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

ОТЧЕТ

по практической работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Алгоритмы сортировки

Студент гр. 8381 Муковский Д.В.

Преподаватель Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с основными понятиями алгоритмов сортировки на языке программирования C++. Разработать программу, реализующую естественную сортировку слиянием.

Задание.

Вариант №13

Реализовать сортировку массива естественным слиянием с графическим интерфейсом.

Основные теоретические положения.

В случае простого слияния частичная упорядоченность сортируемых данных не дает никакого преимущества. Это объясняется тем, что на каждом проходе сливаются серии фиксированной длины. При естественном слиянии длина серий не ограничивается, а определяется количеством элементов в уже упорядоченных подпоследовательностях, выделяемых на каждом проходе.

Сортировка, при которой всегда сливаются две самые длинные из возможных последовательностей, является естественным слиянием. В данной сортировке объединяются серии максимальной длины.

Шаг 1. Исходный массив разбивается на два вспомогательных *f*1 и *f*2. Распределение происходит следующим образом: поочередно считываются записи a_i исходной последовательности (неупорядоченной) таким образом, что если значения ключей соседних записей удовлетворяют условию $f(a_i)<=f(a_{i+1})$, то они записываются в первый вспомогательный файл *f*1. Как только встречаются $f(a_i)>f(a_{i+1})$, то записи a_{i+1} копируются во второй вспомогательный файл *f*2. Процедура повторяется до тех пор, пока все записи исходной последовательности не будут распределены по файлам.

Шаг 2. Вспомогательные массивы *f*1 и *f*2 сливаются в *f*, при этом серии образуют упорядоченные последовательности.

Шаг 3. Полученный массив *f* вновь обрабатывается, как указано в шагах 1 и 2.

Шаг 4. Повторяя шаги, сливаем упорядоченные серии до тех пор, пока не будет упорядочен целиком весь массив.

Признаками конца сортировки естественным слиянием являются следующие условия:

- количество серий равно 1 (определяется на фазе слияния).
- при однофазной сортировке второй по счету вспомогательный массив после распределения серий остался пустым.

Естественное слияние, у которого после фазы распределения количество серий во вспомогательных файлах отличается друг от друга не более чем на единицу, называется сбалансированным слиянием, в противном случае – несбалансированное слияние.

На рис.1 представлен пример первого шага.

Исходный файл f: **2 3 17 7 8 9 1 4 6 9 2 3 1 18**

	Распределение	***Слияние***
1 проход	*f1*: 2 3 17 ` 1 4 6 9 ` 1 18 *f2*: 7 8 9 ` 2 3	*f*: 2 3 7 8 9 17 1 2 3 4 6 9 1 18
2 проход	*f1*: 2 3 7 8 9 17 ` 1 18 *f2*: 1 2 3 4 6 9 `	*f*: 1 2 2 3 3 4 6 7 8 9 9 17 1 18
3 проход	*f1*: 1 2 2 3 3 4 6 7 8 9 9 17 *f2*: 1 18	*f*: 1 1 2 2 3 3 4 6 7 8 9 9 17 18

Рисунок 1 - Пример первого шага сортировки естественным слиянием.

Выполнение работы.

Для решения задачи был разработан графический интерфейс с помощью QtCreator. В интерфейсе были реализованы: главное окно с кнопками вызова модальных форм, а также первое диалоговое окно, в котором производилась

генерация случайных чисел и второе диалоговое окно, в котором непосредственно сортировался заданный массив.

Для класса *mainwindow* были реализованы следующие приватные методы и слоты, все нижеприведённые слоты, методы и сигналы приведены в исходном коде в приложении А:

- *on_arrayCreation_clicke*d() – создание модального окна для генерации массива из случайных чисел и его вызов
- *array_check*() – проверка введенной строки
- *on_pushButton_clicked*() – создание модального окна, в котором производилась сортировка массива, и его вызов
- *esirvedString*(*QString str*) – слот, который принимал случайно сгенерированный массив в виде строки

Также был реализован сигнал *sendArray*(*QString str*), который позволял отправить строку в другую форму.

