МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Сортировки

Студентка гр. 9381 _____ Москаленко Е.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Реализовать алгоритм сортировки Шелла на примере целочисленного массива.

Задание.

Вариант 18.

Сортировка Шелла.

Основные теоретические положения.

Сортировка Шелла (англ. *Shell sort*) — алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений , а завершается сортировка Шелла упорядочиванием элементов при d = 1 (сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Пример сортировки Шелла:

Исходный массив	32	95	16	82	24	66	35	19	75	54	40	43	93	68	
После сортировки с шагом 5	32	35	16	68	24	40	43	19	75	54	66	95	93	82	6 обменов
После сортировки с шагом 3	32	19	16	43	24	40	54	35	75	68	66	95	93	82	5 обменов
После сортировки с шагом 1	16	19	24	32	35	40	43	54	66	68	75	82	93	95	15 обменов

Описание алгоритма.

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

- отсутствие потребности в памяти под стек;
- отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до $O(n^2)$, что хуже, чем худшее гарантированное время для сортировки Шелла.

Пользователю предлагается выбрать, как вводить размер массива и его элементы: через консоль или путем считывания из файла.

После считывания создается вектор **vec,** все элементы которого равны элементам считанного массива, и сортируется с помощью библиотечной функции **sort()**.

Затем пользователь выбирает реализацию сортировки Шелла: более эффективную **по Седжвику** ($\sim O(n^{7/6})$) или **классическую** ($\sim O(n^2)$) (при каждой итерации расстояние между сортируемыми элементами уменьшается в два раза). В зависимости от выбора вызывается функция с той или иной реализацией.

Отсортированный массив выводится на экран и вызывается функция **check()**, сравнивающая элементы вектора vec и массива array. Если все элементы одинаковые, то выводится сообщение, что сортировка работает корректно и тест пройден, в ином случае – тест не пройден.

ФУНКЦИИ:

1) **template <typename T> void printArr(T* arr, int size, int f, int l)** – шаблонная функция печати массива.

T* arr – массив

int size – размер массива

int f, int l – индексы элементов, которые надо подсветить при выводе (замене)

2) **template <typename T> void insert(T* array, int step, int size) –** шаблонная функция сортировки вставками с использованием шага. Вызывается в shellSortSedgwick() и shell().

T* array – массив

int step – расстояние, на котором друг от друга сортируются элементы массива

int size – размер массива

3) **template <typename T> void shellSortSedgwick(T* array, int size) –** функция сортировки Шелла с использованием последовательности Седжвика.

T* array – массив

int size – размер массива

Последовательность Седжвика имеет вид:

$$inc[s] = \begin{cases} 9*2^s - 9*2^{s/2} + 1, \text{ если } s \text{ четно} \\ 8*2^s - 6*2^{(s+1)/2} + 1, \text{ если } s \text{ нечетно} \end{cases}$$

Для массива инкремент(приращений) выделяется буфер размером в 50 элементов. В цикле он заполняется по выше названной формуле до тех пор пока, текущая инкремента хотя бы в 3 раза меньше количества элементов в массиве (3*inc[s] < size).

При использовании таких приращений среднее количество операций: $O(n^{7/6})$, в худшем случае - порядка $O(n^{4/3})$.

4) **template <typename T> void shell(T* array, int size)** – сортировка массива с использованием классической последовательности Шелла – первый элемент (инкремента, расстояние) равен длине массива, деленной на 2, каждый следующий вдвое меньше предыдущего.

Асимптотика в худшем случае – $O(n^2)$.

T* array – массив

int size – размер массива

5) **template <typename T> int check(vector <T> vec, T* array, int size)-** шаблонная функция сравнения отсортированных вектора и массива. Возвращает 1, если все элементы совпадают, 0 – в ином случае.

vector <T> vec – вектор типа T

T* array – массив

int size – размер массива

6) **template <typename T> void typeSort(T* array, int size) –** шаблонная функция выбора одной из двух реализаций сортировки Шелла (пользователь вводит 1 или 2 в зависимости от желаемой реализации).

