МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В. Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**Лабораторная работа № 3**

по дисциплине: Алгоритмы и структуры данных

тема: «Сравнительный анализ методов сортировки C»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверил:

асс. Солонченко Роман Евгеньевич

Белгород 2023г.

**Лабораторная работа №3**

**«Сравнительный анализ методов сортировки C»**

**Цель работы:** изучение методов сортировки массивов и приобретение навыков в проведении сравнительного анализа различных методов сортировки.**Содержание отчета:**

- Тема лабораторной работы;

- Цель лабораторной работы;

- Условия задач и их решение;

- Вывод.**Задание к лабораторной работе :**

1. Изучить временные характеристики алгоритмов.

2. Изучить методы сортировки:

1) включением;

2) выбором;

3) обменом:

3.1) улучшенная обменом 1;

3.2) улучшенная обменом 2;

4) Шелла;

5) Хоара;

6) пирамидальная.

3. Программно реализовать методы сортировки массивов.

4. Разработать и программно реализовать средство для проведения экспериментов по определению временных характеристик алгоритмов сортировки.

5. Провести эксперименты по определению временных характеристик алгоритмов сортировки. Результаты экспериментов представить в виде таблицы 9, клетки которой содержат количество операций сравнения при выполнении алгоритма сортировки массива с заданным количеством элементов. Провести эксперимент для упорядоченных, неупорядоченных и упорядоченных в обратном порядке массивов (для каждого типа массива заполнить отдельную таблицу).

6. Построить график зависимости количества операций сравнения от количества элементов в сортируемом массиве.

7. Определить аналитическое выражение функции зависимости количества операций сравнения от количества элементов в массиве.

8. Определить порядок функций временной сложности алгоритмов сортировки при сортировке упорядоченных, неупорядоченных и упорядоченных в обратном порядке массивов.

Листинг программы:

Файл standard\_functions.h

// Функция для обмена двух элементов массива  
void swap(void \*a, void \*b, int size);  
  
// Функция генерации рандомного массива размера size  
void generateRandomArray(int \*array, const size\_t size);  
  
// Возвращает 'истину', если массив отсортирован, иначе -- 'ложь'  
bool isOrdered(int \*array, size\_t size);  
  
// Выводит массив array размера size  
void outputArray(int \*array, size\_t size);

Файл standard\_functions.c

#include "standart\_functions.h"  
  
void swap(void \*a, void \*b, int size) {  
 char \*pa = a;  
 char \*pb = b;  
 for (int i = 0; i < size; i++, pa++, pb++) {  
 char t = \*pa;  
 \*pa = \*pb;  
 \*pb = t;  
 }  
}  
  
void generateRandomArray(int \*array, const size\_t size) {  
 srand(time(0));  
  
 for (size\_t i = 0; i < size; i++)  
 array[i] = rand() % 10000;  
}  
  
bool isOrdered(int \*array, size\_t size) {  
 for (size\_t i = 1; i < size; i++)  
 if (array[i] < array[i - 1])  
 return false;  
  
 return true;  
}  
  
  
void outputArray(int \*array, size\_t size) {  
 printf("[");  
  
 for (size\_t i = 0; i < size; i++) {  
 printf("%d", array[i]);  
  
 if (i < size - 1)  
 printf(", ");  
 }  
  
 printf("]\n");  
}

Файл sort.h

// Сортировка включением  
long long insertionSort(int A[], int n);  
  
// Сортировка выбором  
long long selectionSort(int A[], int n);  
  
// Сортировка обменом  
long long bubbleSort(int A[], int n);  
  
// Улучшенная сортировка обменом 1  
long long bubbleSort1(int arr[], int n);  
  
// Улучшенная сортировка обменом 2  
long long bubbleSort2(int arr[], int n);  
  
// Сортировка массива методом Шелла  
long long shellSort(int arr[], int n);  
  
// Сортировка Хоара (быстрая сортировка)  
long long hoarSort(int arr[], int high);  
  
