Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Отчёт**

по дисциплине «Логика и основы алгоритмизации в инженерных задачах»

по лабораторной работе №2

«Оценка времени выполнения программ»

Выполнили:

студенты группы 22ВВС1

Эрмедов А.Э.

Казаров И.И.

Приняли:

к.т.н, доцент Юрова О.В.

к.э.н, доцент Акифьев И.В.

Пенза 2023

**Название**

Оценка времени выполнения программ

**Практическая часть**

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.

**Листинг LBM2.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main(void)

{

int sizes[] = { 100, 200, 400, 1000, 2000, 4000, 10000 };

int num\_sizes = sizeof(sizes) / sizeof(sizes[0]);

for (int k = 0; k < num\_sizes; k++)

{

int size = sizes[k];

int\*\* a = (int\*\*)malloc(size \* sizeof(int\*));

int\*\* b = (int\*\*)malloc(size \* sizeof(int\*));

int\*\* c = (int\*\*)malloc(size \* sizeof(int\*));

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

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

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

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

}

clock\_t start, end;

// Заполнение матриц случайными числами

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

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

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

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

}

}

start = clock();

// Выполнение умножения матриц

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

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

int elem\_c = 0;

for (int r = 0; r < size; r++) {

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

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

}

}

}

end = clock();

double cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Matrix size: %dx%d, Time: %f seconds\n", size, size, cpu\_time\_used);

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

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

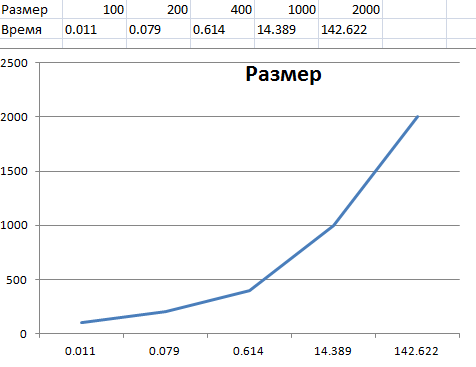
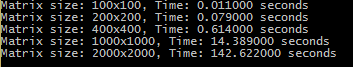
free(c);

}

return 0;

}

**Результат**



**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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Возрастающая | Убывающая | Убывающе-возрастающая | Случайная |
| Шелла | 0,001 | 0,001 | 0,001 | 0,003 |
| qs | 1,097 | 0,002 | 0,004 | 0,002 |
| qsort | 0,004 | 0,003 | 0,004 | 0,004 |

**Листинг LBM22.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

// Функция сортировки Шелла

void shell(int arr[], int n) {

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;

}

}

}

// Рекурсивная функция сортировки QuickSort

void quicksort(int arr[], int low, int high) {

if (low < high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

int pi = i + 1;

quicksort(arr, low, pi - 1);

quicksort(arr, pi + 1, high);

}

}

// Функция сравнения для qsort

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

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

}

int main(void) {

setlocale(LC\_ALL, "Russian");

int count = 10000; // Количество элементов в массиве

int\* items\_shell = (int\*)malloc(count \* sizeof(int));

int\* items\_qs = (int\*)malloc(count \* sizeof(int));

int\* items\_qsort = (int\*)malloc(count \* sizeof(int));

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

int value = rand() % 1000;

items\_shell[i] = value;

items\_qs[i] = value;

items\_qsort[i] = value;

}

clock\_t start, end;

double cpu\_time\_used;

// Оценка времени работы алгоритма сортировки Шелла

start = clock();

shell(items\_shell, count);

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы алгоритма сортировки Шелла на случайном наборе значений: %f секунд\n", cpu\_time\_used);

// Оценка времени работы алгоритма QuickSort (qs)

start = clock();

quicksort(items\_qs, 0, count - 1);

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы алгоритма QuickSort (qs) на случайном наборе значений: %f секунд\n", cpu\_time\_used);

// Оценка времени работы функции qsort

start = clock();

qsort(items\_qsort, count, sizeof(int), compare);

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы функции qsort на случайном наборе значений: %f секунд\n", cpu\_time\_used);

free(items\_shell);

free(items\_qs);

free(items\_qsort);

return 0;

}

**Листинг LBM222.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

// Функция сортировки Шелла

void shell(int arr[], int n) {

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;

}

}

}

// Функция для обмена элементов

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Итеративная функция сортировки QuickSort

void quicksort(int arr[], int low, int high) {

int\* stack = (int\*)malloc(sizeof(int) \* (high - low + 1));

int top = -1;

stack[++top] = low;

stack[++top] = high;

while (top >= 0) {

high = stack[top--];

low = stack[top--];

int pivot = arr[high];

int i = low - 1;

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

int pi = i + 1;

if (pi + 1 < high) {

stack[++top] = pi + 1;

stack[++top] = high;

}

if (low < pi - 1) {

stack[++top] = low;

stack[++top] = pi - 1;

}

}

free(stack);

}

// Функция сравнения для qsort

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

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

}

int main(void) {

setlocale(LC\_ALL, "Russian");

int count = 10000; // Количество элементов в массиве

int\* items\_shell = (int\*)malloc(count \* sizeof(int));

int\* items\_qs = (int\*)malloc(count \* sizeof(int));

int\* items\_qsort = (int\*)malloc(count \* sizeof(int));

if (items\_shell == NULL || items\_qs == NULL || items\_qsort == NULL) {

printf("Ошибка выделения памяти.\n");

return 1;