Для класса *randomEntryWindow*() был реализован слот *on_pushButton_clicked*(), который по нажатию на кнопку генерировал массив, и сигнал *sendString*(*QString str*) который использовался для отправления полученной строки в главное окно.

Класс *QuickMergeSort* использовал следующие методы и слоты:

- *recieveArray*(*QString inputArray*) – слот для получения и сохранения массива в виде строки
- *on_buttonQMergeSort_clicked*() – запуск сортировки
- *on_pushButton_clicked*() – запуск одного шага сортировки
- *on_save_clicked*() – сохранение результата в текстовый файл
- *check_path*(*QString path*) – проверка введенного пользователем пути к файлу
- *bool NaturalMergingSort*(…) – реализация сортировки

- *splitIntoTwo*(…) – реализация этапа разбиения массива на два вспогательных
- *Merging*(…) – реализация слияния массивов

Реализация алгоритма сортировки.

Алгоритм сортировки был реализован следующим образом: на каждом этапе массив разбивался на два вспомогательных двумерных массива (*firstPart*, *secondPart*), по принципу, изложенному в теоретическом положении. Каждый одномерный массив в вспомогательном соответствовал упорядоченной подпоследовательности(серии), размер одномерного массива хранился в его первом элементе. В случае если два вспомогательных массива были созданы, они сливались в один одномерный массив и алгоритм продолжал работу с ним, если был создан только первый массив алгоритм завершал свое выполнение.

Алгоритм работы.

Массив подавался из главного окна в модальное, в котором пользователь мог выбрать способ выполнения сортировки: обычный или пошаговый. В случае обычного в метод *NaturalMergingSort* атрибут *flagStep* подавался со значением *false*, иначе *true*. От этого зависело выполнится ли алгоритм полностью или только один его шаг. Результат программы записывался в строку result по ходу выполнения программы, в пошаговом режиме данная строка перезаписывалась в *QTextBrowser* с каждой новой итерацией. После завершения сортировки пользователь может записать результат в текстовый файл, путь к нему он может выбрать сам.

Оценка эффективности алгоритма

Время выполнения естественной сортировки слиянием зависит от того насколько хорошо входные данные были частично отсортированы. Поэтому в наилучшем случае скорость выполнения программы будет приближаться к O(n).

В наихудшем случае скорость выполнения будет стремиться к $O(n^2)$, это обусловлено тем что в отличии от классической сортировки слиянием, которая делит массив пополам и сортирует каждую полученную часть отдельно, естественная делит его на вспомогательные массивы, состоящие из упорядоченных серий, и в случае плохой упорядоченности данного массива количество итераций алгоритма будет стремится к n. В среднем случае скорость сортировки оценивается как O(n*log(n)).

Анализируя данную сортировку можно прийти к выводу, что в ситуациях, когда мы знаем, что массив достаточно хорошо упорядочен скорость выполнения программы действительно очень мало.

Сортировка естественным слиянием также требует дополнительную память по размеру входного массива, то есть O(n).

Ниже на рис.2 приведен график зависимости количества итераций от количества элементов массива. Данные для графика были получены в ходе тестирования программы.

