1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. [Институт компьютерных наук и кибербезопасности](https://dl.spbstu.ru/)

**ЛАБОРАТОРНАЯ РАБОТА № 5**

1. «**Сортировки**»
2. по дисциплине «Алгоритмизация и программирование»
3. Выполнил
4. студент гр. 5151004/30001 Лясович С.Д.

<*подпись*>

1. Преподаватель
2. асс. преподавателя

<*подпись*>

1. Санкт-Петербург
2. 2024
3. **Цель работы**

В ходе работы необходимо реализовать 3 сортировки: 2 усовершенствованные и одну простую.

1. **Постановка задачи**

Необходимо реализовать 3 сортировки, выбор пал на сортировку «Пузырёк», «Быструю» сортировку и сортировку вставками.

1. **Теоретические исследования**
   1. **Сортировка «Пузырёк»**

Принцип работы сортировки «Пузырёк»: в заданном массиве, по очереди рассматриваются пары чисел, если i-тый элемент массива больше элемента i+1, то элементы меняются местами, иначе, рассматривается следующая пара чисел. Действия продолжаются до тех пор, пока весь массив не будет отсортирован.

* + 1. **Сложность сортировки «Пузырёк»**

Количество сравнений при пузырьковой сортировке постоянно. Таким образом, данный алгоритм имеет временную сложность O (N2). И постоянную сложность O(1), так как для его работы память затрачивается лишь для хранения временной переменной.

* 1. **«Быстрая» сортировка**

Принцип работы «Быстрой» сортировки: для заданного массива выбирается один элемент, который разбивает остальные элементы на два подмножества — те, что меньше, и те, что не меньше него. Та же процедура рекурсивно применяется и к двум полученным подмножествам. Если в подмножестве менее двух элементов, то сортировать нечего, и рекурсия завершается.

**3.2.1 Сложность «Быстрой» сортировки**

Пусть, n – степень двойки, n=2k (k = log2n). Во время первичного прохода по массиву выполняется n-1 сравнение. В итоге формируются две группы, имеющие величину n/2. Далее происходит проверка каждой подгруппы, что требует n/2 сравнений. Суммарное количество сравнений на данном этапе равняется 2(n/2) = n. На следующем шаге анализируются уже четыре группы данных, и нужно 4(n/4) сопоставлений, и так далее. Цикл завершается после прохождения k сравнений, когда сформированные подгруппы содержат в себе по одному элементу. Таким образом, для стандартного массива сложность быстрой сортировки можно оценить, как O(n log2 n).

* 1. **Сортировка вставками**

Принцип работы сортировки вставками: весь массив делится на 2 части: отсортированную и неотсортированную. Из неотсортированной части извлекается любой элемент. В отсортированной части массива происходит поиск подходящего места для этого элемента. Элемент вставляется в нужное место, в результате чего отсортированная часть массива увеличивается, а неотсортированная уменьшается.

* + 1. **Сложность сортировки вставками**

Процесс сортировки вставками включает в себя проход по каждому элементу массива и вставку текущего элемента на корректное место в уже отсортированную часть массива. Количество сравнений меняется в зависимости от начального состояния массива. Именно поэтому, общая временная сложность сортировки вставками в среднем случае равна O(N2).

1. **Описание решения**

В созданной программе существует 2 варианта входных данных, первый вариант-создание программой произвольного массива, второй вариант-считывание массива из файла пользователя. После выбора одного из вариантов, в каждой сортировке происходит динамическое выделения памяти для копии этого массива и сама сортировка. После того, как весь массив отсортирован, создаются несколько файлов: файл с отсортированным массивом, файл с характеристиками массива и файл журнал, в котором отражены все действия происходящие по ходу работы алгоритма. Ниже, на рисунке 1 представлена блок-схема сортировки «Пузырёк», на рисунке 2 представлена блок-схема «Быстрой» сортировки, на рисунке 3 представлена блок-схема сортировки вставками.

