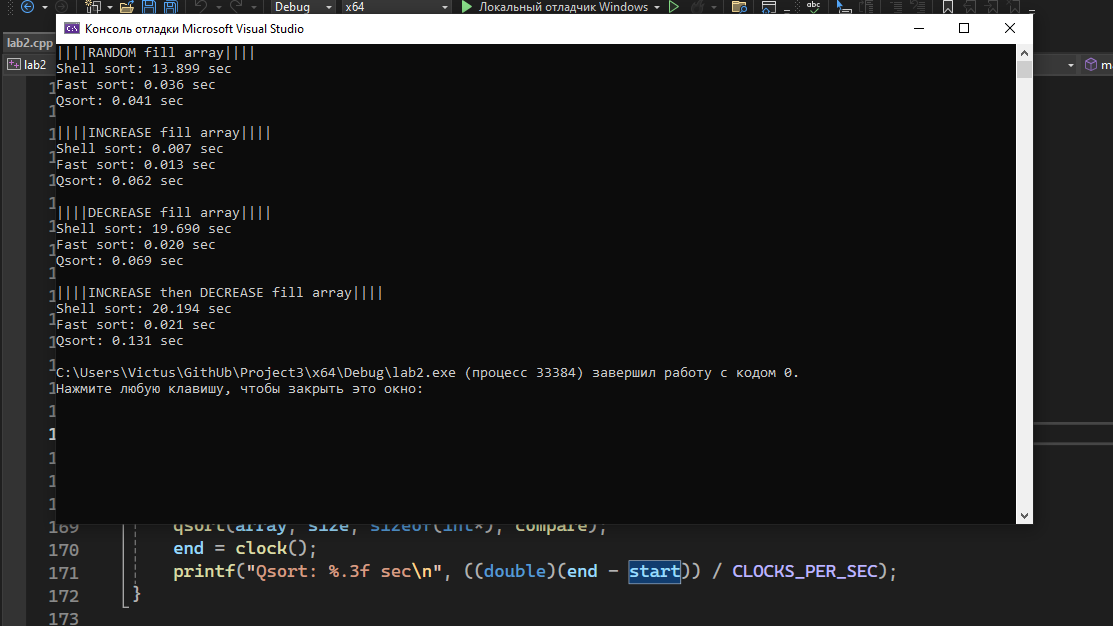
****

|  |  |  |
| --- | --- | --- |
| **Размер массива** | **Кол-во тиков** | **Время работы (сек)** |
| 100 | 3 | 0.003 |
| 200 | 24 | 0.024 |
| 400 | 195 | 0.195 |
| 1000 | 3812 | 3.812 |
| 2000 | 52297 | 52.297 |
| 4000 | 526767 | 526.767 |
| 10000 | 8588806 | 8588.806 |

****

**2 Задание**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Сортировка** | **Последовательности** | | | |
| **Случайная** | **Возрастающая** | **Убывающая** | **Возр. – убыв.** |
| Шелла | 13.899 сек | 0.007 сек | 19.690 сек | 20.194 сек |
| Быстрая | 0.036 сек | 0.013 сек | 0.020 сек | 0.021 сек |
| qsort() | 0.041 сек | 0.062 сек | 0.069 сек | 0.131 сек |

****

**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; // объявляем переменные для определения времени выполнения

int i = 0, j = 0, r, k = 0, size[7] = { 100,200,400,1000,2000,4000,10000 };

while (k != 7) {

// Динамическое выделение памяти для массивов

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

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

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

for (i = 0; i < size[k]; i++) {

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

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

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

}

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

while (i < size[k])

{

while (j < size[k])

{

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

j++;

}

i++;

}

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

while (i < size[k])

{

while (j < size[k])

{

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

j++;

}

i++;

}

start = clock();

for (i = 0; i < size[k]; i++) {

for (j = 0; j < size[k]; j++) {

int elem\_c = 0;

for (r = 0; r < size[k]; r++) {

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

}

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

}

}

end = clock();

printf("Array with %d elements completed by %d ticks (%.3f seconds)\n", size[k], end - start, (double)(end - start) / CLOCKS\_PER\_SEC);

for (i = 0; i < size[k]; i++) {

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

k++;

}

return 0;

}

Задание 2

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int\* createArr(int\* arr, int len) {

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

return arr;

}

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

{

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

}

int\* randFill(int\* arr, int len) {

for (int o = 0; o < len; o++) {

arr[o] = rand() % 100 - 50;

}

return arr;

}

int\* incFill(int\* arr, int len) {

for (int o = 0; o < len; o++) {

arr[o] = o;

}

return arr;

}

int\* decFill(int\* arr, int len) {

for (int o = 0; o < len; o++) {

arr[o] = len/(o+1);

}

return arr;

}

int\* incAndDecFill(int\* arr, int len) {

for (int o = 0; o < len/2; o++) {

arr[o] = o;

arr[o+len/2] = len / (o + 1);

}

return arr;

}

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);

}

void main() {

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

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

clock\_t start, end;

srand(time(NULL));

int size = 400000;

int\* array = { 0 };

printf("||||RANDOM fill array||||\n");

array = createArr(array, size);

array = randFill(array, size);

start = clock();

shell(array, size);

end = clock();

printf("Shell sort: %.3f sec\n", ((double)(end - start))/CLOCKS\_PER\_SEC);

array = randFill(array, size);

start = clock();

qs(array, 0,size-1);

end = clock();

printf("Fast sort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

array = randFill(array, size);

start = clock();

qsort(array,size,sizeof(int \*),compare);

end = clock();

printf("Qsort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

printf("\n||||INCREASE fill array||||\n");

array = incFill(array, size);

start = clock();

shell(array, size);

end = clock();

printf("Shell sort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

array = incFill(array, size);

start = clock();

qs(array, 0, size - 1);

end = clock();

printf("Fast sort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

array = incFill(array, size);

start = clock();

qsort(array, size, sizeof(int\*), compare);

end = clock();

printf("Qsort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

printf("\n||||DECREASE fill array||||\n");

array = decFill(array, size);

start = clock();

shell(array, size);

end = clock();

printf("Shell sort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

array = decFill(array, size);

start = clock();

qs(array, 0, size - 1);

end = clock();

printf("Fast sort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

array = decFill(array, size);

start = clock();

qsort(array, size, sizeof(int\*), compare);

end = clock();

printf("Qsort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

printf("\n||||INCREASE then DECREASE fill array||||\n");

array = incAndDecFill(array, size);

start = clock();

shell(array, size);

end = clock();

printf("Shell sort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

array = incAndDecFill(array, size);

start = clock();

qs(array, 0, size - 1);

end = clock();

printf("Fast sort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

array = incAndDecFill(array, size);

start = clock();

qsort(array, size, sizeof(int\*), compare);

end = clock();

printf("Qsort: %.3f sec\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

}

**Вывод**

В ходе работы выполнения лабораторной работы была рассчитана сложность программы (О-символикой). Так же было выполнено измерение времени выполнения перемножения массивов и разных видов сортировки.