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: Алгоритмы сортировки

Студент гр. 9381 _____ Колованов Р.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться с алгоритмами сортировки, реализовать алгоритм цикличной сортировки на языке C++.

Задание.

Вариант 7.

Цикличная сортировка.

Описание алгоритма.

Достоинства цикличной сортировки: Цикличная сортировка, одна из сортировок выбором, ценна с практической точки зрения тем, что изменения среди элементов массива происходят тогда и только тогда, когда элемент ставится на своё конечное место. Это может пригодиться, если перезапись в массиве — слишком дорогая операция.

Недостатки цикличной сортировки: Алгоритмическая сложность цикличной сортировки остаётся в пределах $O(n^2 / 2)$. На практике цикличная сортировка работает даже в несколько раз медленнее, чем обычная сортировка выбором, так как приходится больше перебирать элементы массива и их сравнивать.

Суть алгоритма заключается в следующем: в начале начинается перебор массива, пусть X - очередная ячейка в этом внешнем цикле. Алгоритм смотрит, на какое место в массиве нужно вставить очередной элемент из буфера обмена, в который записан элемент из ячейки X. На том месте, куда нужно вставить элемент, находится какой-то другой элемент, который после обмена местами отправляется в буфер обмена. Элемент, который был до этого в буфере обмена, отправляется в ячейку массива. Для нового элемента в буфере тоже ищем его место в массиве (и вставляем на это место, а в буфер отправляем элемент, оказавшийся в этом месте). И для нового числа в буфере производим те же действия. Эти действия продолжаются до тех пор, пока очередной элемент в буфере обмена не окажется тем элементом, который нужно вставить именно в

ячейку X (текущее место в массиве во внешнем цикле, который на текущим момент пусто). Рано или поздно этот момент произойдёт, и тогда во внешнем цикле алгоритм перейдет к следующей ячейке и повторит для неё ту же последовательность действий с самого начала. В цикличной сортировке минимальный элемент встает сам на первое место в массиве, в процессе того, как несколько других элементов ставятся на свои места где-то в середине массива.

Описание структур и классов.

Класс Logger.

Класс предоставляет функционал для вывода сообщений в консоль и файл из любой точки программы. Реализован с использованием паттерна *Singleton*. Поля и методы класса приведены в таблицах 4 и 5.

Таблица 4 - Поля класса *Logger*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
private	*int indentSize_*	Хранит размер отступа в пробелах.	*4*
private	*bool silentMode_*	Хранит информацию о том, включен ли тихий режим. При тихом режиме будут печататься сообщения типа COMMON, сообщения типа DEBUG будут игнорироваться.	*false*
private	*bool fileOutput_*	Хранит информацию о том, нужно ли выводить сообщения в файл.	*false*
private	*std::string filePath_;*	Содержит путь к файлу для записи сообщений.	-
private	*std::ofstream file_*	Поток вывода данных в файл.	-

Таблица 5 - Методы класса *Logger*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*Logger&*	*getInstance()*

public	*void*	*log(const std::string& message, MessageType type = COMMON, int indents = 0)*
public	*void*	*getCurrentDataTime()*
public	*void*	*setSilentMode(bool value)*
public	*bool*	*getSilentMode()*
public	*void*	*setFileOutput(const std::string& filePath)*

Метод Logger::getInstance.

Ничего не принимает. Создает статическую переменную объекта класса *Logger* (создается только один раз — при первом вызове данного метода). Возвращает ссылку на созданный объект.

Метод Logger::log.

Принимает на вход три аргумента: *message* — сообщение, *type* — тип сообщения и *indents* — количество отступов. Для начала метод получает единственный объект класса *Logger — logger*. Далее проверяется, если включен тихий режим и тип сообщения — *DEBUG*, то происходит выход из функции. Иначе создает строку отступа, которая состоит из пробелов, количество которых равно *indentSize_ * indents*. Далее функция выводит сообщение с отступом на консоль, а также при наличии флага fileOutput_ — в файл. Ничего не возвращает.

Метод Logger::getCurrentDataTime.

Ничего не принимает. Возвращает текущие дату и время в виде следующей строки: *<день>-<месяц>-<год>_<часы>-<минуты>-<секунды>*. Используется для генерации имени файла с логами.

Метод Logger::setFileOutput.