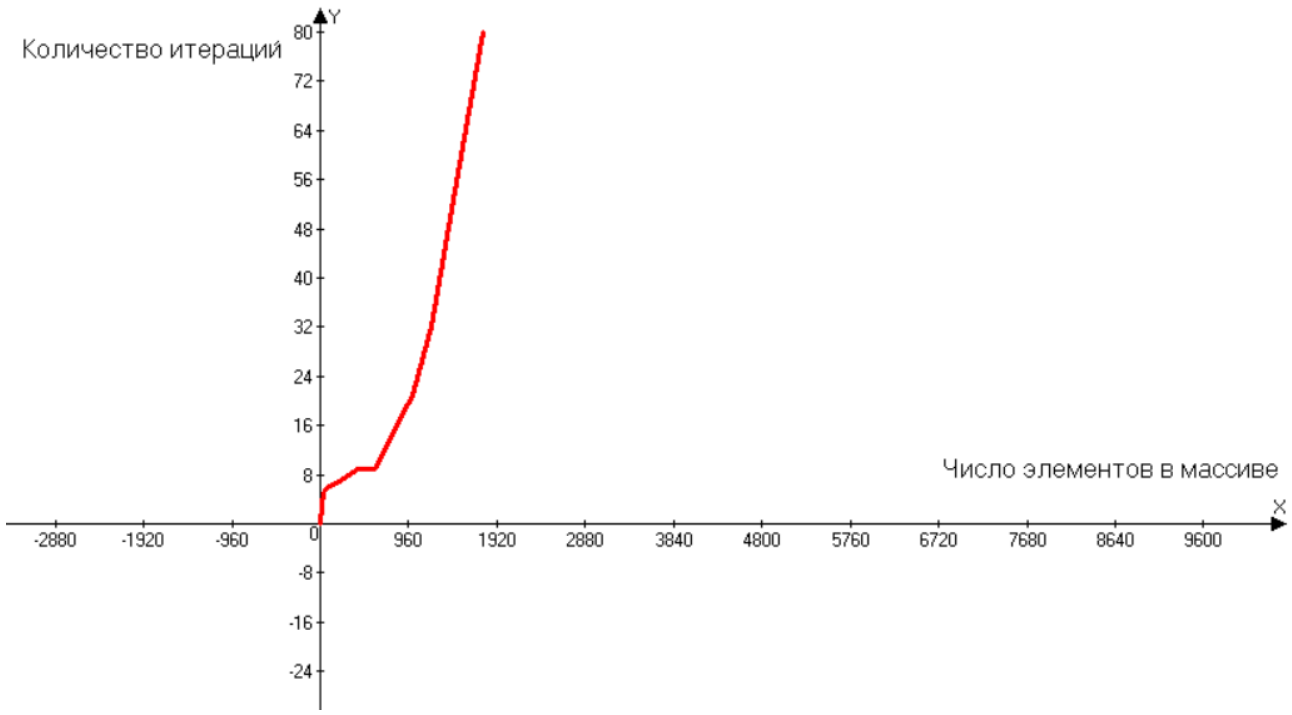

Рисунок 2 – график зависимости числа итераций от количества элементов

Тестирование программы.

Набор тестовых данных		Предполагаемые результаты, высчитанные вручную	Результаты выполнения программы
№	Данные		
1	971 331 447 835 324	324 331 447 835 971	324 331 447 835 971
2	2 1 -2 2 8 1 1 1 1 1 2	-2 1 1 1 1 1 1 2 2 2 8	-2 1 1 1 1 1 1 2 2 2 8
3	6 5 1 9 4 4	1 4 4 5 6 9	1 4 4 5 6 9
4	1 0 0 0 1	0 0 0 1 1	0 0 0 1 1
5	444 986 80 608 610 862 902 393 219 351	80 219 351 393 444 608 610 862 902 986	80 219 351 393 444 608 610 862 902 986

Выводы.

В ходе выполнения лабораторной работы был изучен такой алгоритм сортировки как естественная сортировка слиянием. Была реализована программа на C++, которая сортирует заданный пользователем массив алгоритмом естественного слияния в пошаговом и обычном режиме, а также реализован графический интерфейс.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    this->setWindowTitle("Сортировка естественным слиянием");
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::resirvedString(QString str){
    ui->Array->setText(str);
}

void MainWindow::on_arrayCreation_clicked()
{
    random = new randomEntryWindow();
    connect(random,SIGNAL(sendString(QString)),this,SLOT(resirvedString(QString)));
    random->exec();
}

bool MainWindow::array_check(QString inputArray){
    QRegExp reg( "^-?\\d{1,6} *$|^(-?\\d{1,6} )+-?\\d{1,6} *$" );
    return reg.exactMatch(inputArray);
}

