Выполнила

студентка группы КТбо1-2 Н. А. Свидич

Принял

ассистент ИКТИБ Д. С. Кочубей

Таганрог 2023

Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

(ФГАОУ ВО «ЮФУ»)

Институт компьютерных технологий и информационной безопасности

Кафедра системного анализа и телекоммуникаций

**ЛАБОРАТОРНАЯ РАБОТА № 6**

по дисциплине «Основы алгоритмизации и программирования»

Вариант 2

на тему:

«Алгоритмы сортировки»

**Содержание**

Содержание 2

Техническое задание 3

Цель работы 3

Задача 3

Ход работы 4

Алгоритм 4

Блок-схема 6

Написание программы 8

Примеры работы программы 9

Вывод 10

Листинг (Приложение а) 11

**Техническое задание**

**Цель работы**

Цель данной лабораторной работы – ознакомиться с сортировкой данных и ее видами, а также запрограммировать на языке Си некоторые из алгоритмов сортировок в соответствии с индивидуальным заданием.

**Задача**

Вариант 2.

Написать программу, реализующую несколько алгоритмов сортировок, и провести их сравнительный анализ на одном входном наборе данных. Для выполнения лабораторной работы необходимо подготовить исходный файл данных следующим образом. 10

Написать программу, которая должна сгенерировать массив и з 1000000 (один миллион) случайных целых чисел и записать их в файл input.txt. Данный файл представляет собой исходный массив для сортировки.

Произвести сортировку алгоритмами: сортировка пузырьком, сортировка Шелла, сортировка слиянием.

**Ход работы**

**Алгоритмы**

Сортировка пузырьком:

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим наибольшим элементом, а наименьший элемент перемещается на одну позицию к началу массива. Блок-схема алгоритма показана на рисунке 1.

Сортировка Шелла (Рисунок 2):

1. Задаем исходный массив элементов, который нужно отсортировать.

2. Выбираем шаг сортировки. Шаг – это расстояние между элементами, которые будут сравниваться и меняться местами. Шаг может быть выбран различными способами, например, делением на 2 или умножением на фактор.

3. Пока шаг больше 0, выполняем следующие действия:

3.1. Проходим по массиву, начиная с элемента с индексом, равным шагу, и сравниваем его с предыдущим элементом на расстоянии шага. Если текущий элемент меньше предыдущего, меняем их местами.

3.2. Повторяем шаг 3.1 для всех элементов массива с индексами, большими шага.

3.3. Уменьшаем шаг, например, делением на 2 или умножением на фактор.

4. Повторяем шаги 3.1-3.3, пока шаг не станет равным 0.

Сортировка слиянием (Рисунок 3):

1. Разделение: Исходный массив разделяется на две равные (или почти равные) половины. Это делается путем нахождения середины массива и создания двух новых массивов, в которые будут скопированы элементы из левой и правой половин.

2. Рекурсивная сортировка: Каждая половина массива рекурсивно сортируется с помощью алгоритма сортировки слиянием. Этот шаг повторяется до тех пор, пока размер каждой половины не станет равным 1.

3. Слияние: Отсортированные половины массива объединяются в один отсортированный массив. Для этого создается новый массив, в который будут последовательно добавляться элементы из левой и правой половин. При добавлении элементов выбирается наименьший элемент из двух половин и добавляется в новый массив.

Сравнение алгоритмов приведено в Таблице 1.

Таблица 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Сортировка | Временная сложность | | | Сложность пространства | Стабильность |
| Луч. | Ср. | Худш. |
| Пузырьком | O(n) | O(n2) | O(n2) | O(1) | Да |
| Шелла | O(n log n) | O(n log n) | O(n2) | O(n) | Нет |
| Слиянием | O(n.log(n)) | O(n.log(n)) | O(n.log(n)) | O(n) | Да |

