**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

**отчет**

**по лабораторной работе №4**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: Сортировка**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9304 |  | Селезнёва А.В. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2020

**Цель работы.**

Ознакомиться с алгоритмами сортировки. Реализовать программу на языке программирования С++, сортирующую массив входных данных данных.

**Задание.**

Вариант – 18. Сортировка Шелла.

**Выполнение работы.**

На вход программа получает строку, корректность которой проверяет функции *bool IsCorrect().* Если элемент не является целым числом, то он не будет записан в вектор. Полученный вектор копируем в другой вектор, который в последующем будет отсортирован с помощью std::sort из библиотеки algorithm и сравнен с результатом сортировки Шелла первого вектора (функция *ShellSort()*).

Функция *void ShellSort()* сортирует вектор типа int. Сначала сортируются элементы, стоящие на расстоянии , где – длина вектора. Далее процедура повторяется для . Сортировка осуществляется до тех пор, пока .

Также была реализована функция *void PrintVector()*, которая выводит полученный вектор в поток.

Разработанный программный код находится в приложении А.

**Тестирование.**

Тестирование осуществляется с помощью bash-скрипта ./script. Скрипт запускает программу и в качестве входных аргументов подает строки, прописанные в текстовых файлах, расположенных в папке ./Tests.

Пример вывода некоторых результатов тестирования представлен на рисунке 1:

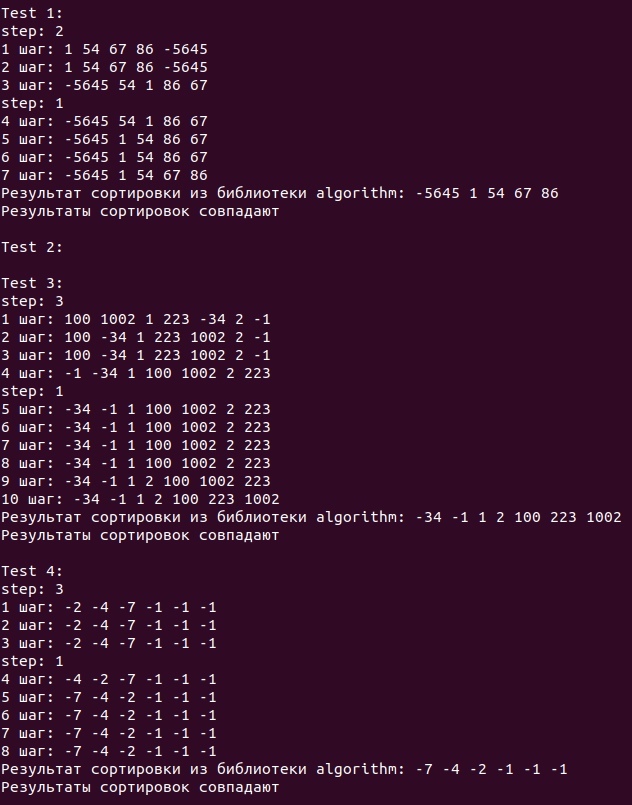


Рисунок 1 – Результаты тестирования

Результаты тестирования представлены в приложении Б.

**Выводы.**

В ходе выполнения лабораторной работы была реализована сортировка Шелла на языке программирования С++.

Разработана программа, сортирующая вектор типа int. Результат работы функции ShellSort сравнивается с std::sort из библиотеки algorithm.

У сортировки Шелла сложность в худшем случае O(), в лучшем – . Данная сортировка подразумевает отсутствие потребности в памяти под стек. Сортировка Шелла уступает в эффективности быстрой сортировке, но выигрывает у сортировки вставками.

**Приложение А**

**Исходный код программы**

Название файла: lab4.cpp

#include <iostream>

#include <vector>

#include <algorithm>

#include <cstdlib>

void PrintVector(const std::vector<int>& v){

for (int x : v){

std::cout << x << " ";

}

std::cout << '\n';

}

void ShellSort (int& n, std::vector<int>& v){

int i, j, step;

int tmp;

int o = 1;

for (step = n / 2; step > 0; step /= 2){

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

tmp = v[i];

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

if (tmp < v[j - step]) {

v[j] = v[j - step];

}

else break;

}

v[j] = tmp;

std::cout << o << " шаг: ";

PrintVector(v);

o++;

}

}

}

bool isCorrect(const std::string str){

size\_t i = 0;

if(str.at(0) == '-' && i != str.size()-1){

++i;

}

while(i < str.size()){

if(isdigit(str.at(i))){

++i;

}

else {

return false;

}

}

return true;

}

int main() {

std::vector<int> v;

std::string x;

int i = 0;

do {

std::cin >> x;

if(isCorrect(x)){

if(x.at(0) == '-'){

std::string x\_1 = x.substr(1,x.size()-1);

v.push\_back(-atoi(x\_1.c\_str()));