Принимает на вход *filePath* — путь к файлу для записи сообщений. Присваивает полю *filePath_* значение *filePath*, открывает поток вывода в файл и присваивает значение полю *fileOutput_* значение *true*. Ничего не возвращает.

Метод Logger::setSilentMode.

Принимает на вход *value* — новое значение флага тихого режима. Устанавливает полю *silentMode_* значение *value*. Ничего не возвращает.

Метод Logger::getSilentMode.

Возвращает значение поля *silentMode_*.

Выполнение работы.

Для решения поставленной задачи была написана функция *cycleSort,* выполняющая цикличную сортировку массива чисел. Для вывода основной и промежуточной информации на экран и в файл был использован класс *Logger*. Для вывода массива чисел на экран была написана функция *printArray*. Для вывода справки программы была написана функция *printHelp*. Для удаления некорректных символов, которые не были считаны с потока ввода, используется функция *clearInput*. Помимо этого, был реализован CLI-интерфейс для удобной работы с программой.

Функция cycleSort.

Выполняет цикличную сортировку массива. Принимает на вход два аргумента: *array* – указатель на массив и *arraySize* – количество элементов в массиве. Ничего не возвращает.

Функция printArray.

Выводит значения элементов массива на экран с определенной раскраской. Принимает на вход восемь аргументов: *array* – указатель на массив, *size* – количество элементов в массиве, *type* – тип сообщений логгера, *indent* – отступ от начала строки, *cycleStart* – элемент, с которого начался внешний цикл сортировки, *position* – новая позиция для элемента, isEmpty – является ли элемент с индексом *cycleStart* пустым и *index* – текущий проверяемый элемент массива.

В зависимости от переданных параметров *cycleStart*, *position*, *index* элементы массива будут окрашены в соответствующие цвета. В зависимости от isEmpty вместо элемента с индексом *cycleStart* будет выводиться точка (означает, что в ячейке ничего не лежит). Ничего не возвращает.

Функция printHelp.

Выводит информацию о принимаемых программой аргументах на консоль. Ничего не принимает; ничего не возвращает.

Функция clearInput.

Очищает поток ввода *stdin* до первого символа перевода строки. Требуется для удаления некорректных символов, которые не были считаны с потока ввода. Например, когда программа ожидает получить число, а пользователь вводит букву. Ничего не принимает; ничего не возвращает.

Функция main.

Для начала объявляются следующе переменные:

- *isLoopEnabled* — хранит информацию о том, надо ли продолжать выполнение основного цикла программы;
- *isSilentMode* — хранит информацию о тихом режиме;

После у логгера *logger* вызывается метод *setFileOutput* для установки файла для вывода сообщений. Далее происходит проверка аргументов, подаваемых на вход программе, и в зависимости от переданных аргументов инициализируются переменная *isSilentMode* новым значением. Если один из аргументов неверен, то происходит печать информации об этом и завершение программы. После устанавливается тихий режим при помощи метода *setSilentMode*.

Далее происходит вход в основной цикл программы. Для начала считывается выбранное пользователем действие (цифра от 1 до 4). В зависимости от выбранного действия выполняется либо считывание массива данных с

консоли (1), либо считывание массива данных с файла (2), либо случайная генерация массива данных (3), либо выход из программы (завершение основного цикла - 4). После получения массива входных данных (в данной лабораторной работе мы работаем с массивами целых чисел), создается копия данного массива и заносится в объект класса *std::vector*. Далее производятся сортировки: оригинала массива входных данных при помощи тестируемой функции cycleSort и копии массива входных данных при помощи функции сортировки стандартной библиотеки. Далее результат сортировки функции cycleSort сравнивается с результатом сортировки библиотечной функции сортировки. Если они равны, то функция цикличной сортировки отработала правильно, иначе – нет. В конце итерации выводится отсортированный массив входных данных, результат проверки сортировки на корректность, и программа переходит на следующую итерацию.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Были изучены алгоритмы сортировки, реализован алгоритм цикличной сортировки на языке программирования C++.