**Блок-схемы**

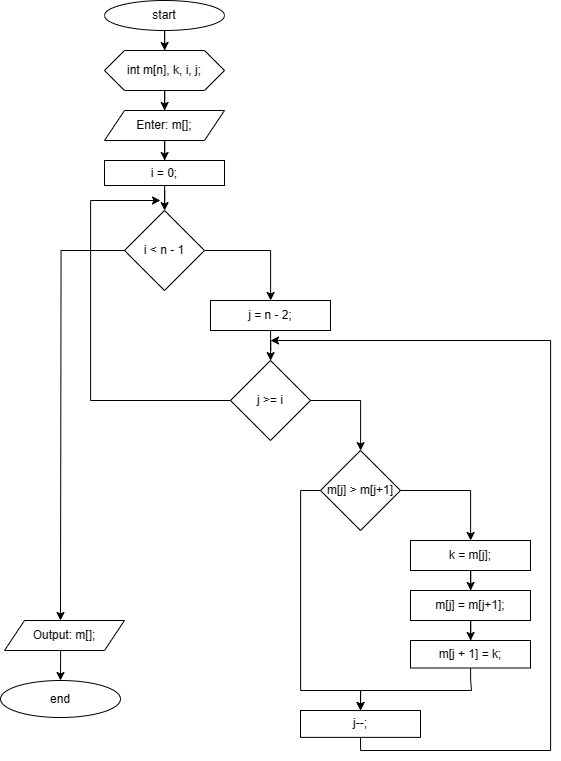


Рисунок 1 – Блок-схема алгоритма сортировки пузырьком.

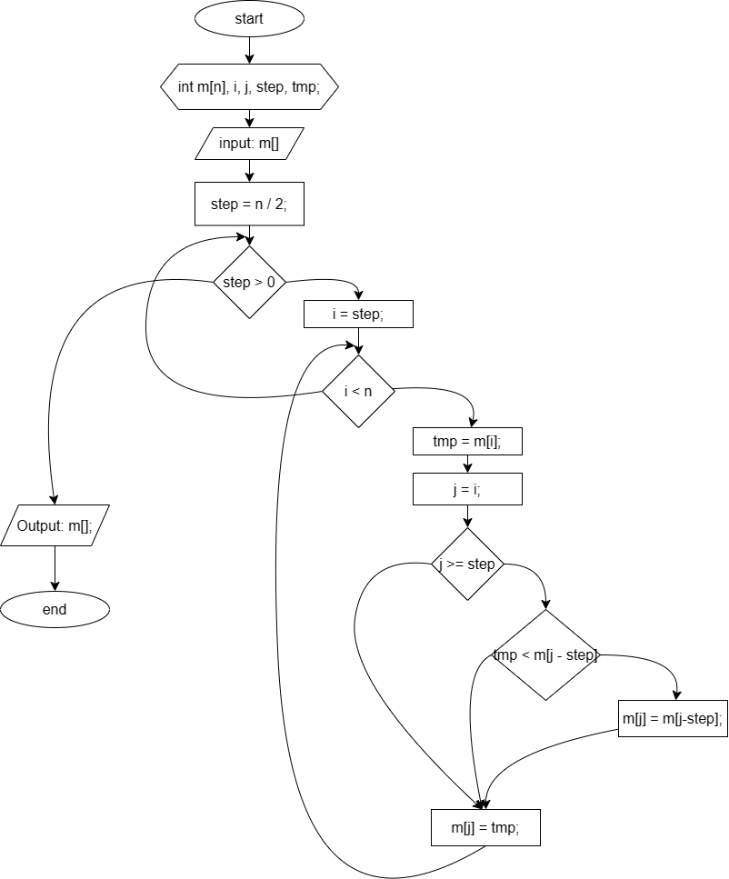


Рисунок 2 – Блок-схема алгоритма сортировки Шелла.