T* array – массив

int size – размер массива

Тестирование.

№	Входные данные	Вывод
1	5 4 8 1 0 9 1 // Sedgwick	The array of increments: 1 5 STEP = 5 STEP = 1 ЗАМЕНА № 1 Change 8 and 1 4 1 8 0 9 ЗАМЕНА № 2 Change 4 and 1 1 4 8 0 9 ЗАМЕНА № 3 Change 8 and 0 1 4 0 8 9 ЗАМЕНА № 4 Change 4 and 0 1 0 4 8 9 ЗАМЕНА № 5

		Change 1 and 0 0 1 4 8 9 ИТОГ: 0 1 4 8 9 Results of std::sort and Shell sorting are SAME. Test passed.
2	9 10 9 8 7 6 5 4 3 2 2 //division by 2	STEP = 4 ЗАМЕНА № 1 Change 10 and 6 6 9 8 7 10 5 4 3 2 ЗАМЕНА № 2 Change 9 and 5 6 5 8 7 10 9 4 3 2 ЗАМЕНА № 3 Change 8 and 4 6 5 4 7 10 9 8 3 2 ЗАМЕНА № 4 Change 7 and 3 6 5 4 3 10 9 8 7 2 ЗАМЕНА № 5 Change 10 and 2 6 5 4 3 2 9 8 7 10 ЗАМЕНА № 6 Change 6 and 2 2 5 4 3 6 9 8 7 10 STEP = 2 ЗАМЕНА № 7 Change 5 and 3 2 3 4 5 6 9 8 7 10 ЗАМЕНА № 8 Change 9 and 7 2 3 4 5 6 7 8 9 10 STEP = 1

		ИТОГ: 2 3 4 5 6 7 8 9 10 Results of std::sort and Shell sorting are SAME. Test passed.
3	15 18 15 14 67 54 31 78 66 55 12 -1 -567 0 665 11 1	ИТОГ: -567 -1 0 11 12 14 15 18 31 54 55 66 67 78 665 Results of std::sort and Shell sorting are SAME. Test passed.
4	5 9 10 8 6 7 1	ИТОГ: 6 7 8 9 10 Results of std::sort and Shell sorting are SAME. Test passed.
5	4 0 1 0 1 2	ИТОГ: 0 0 1 1 Results of std::sort and Shell sorting are SAME. Test passed.
6	6 1 6 6 1 5 4 1	ИТОГ: 1 1 4 5 6 6 Results of std::sort and Shell sorting are SAME. Test passed.
7	6 8 9 0 1 2 3 2	ИТОГ: 0 1 2 3 8 9 Results of std::sort and Shell sorting are SAME. Test passed.

Выводы.

Был изучен алгоритм сортировки Шелла и реализован на языке программирования C++ двумя разными способами, отличающимися эффективностью. Тестирование проводилось на примере массива целочисленных чисел.