Разработана функция *cycleSort*, при помощи которой можно отсортировать массив данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <algorithm>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <ctime>
#include <conio.h>
#include "Logger.h"
#include <Windows.h>

void printHelp() {
    std::cout << "List of available options:\n";
    std::cout << "    -s    Enable silent mode.\n";
    std::cout << "    -h    Print help.\n";
    std::cout << "\n";
}

void clearInput() {
    // Удаляем из потока несчитанные символы
    std::cin.clear();
    while (std::cin.get() != '\n');
}

template<typename T>
void printArray(T* array, int size, MessageType type = COMMON, int indent
= 0, int cycleStart = -1, int position = -1, bool isEmpty = false, int index = -
1) {
    Logger::log("", type, indent);
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

    // Выводим элементы массива при помощи логгера
    for (int i = 0; i < size; i++) {
        if (i == position && i == cycleStart) {
            SetConsoleTextAttribute(hConsole, (WORD)((5 << 4) | 3));
            if (isEmpty) {
                Logger::log(".", type);
            } else {
                Logger::log(std::to_string(array[i]), type);
            }
        }
        else if (i == cycleStart) {
            SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 3));
            if (isEmpty) {
                Logger::log(".", type);
            } else {
                Logger::log(std::to_string(array[i]), type);
            }
        }
        else if (i == position) {
            SetConsoleTextAttribute(hConsole, (WORD)((5 << 4) | 7));
            Logger::log(std::to_string(array[i]), type);
        }
        else if (i == index) {
            SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 6));
            Logger::log(std::to_string(array[i]), type);
        }
        else {
```

```
                Logger::log(std::to_string(array[i]), type);
            }
            SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 7));
            Logger::log(" ", type);
        }
        Logger::log("\n", type);
    }

    template<typename T>
    void cycleSort(T* array, int size) {
        Logger::log("\nCycle sorting start.\n");

        for (int cycleStart = 0; cycleStart < size; cycleStart++) {
            int position = cycleStart;              // Индекс текущего
элемента в массиве
            T value = std::move(array[cycleStart]); // Значение текущего
элемента

            Logger::log("\n[Outer loop iteration #" +
std::to_string(cycleStart) + "]\n", DEBUG);
            Logger::log("Finding a place for an array element '" +
std::to_string(value) + "':\n", DEBUG, 1);

            // Находим новую позицию для текущего элемента
            for (int i = cycleStart + 1; i < size; i++) {
                printArray(array, size, DEBUG, 2, cycleStart, position, false,
i);
                if (array[i] < value) {
                    position++;
                }
            }
            printArray(array, size, DEBUG, 2, cycleStart, position, false);
            Logger::log("\n", DEBUG);

            // Если новую позицию занимает другой элеменет
            if (position != cycleStart) {
                // Пропускаем равные по значению элементы
                while (value == array[position]) {
                    Logger::log("Skipping elements with the same value: '" +
std::to_string(array[position]) + "' with index " + std::to_string(position) +
".\n", DEBUG, 1);
                    position++;
                }

                Logger::log("A new position was found for element '" +
std::to_string(value) + "' with index " + std::to_string(cycleStart) + ": index
" + std::to_string(position) + ".\n", DEBUG, 1);
                Logger::log("Placing element '" + std::to_string(value) + "'
instead element '" + std::to_string(array[position]) + "' with index " +
std::to_string(position) + ":\n", DEBUG, 1);

                // Меняем значения текущего элемента и элемента, который стоит
на новой позиции текущего элемента
                std::swap(array[position], value);

                // Ищем новую позицию для нового текущего элемента
                while (position != cycleStart) {
                    printArray(array, size, DEBUG, 2, cycleStart, position,
true);
                    Logger::log("\n", DEBUG);
                    Logger::log("Finding a place for an array element '" +
std::to_string(value) + "':\n", DEBUG, 1);
                    position = cycleStart;
```

```
                    // Находим новую позицию для текущего элемента
                    for (int i = cycleStart + 1; i < size; i++) {
                        printArray(array, size, DEBUG, 2, cycleStart,
position, true, i);
                        if (array[i] < value) {
                            position++;
                        }
                    }
                    printArray(array, size, DEBUG, 2, cycleStart, position,
true);
                    Logger::log("\n", DEBUG);

                    // Пропускаем равные по значению элементы
                    while (value == array[position]) {
                        Logger::log("Skipping elements with the same value: '"
+ std::to_string(array[position]) + "' with index " + std::to_string(position) +
".\n", DEBUG, 1);
                        position++;
                    }

                    Logger::log("A new position was found for element '" +
std::to_string(value) + "': index " + std::to_string(position) + ".\n", DEBUG,
1);
                    if (position != cycleStart) {
                        Logger::log("Placing element '" +
std::to_string(value) + "' instead element '" + std::to_string(array[position])
+ "' with index " + std::to_string(position) + ":\n", DEBUG, 1);
                    }