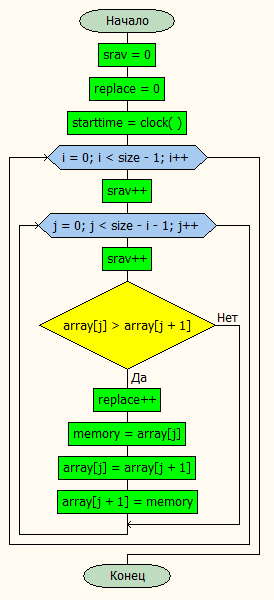


Рисунок 1 ­­­‒ Блок-схема сортировки «Пузырёк»

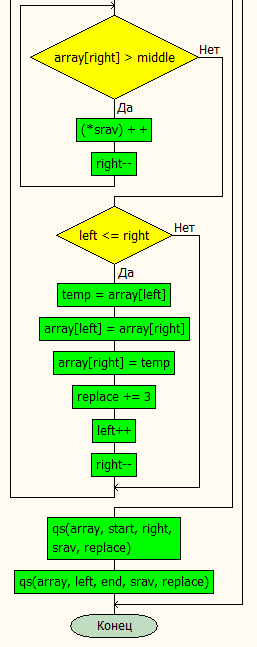
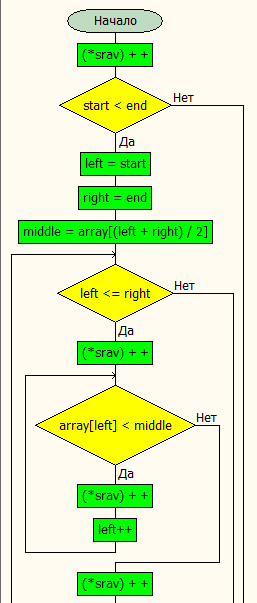