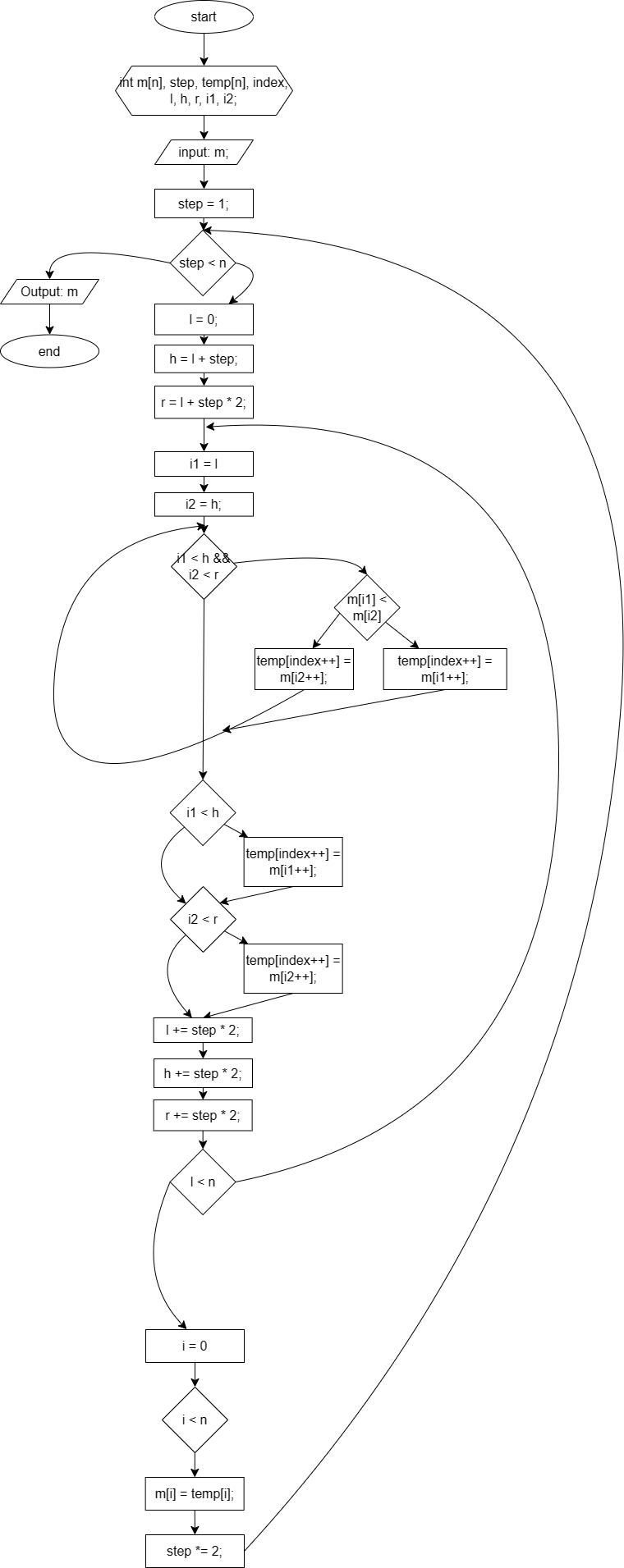


Рисунок 3 – Блок-схема сортировки слиянием.

**Написание программы**

1. Определение и инициализация переменных.

Для удобства, будут использованы 4 массива целых чисел, длиной n: m – исходный массив, bbl – массив для сортировки пузырьком, sh – массив для сортировки Шелла, mrg – массив для сортировки слиянием.

Для вывода массивов в файл используется переменная output, хранящая адрес файла.

Для сортировки используются отдельные функции с переменными:

В сортировке пузырьком: k - вспомогательная целочисленная переменная для хранения промежуточного значения.

В сортировке Шелла: step – задает шаг в сортировке массива (n/2), temp - вспомогательная целочисленная переменная для хранения промежуточного значения.

В сортировке слиянием: step – шаг разбиения массива, temp[n] – дополнительный массив, который записываются отсортированные элементы, l - левая граница участка, h - середина участка, r - правая граница участка, i1 и i2 index - индекс результирующего массива.

1. Основные процессы.

Для заполнения исходного массива m используется функция rand(), которая генерирует случайные числа. Массивы bbl, shl, mrg приравниваются к исходному для получения одинаковых массивов и последующей сортировки.

Каждый массив сортируется отдельным методом и полученные массивы, вместе с исходным, записываются в файл.