                    // Меняем значения текущего элемента и элемента, который
стоит на новой позиции текущего элемента
                    std::swap(array[position], value);
                }

                Logger::log("Placing element on index " +
std::to_string(position) + ":\n", DEBUG, 1);
                printArray(array, size, DEBUG, 2, cycleStart, position,
false);
            } else {
                Logger::log("A new position was found for element '" +
std::to_string(value) + "': index " + std::to_string(position) + ".\n", DEBUG,
1);
                array[position] = std::move(value);
                Logger::log("Placing element on index " +
std::to_string(position) + ":\n", DEBUG, 1);
                printArray(array, size, DEBUG, 2, cycleStart, position,
false);
            }

            if (size <= 10 && !Logger::getInstance().getSilentMode()) {
                Logger::log("\nOuter loop iteration #" +
std::to_string(cycleStart) + " is over. Press 'Enter' to move to the next
iteration...\n", DEBUG);
                clearInput();
            }
        }

        Logger::log("\nCycle sorting end.\n");
    }

    int main(int argc, char* argv[]) {
        bool isLoopEnabled = true;
        bool isSilentMode = false;
        Logger& logger = Logger::getInstance();
```

```
        // Настройка файла вывода сообщений логгера
        logger.setFileOutput("logs\\" + Logger::getCurrentDateTime() +
".txt");

        // Обработка аргументов командной строки
        if (argc > 0) {
            for (int i = 1; i < argc; i++) {
                if (strcmp(argv[i], "-s") == 0) {
                    isSilentMode = true;
                } else if (strcmp(argv[i], "-h") == 0) {
                    printHelp();
                    return 0;
                } else {
                    Logger::log("Unknown option: " + std::string(argv[i]) +
"\n");
                    return 0;
                }
            }
        }

        // Установка тихого режима логгера
        logger.setSilentMode(isSilentMode);

        while (isLoopEnabled) {
            HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

            // Считывание выбора действия пользователя
            Logger::log("Available actions:\n\n  1) Read array from console.\n
2) Read array from file.\n  3) Generate a random array.\n");

            if (!isSilentMode) {
                SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 2));
            }
            Logger::log("  4) Enable output of intermediate data.\n");
            SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 7));

            if (isSilentMode) {
                SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 2));
            }
            Logger::log("  5) Disable output of intermediate data.\n");
            SetConsoleTextAttribute(hConsole, (WORD)((0 << 4) | 7));

            Logger::log("  6) Exit.\n\nChoose one of the actions: ");

            int action = -1;
            std::cin >> action;

            while (action < 1 || action > 6) {
                clearInput();
                Logger::log("Incorrect action. Select the action again: ");
                std::cin >> action;
            }

            if (action == 6) {
                isLoopEnabled = false;
                continue;
            }

            if (action == 4) {
                isSilentMode = false;
                logger.setSilentMode(isSilentMode);
                Logger::log("Intermediate data output enabled.\n");
                continue;
```

```
            }
            else if (action == 5) {
                isSilentMode = true;
                logger.setSilentMode(isSilentMode);
                Logger::log("Intermediate data output disabled.\n");
                continue;
            }

            int arraySize = 0;
            int* array(nullptr);

            if (action == 1 || action == 3) {
                // Считывание размера массива
                Logger::log("Enter array size: ");
                std::cin >> arraySize;

                if (arraySize <= 0) {
                    Logger::log("Invalid size!\n");
                    clearInput();
                    continue;
                }

                array = new int[arraySize];

                // Считываем значения массива с консоли
                if (action == 1) {
                    Logger::log("Enter array:\n");

                    for (int i = 0; i < arraySize; i++) {
                        std::cin >> array[i];

                        // Пропускаем некорректные значения
                        if (std::cin.fail()) {
                            clearInput();
                            i--;
                        }
                    }
                }
                // Рандомная генерация значений массива
                else {
                    srand(time(nullptr));
                    for (int i = 0; i < arraySize; i++) {
                        array[i] = rand() % 2001 - 1000; // Генерируем числа
от -1000 до 1000
                    }
                }

                clearInput();
            }
            // Ввод с файла
            else {
                std::ifstream file("input.txt");