Рисунок 2 ­­­‒ Блок-схема «Быстрой» сортировки

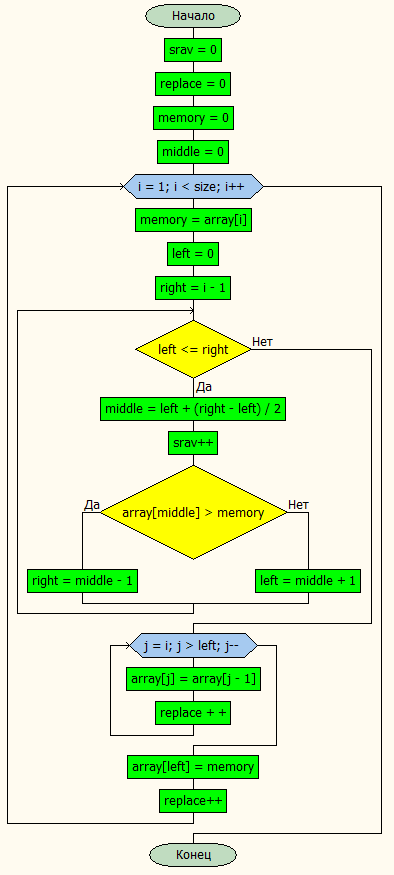


Рисунок 3 ­­­‒ Блок-схема сортировки вставками

1. **Тестирование и результаты работы программы**

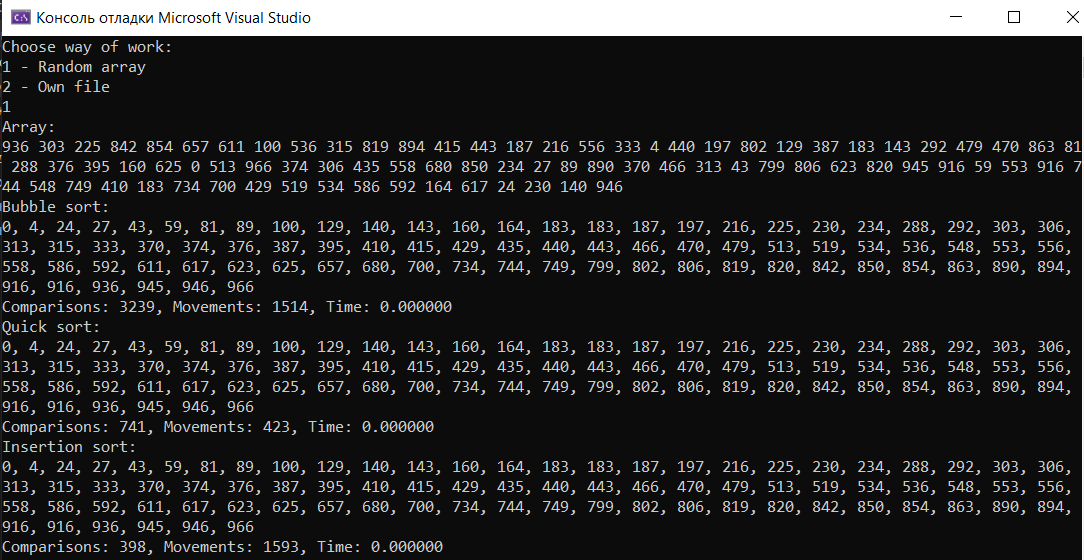


Рисунок 4 ­­­‒ Сортировка произвольного массива

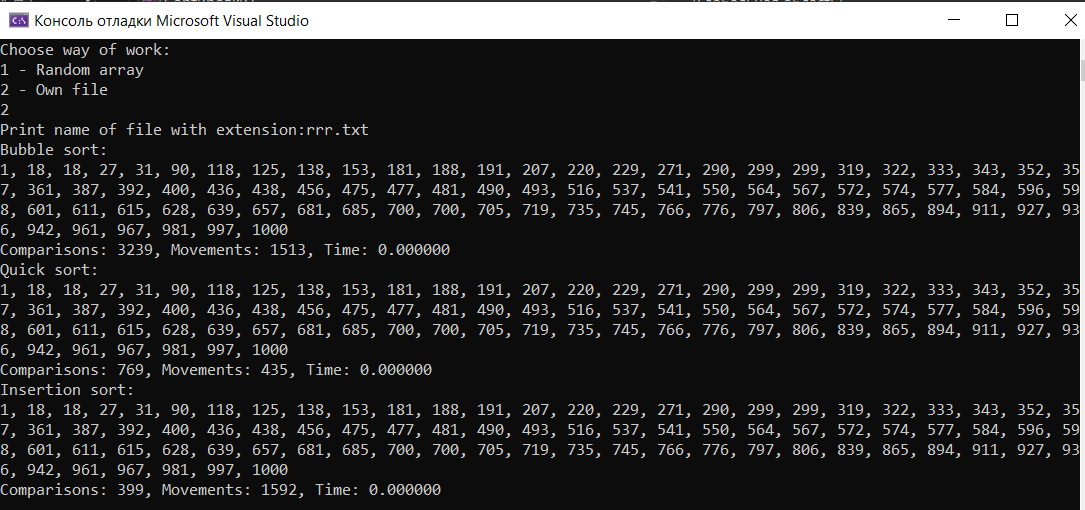


Рисунок 5 ­­­‒ Сортировка массива из файла

Таблица 1 ­­­‒ Таблица зависимости времени работы сортировок в секундах от количества входных данных

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Сортировка «Пузырёк»** | **«Быстрая» сортировка** | **Сортировка вставками** |
| **5000** | 0.046000 | 0.001000 | 0.016000 |
| **10000** | 0.200000 | 0.001000 | 0.057000 |
| **15000** | 0.464000 | 0.002000 | 0.125000 |
| **20000** | 0.833000 | 0.003000 | 0.223000 |
| **25000** | 1.314000 | 0.003000 | 0.355000 |
| **30000** | 1.995000 | 0.004000 | 0.506000 |
| **35000** | 2.958000 | 0.005000 | 0.696000 |
| **40000** | 4.110000 | 0.007000 | 0.899000 |
| **45000** | 5.536000 | 0.007000 | 1.127000 |
| **50000** | 7.033000 | 0.008000 | 1.403000 |

Рисунок 6 ­­­‒ Зависимость времени работы алгоритма сортировки «Пузырёк» от количества элементов во входном массиве

