Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»  
  
  
  
  
  
  
ОТЧЁТ

по лабораторной работе № 2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнил:

Димитренко Н.С.

Приняли:

к.т.н., доцент Юрова О.А.

д.т.н., профессор, зав. каф. ВТ Митрохин М. А.

Пенза 2022

**Название:** Оценка времени выполнения программ

**Цель работы:** выполнить лабораторные указания 1-2 используя оценку времени выполнения программ.

**Лабораторные задания:**

**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, 500, 750, 1000, 1500, 2000.
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.

**Задание 1:**

**Листинг:**

#include <Windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

const int N = 2000;

int a[N][N], b[N][N], c[N][N], elem\_c, p = N;

int main(void)

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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

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

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

int i = 0, j = 0, r;

double result;

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

while (i < p)

{

while (j < p)

{

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

j++;

}

i++;

}

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

i = 0; j = 0;

while (i < p)

{

while (j < p)

{

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

j++;

}

i++;

}

start = clock();

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

{

for (j = 0;j < p;j++)

{

elem\_c = 0;

for (r = 0;r < p;r++)

{

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

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

}

}

}

stop = clock();

result = (double)(stop - start) / 1000;

printf("Затраченное время: %.3f секунд\n ", result);

return(0);

}

1. Вычислить порядок сложности программы (*О*-символику).

Чтобы вычислить порядок сложности программы, нужно найти наибольшее количество циклов (for и while) которые вложены в другие циклы и т. д. В данном случае наибольшее количество вложенных циклов является эти строчки кода:

for (i = 0;i < 200;i++)

{

for (j = 0;j < 200;j++)

{

elem\_c = 0;

for (r = 0;r < 200;r++)

{

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

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

}

}

}

Сложность программы = O(n3).

1. Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 500, 750, 1000, 1500, 2000.

При 100:



При 200:



При 500:



При 750:



При 1000:



При 1500:

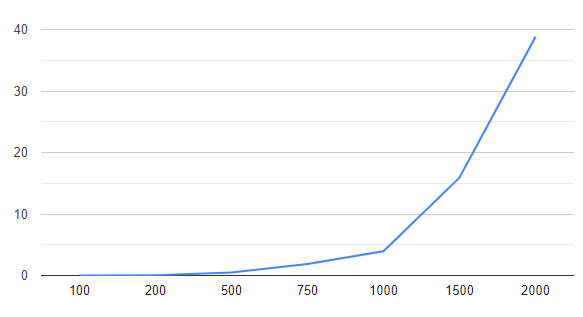


При 2000:



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

Так как сложность программы = O(n3), значит сложность должна похожа на график y=x3. Действительно, график похож на данную функцию.



**Задание 2:**

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <Windows.h>

#include <stdlib.h>

#include <time.h>

#include <stdio.h>

#include <malloc.h>

#include <conio.h>

//сортировка методом Шелла

void ShellSort(int n, int mass[])

{

int i, j, step;

int tmp;

for (step = n / 2; step > 0; step /= 2)

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

{

tmp = mass[i];

for (j = i; j >= step; j -= step)

{

if (tmp < mass[j - step])

mass[j] = mass[j - step];

else

break;

}

mass[j] = tmp;

}

}

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 Save(FILE\* F, double result\_sh, double result\_qs, double result\_qsort)

{

fprintf(F, "%5.3f|%5.3f|%5.3f|", result\_sh, result\_qs, result\_qsort);

return;

}

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

{

int data\_1 = \*(int\*)a;

int data\_2 = \*(int\*)b;

if (data\_1 < data\_2) return -1;

else if (data\_1 == data\_2) return 0;

else return 1;

}

int main()

{

SetConsoleCP(1251); //русификация

SetConsoleOutputCP(1251);

int N\_r[3] = { 100000, 500000, 1000000 };

char n\_file[50];

FILE\* F;

printf("Введите имя файла, в который будут записывать данные: ");

scanf("%s", n\_file);

F = fopen(n\_file, "w");

if (n\_file == NULL)

{

printf("Ошибка открытия файла");

\_getch();

return 0;

}

fprintf(F, "%3s|%17d|%17d|%17d|\n"," ", N\_r[0], N\_r[1], N\_r[2]);

fprintf(F, "%3s|%5s|%5s|%5s|%5s|%5s|%5s|%5s|%5s|%5s|\n"," ", "sh", "qs", "qsort", "sh", "qs", "qsort", "sh", "qs", "qsort");

//выделение памяти под массив

int\* Sh;

int\* Qs;

int\* Qsort;

fprintf(F, "%3s|", "r");

for (int j = 0; j < 3; j++)

{

Sh = (int\*)malloc(N\_r[j] \* sizeof(int));

Qs = (int\*)malloc(N\_r[j] \* sizeof(int));

Qsort = (int\*)malloc(N\_r[j] \* sizeof(int));

for (int i = 0; i < N\_r[j]; i++)

{

Sh[i] = rand() % 1001 - 1000;

Qs[i] = rand() % 1001 - 1000;

Qsort[i] = rand() % 1001 - 1000;

}

clock\_t start\_sh, stop\_sh, start\_qs, stop\_qs, start\_qsort, stop\_qsort;

start\_sh = clock();

ShellSort(N\_r[j], Sh);

stop\_sh = clock();

start\_qs = clock();

qs(Qs, 0, N\_r[j] - 1);

stop\_qs = clock();

start\_qsort = clock();

qsort(Qsort, N\_r[j], sizeof(int), comp);

stop\_qsort = clock();

double result\_sh = (double)(stop\_sh - start\_sh) / 1000;

double result\_qs = (double)(stop\_qs - start\_qs) / 1000;

double result\_qsort = (double)(stop\_qsort - start\_qsort) / 1000;