                // Если файл не удалось открыть
                if (!file.is_open()) {
                    Logger::log("Cannot open file: input.txt\n");
                    continue;
                }

                // Считываем размер массива из файла
                file >> arraySize;

                if (arraySize <= 0) {
                    Logger::log("Invalid size!\n");
```

```
                    continue;
                }

                array = new int[arraySize];

                // Считываем значения массива из файла
                for (int i = 0; i < arraySize; i++) {
                    file >> array[i];
                }
            }

            // Создаем копию неотсортированного массива
            std::vector<int> arrayCopy(arraySize);
            for (int i = 0; i < arraySize; i++) {
                arrayCopy[i] = array[i];
            }

            // Выводим неотсортированный массив
            Logger::log("\nUnsorted array:\n");
            printArray(array, arraySize);

            // Сортируем массивы
            cycleSort(array, arraySize);
            std::sort(arrayCopy.begin(), arrayCopy.end());

            // Проверяем правильность сортировки
            bool correct = true;
            for (int i = 0; i < arraySize; i++) {
                if (array[i] != arrayCopy[i]) {
                    correct = false;
                    break;
                }
            }

            // Выводим отсортированный массив
            Logger::log("\nSorted array:\n");
            printArray(array, arraySize);

            // Выводим результат тестирования
            if (correct) {
                Logger::log("\nCycle sort algorithm sorted the array
correctly.\n\n");
            } else {
                Logger::log("\nCycle sort algorithm sorted the array
incorrectly.\n\n");
            }

            delete[] array;
        }

        return 0;
    }
```

Название файла: Logger.h

```
#ifndef LOGGER_H
#define LOGGER_H

#include <fstream>


enum MessageType {
```

```
        COMMON,
        DEBUG
    };


    class Logger {
        int indentSize_ = 4;        // Размер отступа
        bool silentMode_ = false;   // Тихий режим
        bool fileOutput_ = false;   // Вывод сообщений в файл
        std::string filePath_;      // Путь к выходному файлу
        std::ofstream file_;        // Дескриптор выходного файла

        Logger() = default;
        Logger(const Logger&) = delete;
        Logger(Logger&&) = delete;
        Logger& operator=(const Logger&) = delete;
        Logger& operator=(Logger&&) = delete;
        ~Logger() = default;

    public:
        static Logger& getInstance();
        static void log(const std::string& message, MessageType type = COMMON,
int indents = 0);
        static std::string getCurrentDateTime();
        void setSilentMode(bool value);
        bool getSilentMode();
        void setFileOutput(const std::string& filePath);
    };


    #endif // LOGGER_H
```

Название файла: Logger.cpp

```
    #include "Logger.h"
    #include <iostream>
    #include <ctime>

    Logger& Logger::getInstance() {
        static Logger instance;
        return instance;
    }

    void Logger::setSilentMode(bool value) {
        silentMode_ = value;
    }

    bool Logger::getSilentMode() {
        return silentMode_;
    }

    void Logger::setFileOutput(const std::string& filePath) {
        file_.close();
        file_.open(filePath);

        // Проверка открытия файла
        if (!file_.is_open()) {
            filePath_ = "";
            fileOutput_ = false;
            Logger::log("Cannot open file: " + filePath + "\n");
            return;
        }
```

```
        filePath_ = filePath;
        fileOutput_ = true;
    }

    void Logger::log(const std::string& message, MessageType type, int
indents) {
        Logger& logger = Logger::getInstance();

        // Если включен тихий режим и сообщение - отладочное, то происходит
выход из функции
        if (logger.silentMode_ && type == DEBUG) {
            return;
        }

        std::string indent(logger.indentSize_ * indents, ' '); // Формирование
отступа

        std::cout << indent << message; // Вывод на консоль
        if (logger.fileOutput_) {
            logger.file_ << indent << message; // Вывод в файл
        }
    }

    std::string Logger::getCurrentDateTime() {
        tm timeinfo;                                        // Структура с
информацией о времени
        char buffer[80] = { '\0' };                         // Буфер для
строки
        time_t timestamp = time(nullptr);                   // Временная метка
        errno_t error = localtime_s(&timeinfo, &timestamp); // Получение
информации о времени

        // Если возникла ошибка при получении информации о времени, то
возвращаем "00-00-00_00-00-00"
        if (error) {
            return "00-00-00_00-00-00";
        }
        else {
            strftime(buffer, sizeof(buffer), "%d-%m-%y_%H-%M-%S", &timeinfo);
        }

        return buffer;
    }
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев на некорректных данных