}

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

items\_shell[i] = i;

items\_qs[i] = i;

items\_qsort[i] = i;

}

clock\_t start, end;

double cpu\_time\_used;

// Оценка времени работы алгоритма сортировки Шелла

start = clock();

shell(items\_shell, count);

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы алгоритма сортировки Шелла на массиве возрастающей последовательности: %f секунд\n", cpu\_time\_used);

// Оценка времени работы итеративного алгоритма QuickSort (qs)

start = clock();

quicksort(items\_qs, 0, count - 1);

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы итеративного алгоритма QuickSort (qs) на массиве возрастающей последовательности: %f секунд\n", cpu\_time\_used);

// Оценка времени работы функции qsort

start = clock();

qsort(items\_qsort, count, sizeof(int), compare);

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы функции qsort на массиве возрастающей последовательности: %f секунд\n", cpu\_time\_used);

free(items\_shell);

free(items\_qs);

free(items\_qsort);

return 0;

}

**Листинг LBM223.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

void shell(int arr[], int n) {

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;

}

}

}

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

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

}

int main(void) {

setlocale(LC\_ALL, "Russian");

int count = 10000; // Количество элементов в массиве

int\* items\_shell = (int\*)malloc(count \* sizeof(int));

int\* items\_qs = (int\*)malloc(count \* sizeof(int));

int\* items\_qsort = (int\*)malloc(count \* sizeof(int));

// Генерация убывающего набора значений массива

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

items\_shell[i] = count - i;

items\_qs[i] = count - i;

items\_qsort[i] = count - i;

}

clock\_t start, end;

// Сортировка Шелла

start = clock();

shell(items\_shell, count);

end = clock();

double cpu\_time\_used\_shell = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы алгоритма сортировки Шелла на убывающем наборе значений: %f секунд\n", cpu\_time\_used\_shell);

// QuickSort (qs)

start = clock();

qsort(items\_qs, count, sizeof(int), compare);

end = clock();

double cpu\_time\_used\_qs = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы алгоритма QuickSort на убывающем наборе значений: %f секунд\n", cpu\_time\_used\_qs);

// qsort

start = clock();

qsort(items\_qsort, count, sizeof(int), compare);

end = clock();

double cpu\_time\_used\_qsort = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы функции qsort на убывающем наборе значений: %f секунд\n", cpu\_time\_used\_qsort);

free(items\_shell);

free(items\_qs);

free(items\_qsort);

return 0;

}

**Листинг LBM224.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

void shell(int arr[], int n) {

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) { // Fixed the comparison condition

arr[j] = arr[j - gap];

}

arr[j] = temp;

}

}

}

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

return \*(int\*)a - \*(int\*)b; // Fixed the comparison logic

}

int main(void) {

setlocale(LC\_ALL, "Russian");

int count = 10000; // Количество элементов в массиве

int\* items\_shell = (int\*)malloc(count \* sizeof(int));

int\* items\_qs = (int\*)malloc(count \* sizeof(int));

int\* items\_qsort = (int\*)malloc(count \* sizeof(int));

// Генерация убывающе-возрастающего набора значений массива

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

items\_shell[i] = (i < count / 2) ? count - i : i - count / 2;

items\_qs[i] = items\_shell[i];

items\_qsort[i] = items\_shell[i];

}

clock\_t start, end;

// Сортировка Шелла

start = clock();

shell(items\_shell, count);

end = clock();

double cpu\_time\_used\_shell = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы алгоритма сортировки Шелла на убывающе-возрастающем наборе значений: %f секунд\n", cpu\_time\_used\_shell);

// QuickSort (qs)

start = clock();

qsort(items\_qs, count, sizeof(int), compare);

end = clock();

double cpu\_time\_used\_qs = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы алгоритма QuickSort на убывающе-возрастающем наборе значений: %f секунд\n", cpu\_time\_used\_qs);

// qsort

start = clock();

qsort(items\_qsort, count, sizeof(int), compare);

end = clock();

double cpu\_time\_used\_qsort = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы функции qsort на убывающе-возрастающем наборе значений: %f секунд\n", cpu\_time\_used\_qsort);

free(items\_shell);

free(items\_qs);

free(items\_qsort);

return 0;

}

**Листинг LBM225.cpp**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

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

{

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

}

int main(void)

{

setlocale(LC\_ALL, "russian");

int count = 10000; // Количество элементов в массиве

int\* items = (int\*)malloc(count \* sizeof(int));

srand(time(NULL));

// Заполнение массива случайными значениями

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

items[i] = rand();

}

clock\_t start, end;

start = clock();

qsort(items, count, sizeof(int), compare); // Вызов стандартной функции qsort

end = clock();

double cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время работы стандартной функции qsort: %f секунд\n", cpu\_time\_used);

free(items);

return 0;

}

**Вывод**

Задание 1.

Теоретически нашли сложность О(n^3). Практически подтвердили это с помощью программы, вычисляя время выполнения алгоритма.

Задание 2.

Самая долгая сортировка была для возрастающей матрицы с помощью алгоритма быстрой сортировки (qs).

Самая быстрая сортировка была для возрастающей, убывающей, убывающе-возрастающей матрицы с помощью алгоритма сортировки Шелла.