}else{

v.push\_back(atoi(x.c\_str()));

}

i++;

}

} while(getchar() != '\n');

std::vector<int> v2 = v;

ShellSort(i, v);

sort(v2.begin(), v2.end());

std::cout << "Результат сортировки из библиотеки algorithm: ";

PrintVector(v2);

if (v == v2){

std::cout << "Результаты сортировок совпадают" << '\n';

} else {

std::cout << "Результаты сортировок не совпадают" << '\n';

}

return 0;

}

**Приложение Б**

**Тестирование**

Таблица Б – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Результат проверки |
| 1. | $ a r 1 ere 54 67 ffghf 86 -5645 y | step: 2  1 шаг: 1 54 67 86 -5645  2 шаг: 1 54 67 86 -5645  3 шаг: -5645 54 1 86 67  step: 1  4 шаг: -5645 54 1 86 67  5 шаг: -5645 1 54 86 67  6 шаг: -5645 1 54 86 67  7 шаг: -5645 1 54 67 86  Результат сортировки из библиотеки algorithm: -5645 1 54 67 86  Результаты сортировок совпадают | correct |
| 2. | asd asafsd f |  | correct |
| 3. | 100 1002 1 223 -34 2 -1 | Test 3:  step: 3  1 шаг: 100 1002 1 223 -34 2 -1  2 шаг: 100 -34 1 223 1002 2 -1  3 шаг: 100 -34 1 223 1002 2 -1  4 шаг: -1 -34 1 100 1002 2 223  step: 1  5 шаг: -34 -1 1 100 1002 2 223  6 шаг: -34 -1 1 100 1002 2 223  7 шаг: -34 -1 1 100 1002 2 223  8 шаг: -34 -1 1 100 1002 2 223  9 шаг: -34 -1 1 2 100 1002 223  10 шаг: -34 -1 1 2 100 223 1002  Результат сортировки из библиотеки algorithm: -34 -1 1 2 100 223 1002  Результаты сортировок совпадают | correct |
| 4. | -2 -4 -7 -1 -1 -1 | Test 4:  step: 3  1 шаг: -2 -4 -7 -1 -1 -1  2 шаг: -2 -4 -7 -1 -1 -1  3 шаг: -2 -4 -7 -1 -1 -1  step: 1  4 шаг: -4 -2 -7 -1 -1 -1  5 шаг: -7 -4 -2 -1 -1 -1  6 шаг: -7 -4 -2 -1 -1 -1  7 шаг: -7 -4 -2 -1 -1 -1  8 шаг: -7 -4 -2 -1 -1 -1  Результат сортировки из библиотеки algorithm: -7 -4 -2 -1 -1 -1  Результаты сортировок совпадают | correct |
| 5. | 1000 999 998 907 57 67 845 45395 394583 | Test 5:  step: 4  1 шаг: 57 999 998 907 1000 67 845 45395 394583  2 шаг: 57 67 998 907 1000 999 845 45395 394583  3 шаг: 57 67 845 907 1000 999 998 45395 394583  4 шаг: 57 67 845 907 1000 999 998 45395 394583  5 шаг: 57 67 845 907 1000 999 998 45395 394583  step: 2  6 шаг: 57 67 845 907 1000 999 998 45395 394583  7 шаг: 57 67 845 907 1000 999 998 45395 394583  8 шаг: 57 67 845 907 1000 999 998 45395 394583  9 шаг: 57 67 845 907 1000 999 998 45395 394583  10 шаг: 57 67 845 907 998 999 1000 45395 394583  11 шаг: 57 67 845 907 998 999 1000 45395 394583  12 шаг: 57 67 845 907 998 999 1000 45395 394583  step: 1  13 шаг: 57 67 845 907 998 999 1000 45395 394583  14 шаг: 57 67 845 907 998 999 1000 45395 394583  15 шаг: 57 67 845 907 998 999 1000 45395 394583  16 шаг: 57 67 845 907 998 999 1000 45395 394583  17 шаг: 57 67 845 907 998 999 1000 45395 394583  18 шаг: 57 67 845 907 998 999 1000 45395 394583  19 шаг: 57 67 845 907 998 999 1000 45395 394583  20 шаг: 57 67 845 907 998 999 1000 45395 394583  Результат сортировки из библиотеки algorithm: 57 67 845 907 998 999 1000 45395 394583  Результаты сортировок совпадают | correct |
| 6. | sad fsd -2 343 sdalkjhlj 46 | step: 1  1 шаг: -2 343 46  2 шаг: -2 46 343  Результат сортировки из библиотеки algorithm: -2 46 343  Результаты сортировок совпадают | correct |