// Пирамидальная сортировка  
long long heapSort(int A[], int n);  
  
// Компаратор для qsort  
int compareQsort(const void \*a, const void \*b);

Файл sort.c

#include "sort.h"  
  
long long insertionSort(int A[], int n) {  
 long long comparisons = 0;  
 int i, j, k;  
 for (j = 1; j < n; j++) {  
 k = A[j];  
 i = j - 1;  
 while (k < A[i] && i >= 0) {  
 comparisons++;  
 A[i + 1] = A[i];  
 i -= 1;  
 }  
 comparisons++;  
 A[i + 1] = k;  
 }  
 return comparisons + (n - 1);  
}  
  
long long selectionSort(int A[], int n) {  
 long long comparisons = 0;  
 int i, j, x, k;  
 for (i = 0; i < n - 1; i++) {  
 x = A[i];  
 k = i;  
 for (j = i + 1; j < n; j++)  
 if (A[j] < x) {  
 k = j;  
 x = A[k];  
 }  
 comparisons += (n - (i + 1));  
 A[k] = A[i];  
 A[i] = x;  
 }  
 return comparisons + (n - 1);  
}  
  
long long bubbleSort(int A[], int n) {  
 long long comparisons = 0;  
 int i, j, k, p;  
 for (i = 0; i < n - 1; i++) {  
 p = 0;  
 for (j = n - 1; j > i; j--) {  
 comparisons++;  
 if (A[j] < A[j - 1]) {  
 k = A[j];  
 A[j] = A[j - 1];  
 A[j - 1] = k;  
 p = 1;  
 }  
 }  
 comparisons += (n - i);  
 //Если перестановок не было, то сортировка выполнена  
 if (!p)  
 break;  
 }  
 return comparisons + (n - 1);  
}  
  
long long bubbleSort1(int arr[], int n) {  
 long long comparisons = 0;  
 int temp;  
 bool swapped;  
 for (int i = 0; i < n - 1; i++) {  
 swapped = false;  
 for (int j = 0; j < n - i - 1; j++) {  
 comparisons++;  
 if (arr[j] > arr[j + 1]) {  
 // меняем элементы местами  
 temp = arr[j];  
 arr[j] = arr[j + 1];  
 arr[j + 1] = temp;  
 swapped = true;  
 }  
 }  
 comparisons += (n - i);  
 // если на текущей итерации не было ни одного обмена,  
 // то массив уже отсортирован и можно завершить процесс  
 if (swapped == false)  
 break;  
 }  
 return comparisons + (n - 1);  
}  
  
long long bubbleSort2(int arr[], int n) {  
 long long comparisons = 0;  
 int i, j, temp;  
 int lastSwapIndex = n - 1;  
 for (int i = 0; i < n - 1; i++) {  
 int currentSwapIndex = -1;  
 for (int j = 0; j < lastSwapIndex; j++) {  
 comparisons++;  
 if (arr[j] > arr[j + 1]) {  
 // меняем элементы местами  
 temp = arr[j];  
 arr[j] = arr[j + 1];  
 arr[j + 1] = temp;  
 currentSwapIndex = j;  
 }  
 }  
 comparisons += (lastSwapIndex + 1);  
 // если на текущей итерации не было ни одного обмена,  
 // то массив уже отсортирован и можно завершить процесс  
 if (currentSwapIndex == -1)  
 break;  
 lastSwapIndex = currentSwapIndex;  
 }  
 return comparisons + (n - 1);  
}  
  
long long shellSort(int arr[], int n) {  
 long long comparisons = 0;  
 // Начинаем с большего шага  
 for (int gap = n / 2; gap > 0; gap /= 2) {  
 comparisons++;  
 // Проходим по элементам массива с шагом gap  
 for (int i = gap; i < n; i++) {  
 // Сохраняем текущий элемент в переменную temp  
 int temp = arr[i];  
 // Сдвигаем предыдущие элементы, которые больше текущего, на один шаг вперед  
 int j;  
 for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {  
 comparisons++;  
 arr[j] = arr[j - gap];  
 }  
 comparisons++;  
 // Вставляем текущий элемент на правильную позицию  
 arr[j] = temp;  
 }  
 comparisons += (n - gap);  
 }  
 return comparisons + 1;  
}  
  