Рисунок 7 ­­­‒ Зависимость времени работы алгоритма «Быстрой» сортировки от количества элементов во входном массиве

Рисунок 8 ­­­‒ Зависимость времени работы алгоритма сортировки вставками от количества элементов во входном массиве

1. **Выводы**

В результате работы были усовершенствованы навыки работы с файлами и генерацией случайных чисел. Получены навыки работы с наиболее популярными сортировками, а также навыки замера времени работы сортировок. Так же, основываясь на приведённой выше таблице, можно сделать выводы о том, что Быстрая сортировка является наиболее эффективной при обработке массива данных свыше 10000 элементов.

Список используемых источников

1. Брайан Керниган, Деннис Ритчи «Язык программирования Си» [Электронный ресурс] – Режим доступа: <http://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf> , свободный.

ПРИЛОЖЕНИЕ А

Листинг программы «Сортировки»

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#define SIZE 300

#include <stdlib.h>

#include <string.h>

#include <time.h>

FILE\* rezultfile;

FILE\* characteristicsfile;

FILE\* logfile;

//Пузырьковая сортировка

void bublesort(int \*inputarray,const int size) {

fprintf(logfile, "1.Buble sort started!\n");

int\* array = 0;

array = (int\*)malloc(sizeof(int) \* size);// определяем сколко памяти необходимо для массива

memset(array, 0, size);//для того, чтобы мусор опять не вылез в массиве

memcpy(array, inputarray, sizeof(int) \* size);

int srav = 0;

int replace = 0;

long double starttime;

long double stoptime;

starttime = clock();

for (int i = 0; i < size - 1; i++) {

srav++;

for (int j = 0; j < size - i - 1; j++) {

srav++;

if (array[j] > array[j + 1]) {

replace++;

int memory = array[j];

array[j] = array[j + 1];

array[j + 1] = memory;

}

}

}

stoptime = clock();

fprintf(logfile, " The array is sorted!\n");

printf("Bubble sort:\n");

fprintf(rezultfile, "%d", array[0]);

printf("%d", array[0]);

for (int i = 1; i < size; i++) {

fprintf(rezultfile, ", %d", array[i]);

printf(", %d", array[i]);

}

printf("\n");

fprintf(rezultfile, "\n");

fprintf(logfile, " The result is written to a file!\n");

fprintf(characteristicsfile,"Bubble sort: %d, %d, %Lf\n", srav, replace, (stoptime - starttime) / CLOCKS\_PER\_SEC);

fprintf(logfile, " Characteristics are recorded in a file!\n");

free(array);

}

//Быстрая сортировка

void qs(int\* array, int start, int end, unsigned long\* srav, unsigned long\* replace) {

(\*srav)++;

if (start < end) {

int left = start;

int right = end;

int middle = array[(left + right) / 2]; // Центр сравнения

while (left <= right) {

(\*srav)++;

while (array[left] < middle) {

(\*srav)++;

left++;

}

(\*srav)++;

while (array[right] > middle) {

(\*srav)++;

right--;

}

if (left <= right) {

int temp = array[left];

array[left] = array[right];

array[right] = temp;

\*replace += 3;

left++;

right--;

}

}

qs(array, start, right, srav, replace);

qs(array, left, end, srav, replace);

}

}

void quicksort(int\* inputarray, int size) {

fprintf(logfile, "2.Quick sort started!\n");

int\* array = 0;

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

memset(array, 0, size);

memcpy(array, inputarray, sizeof(int) \* size);

int srav = 0;

int replace = 0;

long double starttime;

long double stoptime;

starttime = clock();

qs(array, 0, size - 1,&srav,&replace);

stoptime = clock();

fprintf(logfile, " The array is sorted!\n");

printf("Quick sort:\n");

fprintf(rezultfile, "%d", array[0]);

printf("%d", array[0]);

for (int i = 1; i < size; i++) {

fprintf(rezultfile, ", %d", array[i]);

printf(", %d", array[i]);