ИСХОДНЫЙ КОД

```
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;
static int h = 0;   //отвечает за количество замен

template <typename T> void printArr(T* arr, int size, int f, int l){
    for (int i = 0; i < size; i++){
        if (i == f || i == l)
            cout << "\033[34m" << arr[i] << "\033[0m ";  //цветной вывод
элементов, которые меняются местами
        else
            cout << arr[i] << ' ';
    }
    cout << '\n';
}

template <typename T> void insert(T* array, int step, int size){
    for (int i = step; i < size; i++) {
        for (int j = i - step; j >= 0 && array[j] > array[j+step]; j -= step) {
//сортировка вставками с учетом инкременты
            T x = array[j];
            h++;       //количество замен увеличивается
            cout << "   ЗАМЕНА № \033[32m" << h << "\033[0m\n";
            cout << "\033[31m Change " << array[j] << " and " << array[j+step]
<< "  \033[0m\n";
            array[j] = array[j + step];
            array[j + step] = x;
            printArr(array, size, j+step, j); //печать измененного массива
        }
    }
}

template <typename T> void shellSortSedgwick(T* array, int size) {
    int steps[50];  //массив для хранения инкремент(step)
    steps[0] = 1;
    int q = 1;
    while (steps[q - 1] * 3 < size) { //заполняем массив, пока текущая
инкремента хотя бы в 3 раза
                                      // меньше количества элементов в массиве
        if (q % 2 == 0)
            steps[q] = 9 * (1 << q) - 9 * (1 << (q / 2)) + 1;  //1 << q =
pow(2,q)
        else
            steps[q] = 8 * (1 << q) - 6 * (1 << ((q + 1) / 2)) + 1;
        q++;
    }
    q--;
    cout << "The array of increments:\n";
    printArr(steps, q + 1, -1, -1);  //вывод массива инкремент без подсветки
    while (q >= 0) {
        int step = steps[q--];  //извлекаем очередную инкременту
        cout << "       STEP = " << step << '\n';
        insert(array,step,size);
```

```
    }
}

template <typename T> void shell(T* array, int size) {
    for (int step = size/2; step > 0; step /= 2) { //первоначальная
последовательность Шелла
                                            //каждый раз инкремента уменьшается в 2
раза
        cout << "       STEP = " << step << '\n';
        insert(array,step,size);
    }
}

template <typename T> int check(vector<T> vec, T* array, int size){
    for (int i = 0; i != size; i++)
        if (vec[i]!= array[i])
            return 0;
        return 1;
}

template <typename T> void typeSort(T* array, int size){
    cout << "Choose the way of sorting:\n1. Using the formula by Sedgwick. \n2.
Using division by 2.\n";
    int type = 0;
    cin >> type;
    switch(type){
        case 1:
            shellSortSedgwick(array, size); //сортировка по формуле Седжвика
            break;
        case 2:
            shell(array,size);  //стандартная сортировка Шелла
            break;
        default:
            cout << "You need to choose 1 or 2. Try again.\n";
            exit(1);
    }
}

int main() {
    int size = 0;  //размер массива
    int type = 0; //тип ввода и тип сортировки Шелла
    int* array;    //массив указателей на int. Для тестирования можно
использовать другой тип данных
    cout << "Choose the way:\n1. Reading from console.\n2. Reading from file."
<< "\n";
    cin >> type;
    switch(type){
        case 1: {
            cout << "Please enter size of your array:" << "\n";
            cin >> size;
            array = new int[size];
            cout << "Enter elements of array one by one:\n";
            for (int i = 0; i != size; i++) {       //заполнение массива с
консоли
                cin >> array[i];
            }
            break;
        }
        case 2: {
            ifstream file;
            string name;
            cout << "Please enter the directory of file:" << "\n";
            cin >> name;
```

```
            file.open(name);        //открываем файл по введенной директории
            if (!file.is_open()) {
                cout << "Can't open the file!\n";
                exit(1);
            }
            file >> size;
            array = new int[size];
            for (int i = 0; i != size; i++) //заполнение массива
                file >> array[i];
            break;
        }
        default:
            cout << "You need to choose 1 or 2. Try again.\n";
            return 0;
    }
    cout << "YOUR INPUT: ";
    printArr(array,size,-1,-1); //вывод введенного массива

    cout << "TEST USING STD::SORT: ";
    vector<int> vec (array, array+size);
    sort (vec.begin(), vec.end());
    for (int i = 0; i < size; i++) {
        cout << vec[i] << " ";
    }
    cout << '\n';
    typeSort(array, size);
    cout << "\nИТОГ:\n";
    printArr(array, size,-1,-1); //вывод отсортированного массива
    if (check(vec,array,size))
        cout << "Results of std::sort and Shell sorting are SAME. Test passed.";
    else
        cout << "Results of std::sort and Shell sorting are DIFFERENT. Test
failed.";
    delete [] array;    //очищение памяти, занятой массивом
    return 0;
}
```