void MainWindow::on_pushButton_clicked()
{
    QString inputArray = ui->Array->text();
    if (inputArray.isEmpty()){
        QMessageBox::critical(this,"Ошибка","Вы не задали массив");
        return;
    }
    if (!array_check((inputArray))){
        QMessageBox::critical(this,"Ошибка","Массив задан некорректно ");
        return;
    }
    else{
        while(inputArray[inputArray.size()-1]==' '){
            inputArray.remove(inputArray.size()-1,inputArray.size()-1);
        }
        qmergeForm = new QuickMergeSort();
        connect(this, SIGNAL(sendArray(QString)), qmergeForm, SLOT(recieveAr-
ray(QString)));
        emit sendArray(inputArray);
        qmergeForm->exec();
    }
```

Название файла main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    bool flag = false;
    for(int i = 1; i < argc; i++)
    {
        if(!strcmp("-console", argv[i]))
            flag = true;
    }
    if(flag){
        consoleMain();
        return 0;
    }
    else
    {
        QApplication a(argc, argv);
        MainWindow w;
        w.show();
        return a.exec();
    }
    return 0;
}
```

Название файла randomentrywindow.cpp

```
#include "randomentrywindow.h"
#include "ui_randomentrywindow.h"

randomEntryWindow::randomEntryWindow(QWidget *parent):
    QDialog(parent),
    ui(new Ui::randomEntryWindow)
{
    ui->setupUi(this);
    this->setWindowTitle("Рандомная генерация");
    setAttribute(Qt::WA_DeleteOnClose);
}

randomEntryWindow::~randomEntryWindow()
{
    delete ui;
}
void randomEntryWindow::on_pushButton_clicked()
{
    srand(time(0));
    int n = 101;
    QString str_n = ui->number->text();
    n = str_n.toInt();
    if (n>100||n<1){
        QMessageBox::critical(this,"Ошибка","Неккоректно заданно число элементов, вве-
дите заново");
        ui->number->text().clear();
        return;
    }
```

```
    QVector <int> inputArray;
    QString stringArray;
    int sign = -1;
    for (int i;i<n;i++){
        sign = rand()%5;
        if (sign == 2){
            inputArray.push_back(-(rand()%100+1));
        }
        else{
            inputArray.push_back(rand()%100+1);
        }
        stringArray += QString::number(inputArray[i])+" ";
    }
    stringArray.remove(stringArray.size()-1,1);
    emit sendString(stringArray);
    close();
}
```

