Министерство образования и науки Российской Федерации

Пензенский государственный университет

Факультет вычислительной техники

**Отчёт**

По лабораторной работе №2

По курсу "ЛиОАвИЗ"

Выполнили:

Студентка группы 20ВВ1

Зацепилина Е.С.

Приняла:

Юрова О.В.

Пенза 2021

**Название:** Оценка времени выполнения программ

**Цель работы:** выполнить ряд заданий

**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.

1. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.
2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.
3. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.
4. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

**Описание метода решения задачи:**

Задание 1.

1. Оценивая порядок сложности алгоритма, необходимо использовать только ту часть, которая возрастает быстрее всего. В нашем случае рабочий цикл описывается выражением N^2+N^2+N^3, таким образом порядок сложности O(N^3)
2. В начале\конце работы программы переменной start\end присваивается значение clock(), которая означает время начала\конца работы алгоритма. В конце работы выводится время, за которое алгоритм завершает свою работу. Текст программы представлен в пункте «Листинг». Результаты работы программы приведены в таблице 1 и пункте «Результаты работы программы» (рисунок 2-9). График зависимости времени работы от объема данных изображён на рисунке 1.

Таблица 1 – результаты работы программы.

|  |  |
| --- | --- |
| **Размер матрицы (n)** | **Время обработки (секунды)** |
| 100 | 0.004 |
| 200 | 0.023 |
| 400 | 0.169 |
| 800 | 1.795 |
| 1000 | 3.679 |
| 1500 | 13.232 |
| 2000 | 51.323 |
| 2500 | 83.524 |

По результатам работы программы можно сделать вывод, что сложность алгоритма соответствует времени выполнения алгоритма.

1. График зависимости времени работы от объема данных изображён на рисунке 1.

Рисунок 1 - График зависимости времени работы от объема данных.

Задание 2.

Для решения пунктов 1-5 задания 2 написана программа, текст которой представлен в пункте «Листинг». В программе разными способами сортируются разные массивы с общей размерностью n = 100000. Результаты работы программы представлены в таблице 2 и пункте «Результаты работы программы» (рисунок 10).

Таблица 2 – результаты работы программы.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Случайные  числа | Возрастающие  Числа | Убывающие  числа | Возрастающие\  Убывающие  числа |
| shell() | 0,484 | 0,001 | 0,955 | 0,717 |
| qs() | 0,007 | 0,003 | 0,002 | 0,004 |
| qsort() | 20,0 | 0,019 | 0,019 | 0,02 |

Жёлтым в таблице отмечены минимальные результаты среди всех алгоритмов сортировки всех видов массивов. Наиболее оптимальным для случаев с массивом случайных чисел, убывающих и возрастающих\убывающих чисел стал алгоритм сортировки qs(). Для массива возрастающих чисел – shell(). Отсюда можно сделать вывод, что наиболее оптимальным алгоритмом сортировки является qs(). Так же стоит заметить, что в тех случаях, когда массив уже отсортирован(возрастающие числа), более оптимальный результат показывает алгоритм shell(), так как проходит этот массив быстрее остальных алгоритмов.

**Вывод:** была изучена библиотека time.h, использующаяся для оценки времени работы алгоритмов. Проведено определение порядка сложности алгоритма умножения матриц. Оценены разные алгоритмы сортировки массивов.

**Листинг:**

1.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <locale.h>

#include <time.h>

#include <windows.h>

int main(void)

{

char\* locale = setlocale(LC\_ALL, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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

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

clock\_t start, end; // объявляем переменные для определения времени выполнения

int elem\_c, r;

const int n = 2500;

start = clock();//Начало работы алгоритма

int\*\* a = new int\* [n];

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

a[i] = new int[n];

}

int\*\* b = new int\* [n];

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

b[i] = new int[n];

}

int\*\* c = new int\* [n];

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

c[i] = new int[n];

}

srand(time(NULL));

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

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

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

}

}

srand(time(NULL));

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

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

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

}

}

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

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

elem\_c = 0;

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

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

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

}

}

}

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

delete[] a[i];

delete[] b[i];

delete[] c[i];

}

delete[] a;

delete[] b;

delete[] c;

end = clock();//Конец работы алгоритма

double time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;//Время работы алгоритма

printf("Цикл умножения прошёл за %f секунд.\n", time\_spent);

return(0);

}

2.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <locale.h>

#include <time.h>

#include <windows.h>

#include <cmath>

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 compare(const void\* x1, const void\* x2) // функция сравнения элементов массива

{

return (\*(int\*)x1 - \*(int\*)x2); // если результат вычитания равен 0, то числа равны, < 0: x1 < x2; > 0: x1 > x2