}

printf("\n");

fprintf(rezultfile, "\n");

fprintf(logfile, " The result is written to a file!\n");

fprintf(characteristicsfile,"Quick sort: %d, %d, %Lf\n", srav, replace, (stoptime - starttime) / CLOCKS\_PER\_SEC);

fprintf(logfile, " Characteristics are recorded in a file!\n");

free(array);

}

//Сортировка вставками

void inputsort(int\* inputarray, int size) {

fprintf(logfile, "3.Insertion sort started!\n");

int\* array = 0;

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

memset(array, 0, size);

memcpy(array, inputarray, sizeof(int) \* size);

int srav = 0;

int replace = 0;

long double starttime;

long double stoptime;

starttime = clock();

int memory = 0;

int middle = 0;

for (int i = 1; i < size; i++) {

memory = array[i];

int left = 0;

int right = i - 1;

while (left <= right) {

middle = left + (right - left) / 2;

srav++;

if (array[middle] > memory) {

right = middle-1;

}

else {

left = middle + 1;

}

}

for (int j = i; j > left; j--) {

array[j] = array[j - 1];

replace ++;

}

array[left] = memory;

replace++;

}

stoptime = clock();

fprintf(logfile, " The array is sorted!\n");

printf("Insertion sort:\n");

fprintf(rezultfile, "%d", array[0]);

printf("%d", array[0]);

for (int i = 1; i < size; i++) {

fprintf(rezultfile, ", %d", array[i]);

printf(", %d", array[i]);

}

fprintf(rezultfile, "\n");

printf("\n");

fprintf(logfile, " The result is written to a file!\n");

fprintf(characteristicsfile,"Insertion sort: %d, %d, %Lf\n", srav, replace, (stoptime - starttime) / CLOCKS\_PER\_SEC);

fprintf(logfile, " Characteristics are recorded in a file!\n");

free(array);

}

int main() {

time\_t mytime = time(NULL);

struct tm\* now = localtime(&mytime);

char rezname[200];

sprintf(rezname, "rezult\_%02d\_%02d\_%02d\_%02d\_%02d.txt", now->tm\_min, now->tm\_hour, now->tm\_mday, now->tm\_mon + 1, now->tm\_year + 1900);

rezultfile = fopen(rezname, "a");

char chname[200];

sprintf(chname, "characteristics\_%02d\_%02d\_%02d\_%02d\_%02d.txt",now->tm\_min, now->tm\_hour, now->tm\_mday, now->tm\_mon + 1, now->tm\_year + 1900);

characteristicsfile = fopen(chname, "a");

char logname[200];

sprintf(logname, "log\_%02d\_%02d\_%02d\_%02d\_%02d.txt",now->tm\_min, now->tm\_hour, now->tm\_mday, now->tm\_mon + 1, now->tm\_year + 1900);

logfile = fopen(logname, "a");

int way = 0;

printf("Coose way of work:\n1 - random array\n2 - own file\n");

scanf("%d", &way);

if (way == 1) {

int random\_array[SIZE];

srand(time(0));

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

random\_array[i] = rand() % 1000;

}

printf("Array:\n");

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

printf("%d ", random\_array[i]);

}

printf("\n");

fprintf(logfile, "Created a random array!\n");

bublesort(random\_array, SIZE);

quicksort(random\_array, SIZE);

inputsort(random\_array, SIZE);

}

else if (way == 2) {

int \*array;

array = (int\*)malloc(sizeof(int) \* SIZE);

char nameoffile[100];

int i = 0;

int size=0;

printf("Print name of file with extension:");

scanf("%s", &nameoffile);

FILE\* workfile = fopen(nameoffile, "r");

if (workfile == NULL) {

printf("Error.\n");

}

else {

for (i = 0; i < SIZE && !feof(workfile); i++) {

fscanf(workfile, "%d", &array[i]);

}

size = i;

}

fclose(workfile);

fprintf(logfile, "Converted users file!\n");

bublesort(array, size);

quicksort(array, size);

inputsort(array, size);

}

else {

printf("Restart a program and write a correct way of work!");

}

fprintf(logfile, "The array is sorted in three ways!\n");

return 0;

}