Название файла quickmergesort.cpp

```
#include "quickmergesort.h"
#include "ui_quickmergesort.h"
#include <QFile>
#include <QMessageBox>
#define MAX_SIZE 100
#define MAX_GROUP_SIZE 100

QuickMergeSort::QuickMergeSort(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::QuickMergeSort)
{
    ui->setupUi(this);
    this->setWindowTitle("Сортировка естественным слиянием");
    ui->comboBox->addItem("Сразу результат");
    ui->comboBox->addItem("Пошагово");
    ui->pushButton->hide();
    ui->save->hide();
    ui->lineSave->hide();
    ui->labelSave->hide();
    setAttribute(Qt::WA_DeleteOnClose);
}

QuickMergeSort::~QuickMergeSort()
{
    delete ui;
    delete []inputArray;
}


void QuickMergeSort::freeFirstSecond(int** firstPart,int** secondPart) {
    for (int i = 0; i < MAX_GROUP_SIZE; i++) {
        delete[] firstPart[i];
        delete[] secondPart[i];
    }
    delete[] firstPart;
    delete[] secondPart;
}
```

```
void QuickMergeSort::Merging(int* inputArray, unsigned int size, int** firstPart, int**
secondPart) {
    unsigned int firstLen = firstPart[0][0];
    unsigned int secondLen = secondPart[0][0];
    unsigned int len = 0;
    unsigned int fir = 1, sec = 1;
    bool flag = false;
    if (firstLen > secondLen) {
        len = secondLen;
        flag = true;
    }
    else len = firstLen;

    unsigned int i = 0,j=0,k=0,p=0;
    for (i = 1; i < len; i++) {
        for (j = p; j < (firstPart[i][0] + secondPart[i][0]-2)+p; j++) {
            if (fir != firstPart[i][0] && sec != secondPart[i][0] && firstPart[i][fir]
<= secondPart[i][sec]) {
                inputArray[j] = firstPart[i][fir++];
                continue;
            }
            if (fir != firstPart[i][0] && sec != secondPart[i][0] && firstPart[i][fir]
>= secondPart[i][sec]) {
                inputArray[j] = secondPart[i][sec++];
                continue;
            }
            if (fir == firstPart[i][0] && sec != secondPart[i][0]) {
                inputArray[j] = secondPart[i][sec++];
                continue;
            }
            if (fir != firstPart[i][0] && sec == secondPart[i][0]) {
                inputArray[j] = firstPart[i][fir++];
                continue;
            }
        }
        p = j;
        sec = 1;
        fir = 1;
    }
    fir = 1;
    if (flag)
        for (i = j; i < firstPart[len][0]; i++)
            inputArray[i++] = firstPart[len][fir++];
}


void QuickMergeSort::splitIntoTwo(int* inputArray, int** firstPart, int** secondPart,
unsigned int size) {
    unsigned int i = 0;
    unsigned int fir = 1;
    unsigned int sec = 1;
    unsigned int fir_arr = 1;
    unsigned int sec_arr = 1;
    bool flag = true;
    while (i < size) {
        if (i == 0 || (inputArray[i - 1] <= inputArray[i] && flag)) {
            firstPart[fir_arr][fir++] = inputArray[i++];
        }
        else if (inputArray[i - 1] >= inputArray[i]) {
```

```
            if (flag) {
                secondPart[sec_arr][sec++] = inputArray[i++];
                firstPart[fir_arr++][0] = fir;
                fir = 1;
                flag = false;
            }
            else {
                firstPart[fir_arr][fir++] = inputArray[i++];
                secondPart[sec_arr++][0] = sec;
                sec = 1;
                flag = true;
            }
        }
        else if (inputArray[i - 1] <= inputArray[i] && !flag) {
            secondPart[sec_arr][sec++] = inputArray[i++];
        }
    }
    flag == true ? firstPart[fir_arr][0] = fir : secondPart[sec_arr][0] = sec;
    flag == true ? fir_arr++ : sec_arr++;
    firstPart[0][0] = fir_arr;
    secondPart[0][0] = sec_arr;
}


bool QuickMergeSort::NaturalMergingSort(int* inputArray,unsigned int size, QString& re-
sult, bool flagStep,int step) {
    unsigned int i = 0;
    //int step =1;
    for (;;) {
        int** firstPart = new int* [MAX_GROUP_SIZE];
        int** secondPart = new int* [MAX_GROUP_SIZE];