На рисунке 4 показан результат работы полученного программного кода.

**Примеры работы программы**

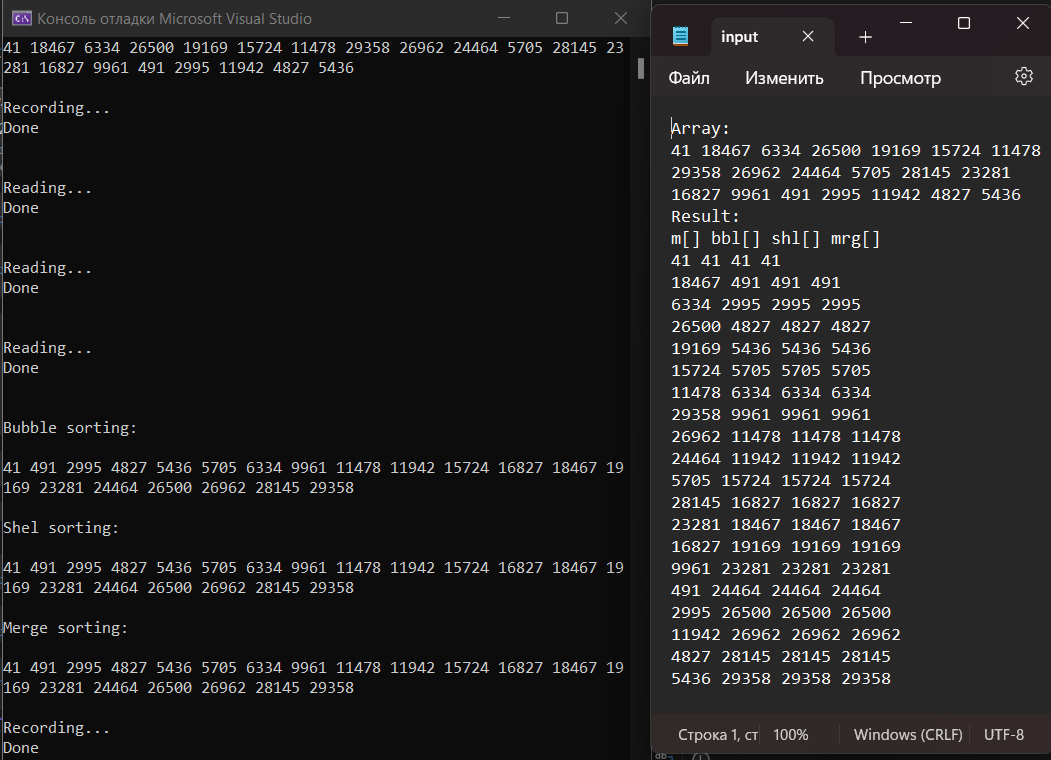


Рисунок 4 – Скриншот выполнения программы.

**Вывод**

# Изучены сортировки данных и их виды, а также запрограммированы на языке Си некоторые из алгоритмов сортировок в соответствии с индивидуальным заданием (сортировки пузырьком, Шелла, слиянием).

# **Листинг (Приложение а)**

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#pragma warning(disable : 4996)

#define n 20 // Константа n, для определния длины массивов

// Функция генерирования случайных чисел, запись их в массивы

void generator(int\* m, int\* bbl, int\* shl, int\* mrg) {

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

m[i] = rand();

/\*bbl[i] = m[i];

shl[i] = m[i];

mrg[i] = m[i];\*/

}

}

// Функция вывода массива в файл

void output(int\* m) {

FILE\* output = fopen("..\\input.txt", "w");

if (output == NULL) perror("Opening file error");

else

{

puts("\nRecording...");

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

fprintf(output, "%i ", m[i]);

}

puts("Done\n");

}

fclose(output);

}

// Функция заполнения массива из данных файла

void input(int\* m) {

FILE\* output = fopen("..\\input.txt", "r");

if (output == NULL) perror("Opening file error");

else

{

puts("\nReading...");

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

fscanf(output, "%i", &m[i]);

}

puts("Done\n");

}

fclose(output);