Save(F, result\_sh, result\_qs, result\_qsort);

free(Sh);

free(Qs);

free(Qsort);

}

fprintf(F, "\n%3s|", "d");

for (int j = 0; j < 3; j++)

{

int Max = 1000000, razn = 1;

Sh = (int\*)malloc(N\_r[j] \* sizeof(int));

Qs = (int\*)malloc(N\_r[j] \* sizeof(int));

Qsort = (int\*)malloc(N\_r[j] \* sizeof(int));

for (int i = 0; i < N\_r[j]; i++)

{

Sh[i] = Max - razn;

Qs[i] = Max - razn;

Qsort[i] = Max - razn;

Max -= razn;

}

clock\_t start\_sh, stop\_sh, start\_qs, stop\_qs, start\_qsort, stop\_qsort;

start\_sh = clock();

ShellSort(N\_r[j], Sh);

stop\_sh = clock();

start\_qs = clock();

qs(Qs, 0, N\_r[j] - 1);

stop\_qs = clock();

start\_qsort = clock();

qsort(Qsort, N\_r[j], sizeof(int), comp);

stop\_qsort = clock();

double result\_sh = (double)(stop\_sh - start\_sh) / 1000;

double result\_qs = (double)(stop\_qs - start\_qs) / 1000;

double result\_qsort = (double)(stop\_qsort - start\_qsort) / 1000;

Save(F, result\_sh, result\_qs, result\_qsort);

free(Sh);

free(Qs);

free(Qsort);

}

fprintf(F, "\n%3s|", "u");

for (int j = 0; j < 3; j++)

{

Sh = (int\*)malloc(N\_r[j] \* sizeof(int));

Qs = (int\*)malloc(N\_r[j] \* sizeof(int));

Qsort = (int\*)malloc(N\_r[j] \* sizeof(int));

int razn = 1, Min = 0;

for (int i = 0; i < N\_r[j]; i++)

{

Sh[i] = Min + razn;

Qs[i] = Min + razn;

Qsort[i] = Min + razn;

Min += razn;

}

clock\_t start\_sh, stop\_sh, start\_qs, stop\_qs, start\_qsort, stop\_qsort;

start\_sh = clock();

ShellSort(N\_r[j], Sh);

stop\_sh = clock();

start\_qs = clock();

qs(Qs, 0, N\_r[j] - 1);

stop\_qs = clock();

start\_qsort = clock();

qsort(Qsort, N\_r[j], sizeof(int), comp);

stop\_qsort = clock();

double result\_sh = (double)(stop\_sh - start\_sh) / 1000;

double result\_qs = (double)(stop\_qs - start\_qs) / 1000;

double result\_qsort = (double)(stop\_qsort - start\_qsort) / 1000;

Save(F, result\_sh, result\_qs, result\_qsort);

free(Sh);

free(Qs);

free(Qsort);

}

fprintf(F, "\n%3s|", "ud");

for (int j = 0; j < 3; j++)

{

Sh = (int\*)malloc(N\_r[j] \* sizeof(int));

Qs = (int\*)malloc(N\_r[j] \* sizeof(int));

Qsort = (int\*)malloc(N\_r[j] \* sizeof(int));

int Max = 1000000, razn = 1, Min = 0;

for (int i = 0; i < N\_r[j]; i++)

{

if (i < N\_r[j] / 2)

{

Sh[i] = Min + razn;

Qs[i] = Min + razn;

Qsort[i] = Min + razn;

Min += razn;

}

else

{

Sh[i] = Max - razn;

Qs[i] = Max - razn;

Qsort[i] = Max - razn;

Max -= razn;

}

}

clock\_t start\_sh, stop\_sh, start\_qs, stop\_qs, start\_qsort, stop\_qsort;

start\_sh = clock();

ShellSort(N\_r[j], Sh);

stop\_sh = clock();

start\_qs = clock();

qs(Qs, 0, N\_r[j] - 1);

stop\_qs = clock();

start\_qsort = clock();

qsort(Qsort, N\_r[j], sizeof(int), comp);

stop\_qsort = clock();

double result\_sh = (double)(stop\_sh - start\_sh) / 1000;

double result\_qs = (double)(stop\_qs - start\_qs) / 1000;

double result\_qsort = (double)(stop\_qsort - start\_qsort) / 1000;

Save(F, result\_sh, result\_qs, result\_qsort);

free(Sh);

free(Qs);

free(Qsort);

}

printf("Данные сохранены в %s", n\_file);

fclose(F);

\_getch();

return 0;

}

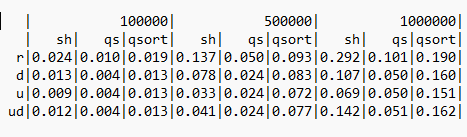
Для выполнения следующих заданий нам понадобится крупные наборы данных, так как данные сортировки очень быстрые. Возьмём значения от 100000 до 1000000: 100000, 500000 и 1000000.

На данном рисунке показывает результаты сортировки при выше указанных значений.

Все измерения времени – в секундах.

sh – сортировка Шелла ; qs – быстрая сортировка (self-written) ; qsort – встроенная функция быстрой сортировки.

r – случайные значения ; d – заранее отсортировано по убыванию ; u - заранее отсортировано по возрастанию; ud – заранее отсортировано: первая половина – по возрастанию, вторая – по убыванию.



**Вывод:** при всех результатах самой быстрой сортировкой является Qs (self-written). Рекомендуется использовать именно её как при маленьком, так и при большом количестве значений в массиве. У сортировки Шелла время работы примерно такое же, в лучшем случае, как и у быстрой сортировки. Встроенная qsort работатет хуже self-written qs.