        for (i = 0; i < MAX_GROUP_SIZE; i++) {
            firstPart[i] = new int[MAX_SIZE];
            secondPart[i] = new int[MAX_SIZE];
        }
        firstPart[0][0] = 0;
        secondPart[0][0] = 0;
        splitIntoTwo(inputArray, firstPart, secondPart, size);
        if (secondPart[0][0] == 1) {
            freeFirstSecond(firstPart,secondPart);
            return true;
        }
        result+= "Step "+QString::number(step++)+"\nfirst: ";

        for (int i = 1; i < firstPart[0][0]; i++) {
            for (int j = 1; j < firstPart[i][0]; j++) {
                result+=QString::number(firstPart[i][j])+' ';
            }
            result+="| ";
        }
        result.remove(result.size()-2,result.size()-1);
        result+="\nsecond: ";
        for (int i = 1; i < secondPart[0][0]; i++) {
            for (int j = 1; j < secondPart[i][0]; j++) {
                result+=QString::number(secondPart[i][j])+' ';
            }
            result+="| ";
        }
        result.remove(result.size()-2,result.size()-1);
```

```
        Merging(inputArray, size, firstPart, secondPart);
        result+="\nAfter merging: ";
        for (unsigned int k = 0; k < size; k++) {
            result+=QString::number(inputArray[k])+' ';
        }
        result+="\n\n";
        freeFirstSecond(firstPart, secondPart);
        if (flagStep) return false;
    }
}
void QuickMergeSort::recieveArray(QString inputArray_){
    ui->input->setText(inputArray_);
    inputArrayString = inputArray_;
}

void QuickMergeSort::on_buttonQMergeSort_clicked(){
    ui->resultText->clear();
    ui->save->hide();
    ui->lineSave->hide();
    ui->labelSave->hide();
    result.clear();
    step =1;
    lst = inputArrayString.split(" ");
    size = lst.size();
    inputArray = new int[size];
    unsigned int i =0;
    for (unsigned int i=0;i<size;i++){
        inputArray[i] = lst[i].toInt();
    }
    int value = ui->comboBox->currentIndex();
    if (!value){
            NaturalMergingSort(inputArray,size,result,false,step);
            result+= "\nRESULT: ";
            for(i=0;i<size;i++){
                result+=QString::number(inputArray[i])+' ';
            }
            ui->resultText->setText(result);
            ui->lineSave->show();
            ui->labelSave->show();
            ui->save->show();
       }
    else {
        ui->comboBox->hide();
        ui->buttonQMergeSort->hide();
        ui->pushButton->show();
    }
}

void QuickMergeSort::on_pushButton_clicked()
{
    bool flagStep = true;
    if (NaturalMergingSort(inputArray,size,result,flagStep,step)){
        result+= "\nRESULT: ";
        for(unsigned int i=0;i<size;i++){
            result+=QString::number(inputArray[i])+' ';
        }
        ui->resultText->setText(result);
        ui->comboBox->show();
        ui->buttonQMergeSort->show();
```

```
        ui->pushButton->hide();
        ui->save->show();
        ui->lineSave->show();
        ui->labelSave->show();
    }
    else{
        step++;
        ui->resultText->setText(result);
    }
}

bool QuickMergeSort::check_path(QString path){
    QRegExp reg("^((C|D):\\\\([a-zA-Z]+\\\\)+[a-zA-Z0-9_]+.txt)|[a-zA-z0-9_]+.txt$");
    return reg.exactMatch(path);
}


void QuickMergeSort::on_save_clicked()
{
    if(ui->lineSave->text().isEmpty()){
        QFile out("C:\\Users\\miair\\Desktop\\output.txt");
        if (out.open(QIODevice::WriteOnly)){
            out.write(result.toUtf8());
        }
        out.close();
        close();
    }
    else {
        QFile realOut(ui->lineSave->text());

        if (realOut.open(QIODevice::WriteOnly)&& check_path(ui->lineSave->text())){
            realOut.write(result.toUtf8());
            realOut.close();
            close();
        }
        else {
            QMessageBox::critical(this,"Ошибка","Путь к файлу указан неверно");
            ui->lineSave->clear();
        }
    }
      }
```