}

// Функция вывода массива в консоль

void print(int\* m) {

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

printf("%i ", m[i]);

printf("\n");

}

// Функция вывода массивов в файл

void result(int\* m, int\* bbl, int\* shl, int\* mrg) {

FILE\* output = fopen("..\\input.txt", "w");

if (output == NULL) perror("Opening file error");

else

{

puts("\nRecording...");

fputs("Array: \n", output);

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

fprintf(output, "%i ", m[i]);

}

fputs("\nResult: \n", output);

fputs("m[] bbl[] shl[] mrg[]\n", output);

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

fprintf(output, "%i ", m[i]);

fprintf(output, "%i ", bbl[i]);

fprintf(output, "%i ", shl[i]);

fprintf(output, "%i \n", mrg[i]);

}

puts("Done\n");

}

fclose(output);

}

// Функция сортировки пузырьком

// Сложность О (n2)

void bubble\_sort(int\* m) {

int k; // Вспомогательная переменная для хранения промежуточного значения

puts("\nBubble sorting: \n");

// Цикл прохождения по каждому элементу массива

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

// Цикл сравнения элементов

for (int j = n - 2; j >= i; j--) {

// Если элемент больше следующего, вспомогательная переменная получает его значение,

// следующий элемент становится на его место

// следующий индекс заполняется значением из буфера

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

k = m[j];

m[j] = m[j + 1];

m[j + 1] = k;

}

}

}

print(m);

}

// Функция сортировки Шела

// Сложность O(1)

void Shel\_sort(int\* m) {

puts("\nShel sorting: \n");

int i, j, step, tmp;

for (step = n / 2; step > 0; step /= 2) // Определение шага сортировки

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

{

tmp = m[i];

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

{

if (tmp < m[j - step])

m[j] = m[j - step];

else

break;

}

m[j] = tmp;

}

print(m);

}

// Функция сортировки слиянием

// Сложность O(n log2 n)

void Merge\_sort(int\* m) {

puts("\nMerge sorting: \n");

//

int step = 1; // Шаг разбиения последовательности

int\* temp = (int\*)malloc(n \* sizeof(int)); // Дополнительный массив

while (step < n) // Пока шаг меньше длины массива

{

int index = 0; // Индекс результирующего массива

int l = 0; // Левая граница участка

int h = l + step; // Середина участка

int r = l + step \* 2; // Правая граница участка

do

{

h = h < n ? h : n; // Сортируемый участок не выходит за границы последовательности

r = r < n ? r : n;

int i1 = l, i2 = h; // Индексы сравниваемых элементов

for (; i1 < h && i2 < r; ) // Пока i1 не дошёл до середины и i2 не дошёл до конца

{

if (m[i1] < m[i2]) { temp[index++] = m[i1++]; } // Заполняем участок результирующей последовательности

else { temp[index++] = m[i2++]; }

}

// Или i1 < m или i2 < r - только один из операторов while может выполниться

while (i1 < h) temp[index++] = m[i1++]; // Заносим оставшиеся элементы сортируемых участков

while (i2 < r) temp[index++] = m[i2++]; // В результирующий массив

l += step \* 2; // Перемещаемся на следующий сортируемый участок

h += step \* 2;

r += step \* 2;

} while (l < n); // Пока левая граница сортируемого участка - в пределах последовательности

for (int i = 0; i < n; i++) // Переносим сформированный массив обратно в a

m[i] = temp[i];

step \*= 2; // Увеличиваем в 2 раза шаг разбиения

}

//

print(m);

}

int main()

{

int m[n], bbl[n], shl[n], mrg[n];

// Генерация

generator(m, bbl, shl, mrg);

// Вывод в консоль исходного массива

print(m);

output(m);

// Заполнение массивов для сортировки из файла

input(bbl);

input(shl);

input(mrg);

bubble\_sort(bbl); // Сортировка пузырьком

Shel\_sort(shl); // Сортировка Шелла

Merge\_sort(mrg); // Сортировка слиянием

result(m, bbl, shl, mrg); // Вывод полученных массивов в файл

return 0;

}