ПМИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Очереди и стеки

Студент гр. 8381 _____ Гоголев Е.Е.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с основными характеристиками и особенностями типов данных стек и очередь, изучить особенности их реализации на языке программирования C++. Разработать программу, использующую иерархические списки и их рекурсивную обработку, высчитывающую значение выражения.

Задание.

Быстрая сортировка, рекурсивная реализация. Процедура трёхчастного разделения. Деление производится не на две группы, а на три: <x, >x, =x.

Основные теоретические положения.

Сложность алгоритма быстрой сортировки с процедурой трёхчастного разделения (quick sort 3-way) в худшем случае такая же, как и у обычной быстрой сортировки — $O(n^2)$. Преимущество данной сортировки в том, что при малой частоте вариативности элементов сокращается количество итераций относительно быстрой сортировки с делением пополам.

Общий принцип работы алгоритма — массив делится на три части, в центральной находятся все одинаковые элементы, а слева и справа — соответственно элементы меньше и больше центральных. Процедура повторяется для левой и правой частей, пока их размер не станет равен 1.

Выполнение работы.

Написание работы производилось на базе операционной системы Windows 10 в среде QTCreator.

Происходит считывание введенных пользователем данных и проверка на режим работы программы. При выполнении в консоли (указан аргумент «-c») запускается считывание аргументов командной строки и сама сортировка с выводом результатов. При выполнении с графически запускается окно

mainwindow, в котором даётся выбор между автоматической генерацией случайного массива и ручным вводом в поле Input.

Вызвать функцию сортировки можно нажатием кнопки «Sort».

Функция сортировки была реализована с системой логгирования. В центральном окне визуализируется процесс работы алгоритма.

После завершения работы программы результат выводится пользователю в поле Output. В поле Log элементы массива выделяются снизу, показывая, над каким интервалом массива идёт работа.

Оценка эффективности алгоритма.

В наиболее сбалансированном варианте при каждой операции разделения массив делится на две одинаковые (плюс-минус один элемент) части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1, составит и сложность алгоритма будет равна $O(n \times \log_2 n)$.

В самом несбалансированном варианте каждое разделение даёт два подмассива размерами 1 и n − 1, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых. Общее время работы составит $O(n^2)$.

Тестирование программы.

Консольный режим:

Таблица 1 — результаты работы консольного режима программы

```
03:07:28: Запускается D:\gogolev_lr3.exe...
Input array:
8 3 4 5 6
```

```
Executing algorithm...
---LOG---
3 4 5 6 8
[-----------]   l:0 r:4 center: 3 depth: 0
3 4 5 6 8
   [-----^--]   l:1 r:4 center: 6 depth: 1
3 4 5 6 8
   [--]         l:1 r:2 center: 4 depth: 2
---END---
Result:
3 4 5 6 8
```

Графический интерфейс:

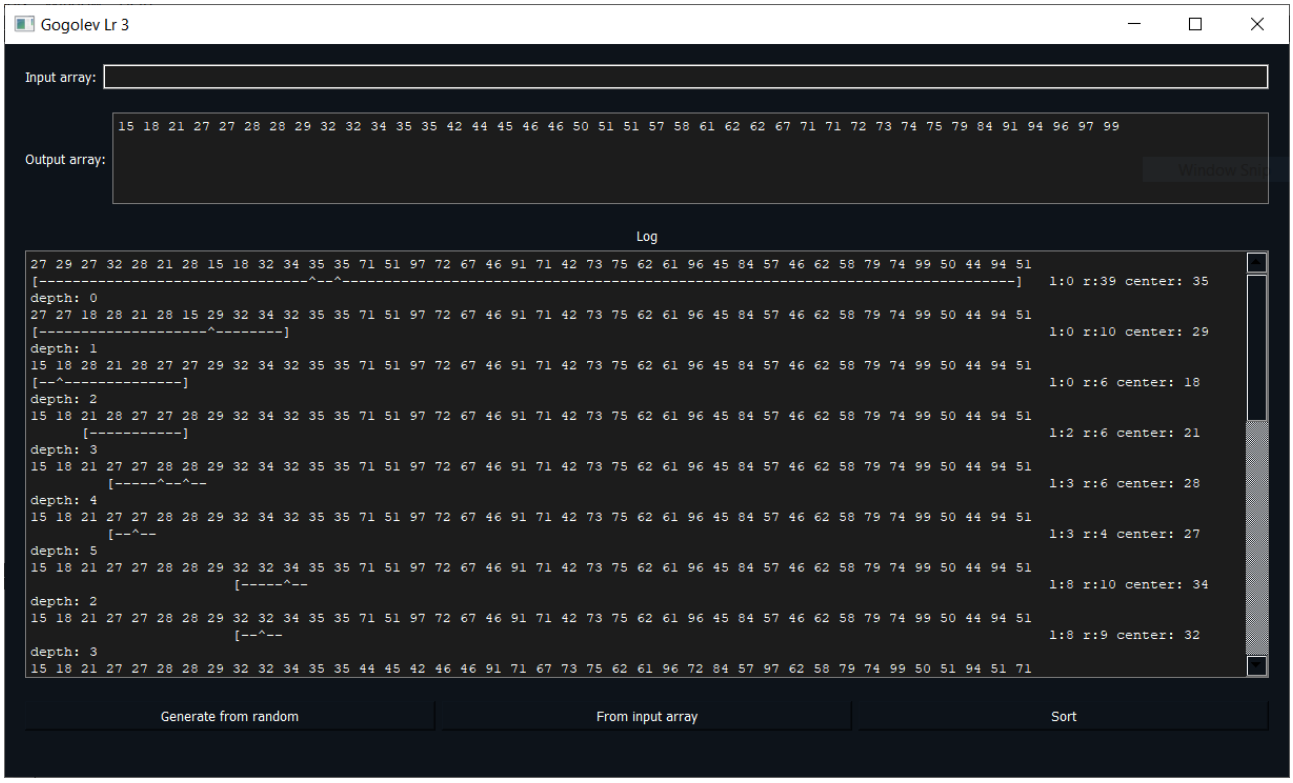

Рисунок 1 — графический интерфейс программы

Выводы.

В ходе выполнения лабораторной работы была написана программа, сортирующая массив целочисленных элементов. Был реализован алгоритм быстрой сортировки Quick sort с трёхчастным делением, имеющий худшую сложность $O(n^2)$, среднюю сложность $O(nlog(n))$ и лучшую сложность $O(n)$.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include "mainwindow.h"
#include <QApplication>
#include <QDebug>
#include <iostream>
#include "quicksort3.h"
#include "array_list.h"

using namespace std;

int console_main(int argc, char *argv[]);

int main(int argc, char *argv[])
{
    if (argc > 1 && !strncmp(argv[1], "-c", 2))
    {
        return console_main(argc - 2, argv + 2);
    }
    else
    {
        QApplication a(argc, argv);
        MainWindow w;
        w.show();
        return a.exec();
    }
}

std::string print_list(array_list& list)
{
    string s;
    for (int i = 0; i < list.size(); i++)
    {
        s += std::to_string(list[i]) + " ";
    }
    return s;
}

int console_main(int argc, char *argv[])
{
    array_list list = array_list();
    cout << "Input array: " << endl;
```

```
    for (int i = 0; i < argc; i++)
    {
        list.push_back(atoi(argv[i]));
    }
    cout << print_list(list);
    cout << endl << "Executing algorithm..." << endl;
    string s;
    qsort3way(s, list, 0, list.size() - 1);
     cout << "---LOG---" << endl << s << endl << "---END---" <<
endl;
      cout << endl << "Result: " << endl << print_list(list) <<
endl;
    return 0;
}
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "array_list.h"
#include "quicksort3.h"
#include <ctime>
#include <cstdlib>
#include <string>
#include <QMessageBox>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_Generate_clicked();

    void on_Input_clicked();
```

```
    void on_Sort_clicked();

private:
    Ui::MainWindow *ui;
    array_list list;
    void visualize();
};

#endif // MAINWINDOW_H
```

Файл array_list.h:

```
#ifndef ARRAY_LIST_H
#define ARRAY_LIST_H

struct array_list
{
    int* array;
    int capacity;
    int count;

    void resize(int new_capacity);
    array_list(int start_capacity = 4);
    int& operator[] (int index);
    void clean();
    void insert(int index, int element);
    int remove(int index);

    int back();
    void push_back(int element);
    int pop_back();

    int front();
    void push_front(int element);
    int pop_front();

    int size();
    bool empty();
};

#endif // ARRAY_LIST_H


#endif // ARRAY_LIST_H
```

Файл quicksort3.h:

```
#ifndef QUICKSORT3_H
#define QUICKSORT3_H
```

```
#include "array_list.h"
#include <string>

using namespace std;

void qsort3way(string& s, array_list& list, int l, int r, int
depth = 0);
string log(array_list& list, int min, int max, int center, int
depth);

#endif // QUICKSORT3_H
```

Файл quicksort3.cpp:

```
#include "quicksort3.h"
void qsort3way(string& s, array_list& list, int l, int r, int
depth)
{
    if (l >= r)
    {
        return;
    }
    int lt = l; // We initiate lt to be the part that is less than
the pivot
    int gt = r;  //The part that is greater than the pivot
     int pivot = list[l + (rand() % (r - l))];//The pivot, chosen
to be the random element of the array
    int i = l; //We scan the array from left to right
    while (i <= gt) // Starting from the first element.
    {
        if (list[i] < pivot)
        {
            int t = list[lt];
            list[lt] = list[i];
            list[i] = t;
            lt += 1;
            i += 1;
        }
        else if (list[i] > pivot)
        {
            int t = list[gt];
            list[gt] = list[i];
            list[i] = t;
            gt -= 1;
        }
        else
```

```
        {
            i += 1;
        }
    }
    s += log(list, l, r, pivot, depth);
    qsort3way(s, list, l, lt - 1, depth + 1);
    qsort3way(s, list, gt + 1, r, depth + 1);
}

string log(array_list &list, int min, int max, int center, int
depth)
{
    string s = "";
    for (int i = 0; i < list.size(); i++)
    {
        s += to_string(list[i]) + ' ';
    }
    s += '\n';
    for (int i = 0; i < list.size(); i++)
    {
        if (i == min) s += "[--";
        else if (list[i] == center) s += "^--";
        else if (i > min && i < max) s += "---";
        else if (i == max) s += "]  ";
        else s += "   ";
    }
      s += " l:" + to_string(min) + " r:" + to_string(max) + "
center: " + to_string(center) + " depth: " + to_string(depth) +'\
n';
    return s;
}
```

Файл array_list.cpp:

```
#ifndef UTILS_VECTOR_H
#define UTILS_VECTOR_H
#include "array_list.h"

void array_list::resize(int new_capacity)
{
    int *arr = new int[count];
    for (int i = 0; i < count; ++i)
    {
        arr[i] = array[i];
    }
    delete [] array;
```

```
    array = new int[new_capacity];
    for (int i = 0 ; i < count; ++i)
    {
        array[i] = arr[i];
    }
    delete [] arr;
    capacity = new_capacity;
}

array_list::array_list(int start_capacity)
{
    capacity = start_capacity;
    count = 0;
    array = new int[capacity];
}

int& array_list::operator[] (int index)
{
    return array[index];
}

void array_list::clean ()
{
    count = 0;
}

void array_list::insert(int index, int element)
{
    if (capacity == count)
    {
        resize(count + 8);
    }
    if (count > 0) {
        for (int i = count; i > index; i--)
        {
            array[i] = array[i - 1];
        }
    }
    count++;
    array[index] = element;
}

int array_list::remove(int index)
{
    auto temp = array[index];
    for (int i = index; i < count - 1; i++)
```

```
    {
        array[i] = array[i + 1];
    }
    count--;
    return temp;
}

int array_list::back()
{
    return array[count - 1];
}

void array_list::push_back(int element)
{
    if (capacity == count)
    {
        resize(count + 8);
    }
    array[count] = element;
    count++;
}

int array_list::pop_back()
{
    return array[--count];
}

int array_list::front()
{
    return *array;
}

void array_list::push_front(int element)
{
    insert(0, element);
}

int array_list::pop_front()
{
    return remove(0);
}

int array_list::size()
{
    return count;
}
```

```
bool array_list::empty()
{
    return !count;
}

#endif //VECTOR_VECTOR_H
```

Файл mainwindow.cpp:

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

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    list = array_list();
}

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

void MainWindow::on_Generate_clicked()
{
    list.clean();
    for (int i = 0; i < 40; i++)
    {
        list.push_back(rand() % 90 + 10);
    }
    visualize();
}

void MainWindow::on_Input_clicked()
{
    QString input = ui->InputEdit->text();
    if (input.isEmpty()) return;
    list.clean();
    for (QString i : ui->InputEdit->text().split(' '))
    {
        bool ok;
        int a = i.toInt(&ok);
        if (!ok)
        {
```

```
                QMessageBox::warning(this, "Warning", "Input is not
number");
            ui->InputEdit->setText("");
            return;
        }
        list.push_back(a);
    }
    visualize();
}

void MainWindow::on_Sort_clicked()
{
    string res;
    qsort3way(res, list, 0, list.size() - 1);
    ui->Log->setText(QString::fromStdString(res));
    visualize();
}

void MainWindow::visualize()
{
    QString s = "";
    for (int i = 0; i < list.size(); i++)
    {
        s += QString::number(list[i]) + ' ';
    }
    ui->CurrentArray->setText(s);
}
```