//Эта функция принимает последний элемент в качестве опорного, помещает  
//этот элемент в правильное положение в отсортированном массиве и помещает  
//все меньшие (меньше опорного) элементы слева от него и все большие  
//элементы справа от него  
int partition(int arr[], int low, int high) {  
 int support = arr[high]; // опорный элемент  
 int i = (low - 1); // индекс меньшего элемента  
 for (int j = low; j <= high - 1; j++) {  
 // Если текущий элемент меньше или равен опорному  
 if (arr[j] <= support) {  
 i++; // увеличиваем индекс меньшего элемента  
 swap(&arr[i], &arr[j], sizeof(arr[i]));  
 }  
 }  
 swap(&arr[i + 1], &arr[high], sizeof(arr[i + 1]));  
 return (i + 1);  
}  
  
// Функция для реализации алгоритма быстрой сортировки  
//arr[] - Массив для сортировки,  
//low - Начальный индекс,  
//high - Конечный индекс  
long long q\_sort(int arr[], int low, int high, long long comparisons) {  
 if (low < high) {  
 // separative - это разделительный индекс, arr[sep] сейчас на правильном месте  
 int separative = partition(arr, low, high);  
 comparisons += (2 \* (high - low));  
 // Рекурсивно сортируем элементы до разделителя и после разделителя  
 q\_sort(arr, low, separative - 1, comparisons);  
 q\_sort(arr, separative + 1, high, comparisons);  
 }  
 return comparisons + 1;  
}  
  
long long hoarSort(int arr[], int high) {  
 return q\_sort(arr, 0, high, 0);  
}  
  
void sift(int A[], int L, int R) {  
 int i, j, x, k;  
 i = L;  
 j = 2 \* L + 1;  
 x = A[L];  
 if ((j < R) && (A[j] < A[j + 1]))  
 j++;  
 while ((j <= R) && (x < A[j])) {  
 k = A[i];  
 A[i] = A[j];  
 A[j] = k;  
 i = j;  
 j = 2 \* j + 1;  
 if ((j < R) && (A[j] < A[j + 1]))  
 j++;  
 }  
}  
  
long long heapSort(int A[], int n) {  
 long long comparisons = 0;  
 int L, R, x, i;  
 L = n / 2;  
 R = n - 1;  
 // Построение пирамиды из исходного массива  
 while (L > 0) {  
 comparisons++;  
 L = L - 1;  
 sift(A, L, R);  
 }  
 comparisons++;  
 // Сортировка: пирамида в отсортированный массив  
 while (R > 0) {  
 comparisons++;  
 x = A[0];  
 A[0] = A[R];  
 A[R] = x;  
 R--;  
 sift(A, L, R);  
 }  
 return comparisons + 1;  
}  
  
int compareQsort(const void \*a, const void \*b) {  
 int arg1 = \*(const int \*) a;  
 int arg2 = \*(const int \*) b;  
 if (arg1 < arg2)  
 return -1;  
 if (arg1 > arg2)  
 return 1;  
 return 0;  
}

Файл lab3.h

#define ARRAY\_SIZE(arr) (sizeof(arr) / sizeof((arr)[0]))  
  
typedef struct sortFunction {  
 long long (\*sort)(int[], int);  
 char \*name;  
} sortFunction;  
  
typedef struct generationFunction {  
 void (\*generate)(int \*, size\_t);  
 char \*name;  
} generationFunction;  
  
void timeExperiment();