№ п/п	Входные данные	Выходные данные
1.	Choose one of the actions: 1 Enter array size: -9	Invalid size!
2.	Choose one of the actions: 3 Enter array size: 0	Invalid size!
3.	Choose one of the actions: 1 Enter array size: 5 a b 2 3 c d 9 8 7 Unsorted array: 2 3 9 8 7	Sorted array: 2 3 7 8 9 Cycle sort algorithm sorted the array correctly.
4.	Choose one of the actions: 1 Enter array size: 5 34.5 2 56.3 4 6 Unsorted array: 34 2 56 4 6	Sorted array: 2 4 6 34 56 Cycle sort algorithm sorted the array correctly.
5.	Choose one of the actions: 1 Enter array size: g	Invalid size!
6.	Choose one of the actions: 3 Enter array size: 5.23 Unsorted array: 452 -184 -584 -385 -571	Sorted array: -584 -571 -385 -184 452 Cycle sort algorithm sorted the array correctly.
7.	Choose one of the actions: fgdgfd Incorrect action. Select the action again: 234.2 Incorrect action. Select the action again: 3	Sorted array: -308 Cycle sort algorithm sorted the array correctly.

	Enter array size: 1 Unsorted array: -308	

Таблица Б.2 - Примеры тестовых случаев на корректных данных

№ п/п	Входные данные	Выходные данные
8.	Choose one of the actions: 3 Enter array size: 5 Unsorted array: 894 679 455 261 699	Sorted array: 261 455 679 699 894 Cycle sort algorithm sorted the array correctly.
9.	Choose one of the actions: 1 Enter array size: 10 8 7 3 9 4 1 0 4 3 -5 Unsorted array: 8 7 3 9 4 1 0 4 3 -5	Sorted array: -5 0 1 3 3 4 4 7 8 9 Cycle sort algorithm sorted the array correctly.
10.	Choose one of the actions: 2 Unsorted array: 364 -417 -1000 -427 -157 810 -524 181 387 -589 -16 983 250 -228 11 678 224 -758 -283 -178 -76 -9 680 793 -742 -683 -681 -7 881 489 -28 603 585 -920 499 225 -860 615 890 -741 -687 278 -965 861 -422 978 -907 -487 220 -240	Sorted array: -1000 -965 -920 -907 -860 -758 -742 -741 -687 -683 -681 -589 -524 -487 -427 -422 -417 -283 -240 -228 -178 -157 -76 -28 -16 -9 -7 11 181 220 224 225 250 278 364 387 489 499 585 603 615 678 680 793 810 861 881 890 978 983 Cycle sort algorithm sorted the array correctly.
11.	Choose one of the actions: 3 Enter array size: 1 Unsorted array: 342	Sorted array: 342 Cycle sort algorithm sorted the array correctly.
12.	Choose one of the actions: 3 Enter array size: 100 Unsorted array: -879 -418 -16 -501 -518 513 -355 -805 -751 -777 -501 -98 -833 551 428 399 -480 -851 -659 -287 -696 -326 -33 -245 -342 231 308 387 753 641 237 -390 -181 -627 -231 -757 237 708 -319 -759 861 496 303 -923 -967 -931 -105 60 -812 936 672 -769 -502 20 152 -456 718 615 140 -870 194 77 764 -347 919 -869 384 -196 861 -913 -710 429 453 852 -996 102 -281 -976 353 -899 158 456 -124 -843 -367 -	Sorted array: -996 -976 -967 -953 -931 -923 -913 -899 -879 -874 -870 -869 -851 -843 -833 -812 -805 -800 -777 -769 -766 -759 -757 -751 -710 -696 -659 -627 -518 -502 -501 -501 -480 -456 -418 -397 -390 -367 -355 -347 -342 -326 -319 -287 -281 -245 -236 -231 -196 -181 -124 -105 -98 -95 -33 -16 20 60 77 102 140 152 158 194 209 231 237 237 292 303 308 338 353 384 387 399 428 429 453 456 496 510 513 551 615 618 641 672 708 718 753 764 791 852 861 861 885 919 936 980 Cycle sort algorithm sorted the array correctly.

	953 -236 885 -874 209 338 791 -397 980 292 510 618 -800 -766 -95	