Название файла console.cpp

```
#include "console.h"


using namespace std;
#define MAX_SIZE 6000
#define MAX_GROUP_SIZE 2000

void freeFirstSecond(int** firstPart,int** secondPart) {
    for (int i = 0; i < MAX_GROUP_SIZE; i++) {
        delete[] firstPart[i];
        delete[] secondPart[i];
    }
    delete[] firstPart;
    delete[] secondPart;
}
```

```
void Merging(int* inputArray, unsigned int size, int** firstPart, int** secondPart) {
    unsigned int firstLen = firstPart[0][0];
    unsigned int secondLen = secondPart[0][0];
    unsigned int len = 0;
    unsigned int fir = 1, sec = 1;
    bool flag = false;
    if (firstLen > secondLen) {
        len = secondLen;
        flag = true;
    }
    else {
        len = firstLen;
    }
    unsigned int i = 0,j=0,k=0,p=0;
    for (i = 1; i < len; i++) {
        for (j = p; j < (firstPart[i][0] + secondPart[i][0]-2)+p; j++) {
            if (fir != firstPart[i][0] && sec != secondPart[i][0] && firstPart[i][fir]
<= secondPart[i][sec]) {
                inputArray[j] = firstPart[i][fir++];
                continue;
            }
            if (fir != firstPart[i][0] && sec != secondPart[i][0] && firstPart[i][fir]
>= secondPart[i][sec]) {
                inputArray[j] = secondPart[i][sec++];
                continue;
            }
            if (fir == firstPart[i][0] && sec != secondPart[i][0]) {
                inputArray[j] = secondPart[i][sec++];
                continue;
            }
            if (fir != firstPart[i][0] && sec == secondPart[i][0]) {
                inputArray[j] = firstPart[i][fir++];
                continue;
            }

        }
        p = j;
        sec = 1;
        fir = 1;
    }
    fir = 1;
    if (flag)
        for (i = j; i < firstPart[len][0]; i++)
            inputArray[i++] = firstPart[len][fir++];
}


void splitIntoTwo(int* inputArray, int** firstPart, int** secondPart, unsigned int
size) {
    unsigned int i = 0;
    unsigned int fir = 1;
    unsigned int sec = 1;
    unsigned int fir_arr = 1;
    unsigned int sec_arr = 1;
    bool flag = true;
    while (i < size) {
        if (i == 0 || (inputArray[i - 1] <= inputArray[i] && flag)) {
            firstPart[fir_arr][fir++] = inputArray[i++];
```

```
        }
        else if (inputArray[i - 1] >= inputArray[i]) {
            if (flag) {
                secondPart[sec_arr][sec++] = inputArray[i++];
                firstPart[fir_arr++][0] = fir;
                fir = 1;
                flag = false;
            }
            else {
                firstPart[fir_arr][fir++] = inputArray[i++];
                secondPart[sec_arr++][0] = sec;
                sec = 1;
                flag = true;
            }
        }
        else if (inputArray[i - 1] <= inputArray[i] && !flag) {
            secondPart[sec_arr][sec++] = inputArray[i++];
        }
    }
    flag == true ? firstPart[fir_arr][0] = fir : secondPart[sec_arr][0] = sec;
    flag == true ? fir_arr++ : sec_arr++;
    firstPart[0][0] = fir_arr;
    secondPart[0][0] = sec_arr;
}


void NaturalMergingSort(int* inputArray,unsigned int size,int& step) {
    unsigned int i = 0;
    //int step =1;
    for (;;) {
        int** firstPart = new int* [MAX_GROUP_SIZE];
        int** secondPart = new int* [MAX_GROUP_SIZE];
        for (i = 0; i < MAX_GROUP_SIZE; i++) {
            firstPart[i] = new int[MAX_SIZE];
            secondPart[i] = new int[MAX_SIZE];
        }
        firstPart[0][0] = 0;
        secondPart[0][0] = 0;
        splitIntoTwo(inputArray, firstPart, secondPart, size);
        if (secondPart[0][0] == 1) {
            freeFirstSecond(firstPart,secondPart);
            break;
        }
        step++;
        //cout<<"Step "<<step++<<"\nfirst: ";
        for (int i = 1; i < firstPart[0][0]; i++) {
            for (int j = 1; j < firstPart[i][0]; j++) {
               // cout<<firstPart[i][j]<<' ';
            }
           // cout <<"| ";
        }
        //cout<<"\nsecond: ";
        for (int i = 1; i < secondPart[0][0]; i++) {
            for (int j = 1; j < secondPart[i][0]; j++) {
              //  cout<<secondPart[i][j]<<' ';
            }
          //  cout <<"| ";
        }
        Merging(inputArray, size, firstPart, secondPart);
```

```
       // cout<<"\nAfter merging: ";
        for (int k = 0; k < size; k++) {
             //cout<<inputArray[k]<<' ';
        }
      //  cout<<"\n\n";
        freeFirstSecond(firstPart, secondPart);
    }
}

int consoleMain(){
     unsigned int n;
     srand(time(0));
     //n = rand () % 20 +20;
     n = 100;
     cout << "Numb of elements: "<<n<<"\n";
     int* inputArray = new int[n];
     if (n<=0) return 0;
    int step = 0;
     unsigned int i=0;
     for (i=0;i<n;i++){
         inputArray[i] = rand () % 10000 +1;
         //inputArray[i] = j--;
        // std::cout << inputArray[i] << " ";
     }
     //inputArray[i++] = 4;
     //inputArray[i] = 3;
    // cout<<"\n";
     NaturalMergingSort(inputArray,n,step);
    // cout<<"\nRESULT: ";
     for (i=0;i<n;i++){
         cout<<inputArray[i]<<' ';
     }
     cout<<'\n'<<step;
     cout<<"\n";
     return 0;
}
```