Файл lab3.c

#include "lab3.h"  
  
void checkTime(long long (\*sortFunc)(int \*, int), void (\*generateFunc)(int \*, size\_t),  
 size\_t size, char \*experimentName) {  
 static size\_t runNum = 1;  
 static int odometer[100000000];  
  
 generateFunc(odometer, size);  
  
 printf("Запуск #%zu | ", runNum++);  
 printf("Название: %s\n", experimentName);  
  
 long long сomparison = sortFunc(odometer, size);  
  
 printf("Состояние: ");  
  
 if (isOrdered(odometer, size)) {  
 printf("OK! Подсчётов %lld\n\n", сomparison);  
  
 char filename[256];  
  
 sprintf(filename, "data/%s.csv", experimentName);  
  
 FILE \*f = fopen(filename, "a");  
  
 if (f == NULL) {  
 printf("Ошибка открытия файла %s", filename);  
  
 exit(1);  
 }  
  
 fprintf(f, "%llu; %lld\n", size, сomparison);  
  
 fclose(f);  
 } else {  
 printf("Неправильно!\n");  
  
 outputArray(odometer, size);  
 exit(1);  
 }  
}  
  
  
void timeExperiment() {  
 sortFunction sorts[] = {  
 {insertionSort, "insertionSort"},  
 {selectionSort, "selectionSort"},  
 {bubbleSort, "bubbleSort"},  
 {bubbleSort1, "bubbleSort1"},  
 {bubbleSort2, "bubbleSort2"},  
 {shellSort, "shellSort"},  
 {hoarSort, "hoarSort"},  
 {heapSort, "heapSort"},  
 };  
 const unsigned FUNCS\_N = ARRAY\_SIZE(sorts);  
 generationFunction generation[] = {  
 {generateRandomArray, "Random"}  
 };  
 const unsigned CASES\_N = ARRAY\_SIZE(generation);  
  
 for (size\_t size = 5; size <= 45; size += 5) {  
 printf("------------------------------------------------\n");  
 printf("Размер: %llu\n", size);  
  
 for (size\_t i = 0; i < FUNCS\_N; i++)  
 for (size\_t j = 0; j < CASES\_N; j++) {  
 static char filename[128];  
  
 sprintf(filename, "%s%sTime", sorts[i].name, generation[j].name);  
  
 checkTime(sorts[i].sort, generation[j].generate, size,  
 filename);  
 }  
  
 printf("\n");  
 }  
}

Файл main.c

int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
  
 timeExperiment();  
 return 0;  
}

Временные характеристики алгоритмов:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Сортировка | Количество элементов в массиве | | | | | | | | |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 |
| Включением | 13 | 35 | 90 | 125 | 195 | 267 | 340 | 477 | 606 |
| Выбором | 14 | 54 | 119 | 209 | 324 | 464 | 629 | 819 | 1034 |
| Обменом | 28 | 84 | 214 | 370 | 600 | 880 | 1195 | 1575 | 2005 |
| Обменом 1 | 25 | 105 | 238 | 394 | 640 | 865 | 1159 | 1614 | 2020 |
| Обменом 2 | 23 | 105 | 238 | 328 | 604 | 805 | 1107 | 1612 | 1910 |
| Шелла | 22 | 61 | 90 | 162 | 222 | 262 | 380 | 435 | 510 |
| Хоара | 11 | 21 | 31 | 41 | 51 | 61 | 71 | 81 | 91 |
| Пирамидальная | 8 | 16 | 23 | 31 | 38 | 46 | 53 | 61 | 68 |

График зависимости функций временной сложности:

Порядок функций временной сложности:

|  |  |
| --- | --- |
| Сортировки | Порядок функций временной сложности |
| Включением |  |
| Выбором |  |
| Обменом |  |
| Обменом 1 |  |
| Обменом 2 |  |
| Шелла |  |
| Хоара |  |
| Пирамидальная |  |

Вывод: в ходе выполнения лабораторной работы были изучены методы сортировки массивов и приобретены навыки в проведении сравнительного анализа различных методов сортировки.