}

int main(void)

{

char\* locale = setlocale(LC\_ALL, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

const int n = 100000; //размерность массивов

int i;

double time\_spent;

int\* arr1 = new int[n]; //случайный массив

int\* arr2 = new int[n];//возрастающий массив

int\* arr3 = new int[n];//убывающий массив

int\* arr4 = new int[n];//возрастающий\убывающий массив

int\* arr = new int[n];//промежуточный массив

clock\_t start, end;//начало\конец работы

//Генерируем массивы

srand(time(NULL));

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

{arr1[i] = rand() % 1000 + 1;}

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

{arr2[i] = i + 1;}

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

{arr3[i] = -i;}

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

{

if (i <= n / 2) { arr4[i] = i; }

else { arr4[i] = -i; }

}

//Сортировка массива случайных чисел

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

{

arr[i] = arr1[i];

}

start = clock();

shell(arr, n);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив случайных чисел отсортирован с помощью shell за %f секунд.\n", time\_spent);

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

{

arr[i] = arr1[i];

}

start = clock();

qs(arr, 0, n-1);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив случайных чисел отсортирован с помощью qs за %f секунд.\n", time\_spent);

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

{

arr[i] = arr1[i];

}

start = clock();

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

end = clock();

time\_spent = end - start;

printf("Массив случайных чисел отсортирован с помощью qsort за %f секунд.\n", time\_spent);

printf("\n");

//Сортировка массива возрастающих чисел

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

{

arr[i] = arr2[i];

}

start = clock();

shell(arr, n);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив возрастающих чисел отсортирован с помощью shell за %f секунд.\n", time\_spent);

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

{

arr[i] = arr2[i];

}

start = clock();

qs(arr, 0, n - 1);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив возрастающих чисел отсортирован с помощью qs за %f секунд.\n", time\_spent);

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

{

arr[i] = arr2[i];

}

start = clock();

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

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив возрастающих чисел отсортирован с помощью qsort за %f секунд.\n", time\_spent);

printf("\n");

//Сортировка массива убывающих чисел

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

{

arr[i] = arr3[i];

}

start = clock();

shell(arr, n);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив убывающих чисел отсортирован с помощью shell за %f секунд.\n", time\_spent);

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

{

arr[i] = arr3[i];

}

start = clock();

qs(arr, 0, n - 1);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив убывающих чисел отсортирован с помощью qs за %f секунд.\n", time\_spent);

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

{

arr[i] = arr3[i];

}

start = clock();

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

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив убывающих чисел отсортирован с помощью qsort за %f секунд.\n", time\_spent);

printf("\n");

//Сортировка массива возрастающих/убывающих чисел

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

{

arr[i] = arr4[i];

}

start = clock();

shell(arr, n);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив возрастающих/убывающих чисел отсортирован с помощью shell за %f секунд.\n", time\_spent);

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

{

arr[i] = arr4[i];

}

start = clock();

qs(arr, 0, n - 1);

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив возрастающих/убывающих чисел отсортирован с помощью qs за %f секунд.\n", time\_spent);

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

{

arr[i] = arr4[i];

}

start = clock();

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

end = clock();

time\_spent = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Массив возрастающих/убывающих чисел отсортирован с помощью qsort за %f секунд.\n", time\_spent);

delete[] arr;

delete[] arr1;

delete[] arr2;

delete[] arr3;

delete[] arr4;

return(0);

}

**Результаты работы программы:**

1.



Рисунок 2 – результаты работы программы.



Рисунок 3 – результаты работы программы.



Рисунок 4 – результаты работы программы.



Рисунок 5 – результаты работы программы.



Рисунок 6 – результаты работы программы.



Рисунок 7 – результаты работы программы.



Рисунок 8 – результаты работы программы.



Рисунок 9 – результаты работы программы.

2.

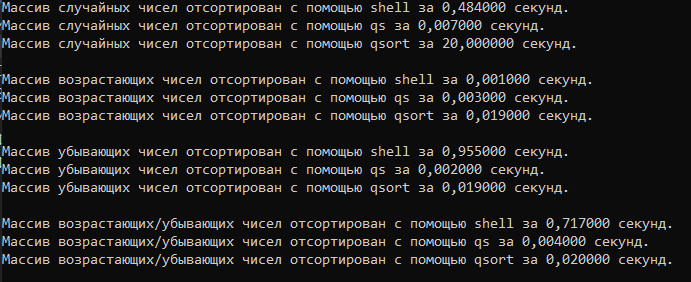


Рисунок 10 – результаты работы программы.