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 _____ Сахаров В.М.

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.

14. Нитевидная сортировка.

Основные теоретические положения.

Сложность алгоритма нитевидной сортировки (Strand sort) - в среднем $O(n^2)$. Однако, данный вид сортировки эффективен при работе с почти упорядоченными списками - O(n).

Общий принцип работы алгоритма — в несколько итераций создаются упорядоченные подсписки исходного списка, после чего происходит их слияние без нарушения порядка.

Порядок работы алгоритма:

- При последовательном прохождении исходного массива строится упорядоченный список, с учётом одной итерации и того, что вставлять элементы можно только в конец подсписка
- Полученный подсписок сливается с третьим, итоговым списком, путём последовательных сравнений элементов и вставки из подсписка на подходящие места
- Данные две процедуры повторяются, пока в исходном массиве остаются элементы
- В конце алгоритма в третьем массиве будут находится отсортированные элементы

Выполнение работы.

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

Сначала происходило 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 ручным вводом и автоматической генерацией случайного неотсортированного массива. При введении пользователем символов, которые нельзя представить в виду целого числа, данный набор символов считается нулём.

Нажатиями на кнопки «Array List»и «Linked List» имеется возможность переключить алгоритмы программы на работу со списком, реализованном на базе массива, и на работу со списком, реализованном на узлах (двусвязным списком) соответственно.

Вызвать функцию сортировки можно двумя способами — кнопка «Step» отвечает за пошаговое выполнение сортировки, а функцией «Sort» можно выполнить сортировку до конца, даже если она уже была начата пошагово.

Функция сортировки была реализована двумя способами — для полной сортировки был реализован алгоритм, основанный на циклах, а для пошагового выполнения он был переделан в машину состояний с сохранением в классе state текущего состояния и позиции в массиве.

После завершения работы программы результат выводится пользователю в поле Output. Кроме того, результат представлен в виде последовательной работы алгоритма. Элементы массива выделяются цветами для наглядности и быстрой проверки успешности работы алгоритма сортировки.

Оценка эффективности алгоритма.

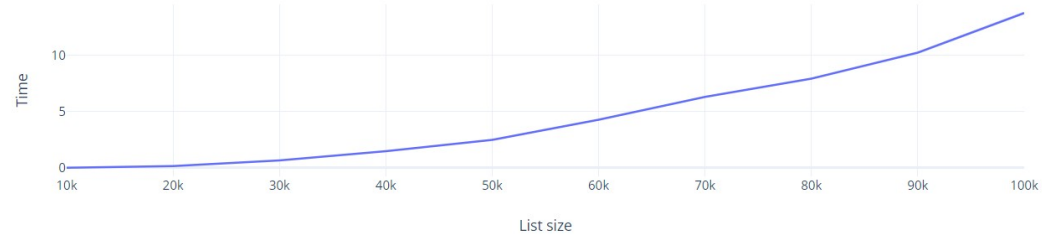
В лучшем случае массив уже отсортирован, тогда сложность алгоритма — O(n), т. к. происходит n итераций по исходному массиву, причём все элементы сразу попадают в конечный массив. При частично отсортированном массиве сложность также приближается к O(n), т. к. основная масса элементов отсортируется при первых итерациях.

В худшем варианте массив отсортирован и инвертирован. Тогда при каждой итерации подсписок будет содержать один элемент, тогда сложность будет равна $O(n^2)$

Для частично отсортированного массива

Для массива случайных чисел

Для инвертированного массива



Тестирование программы.

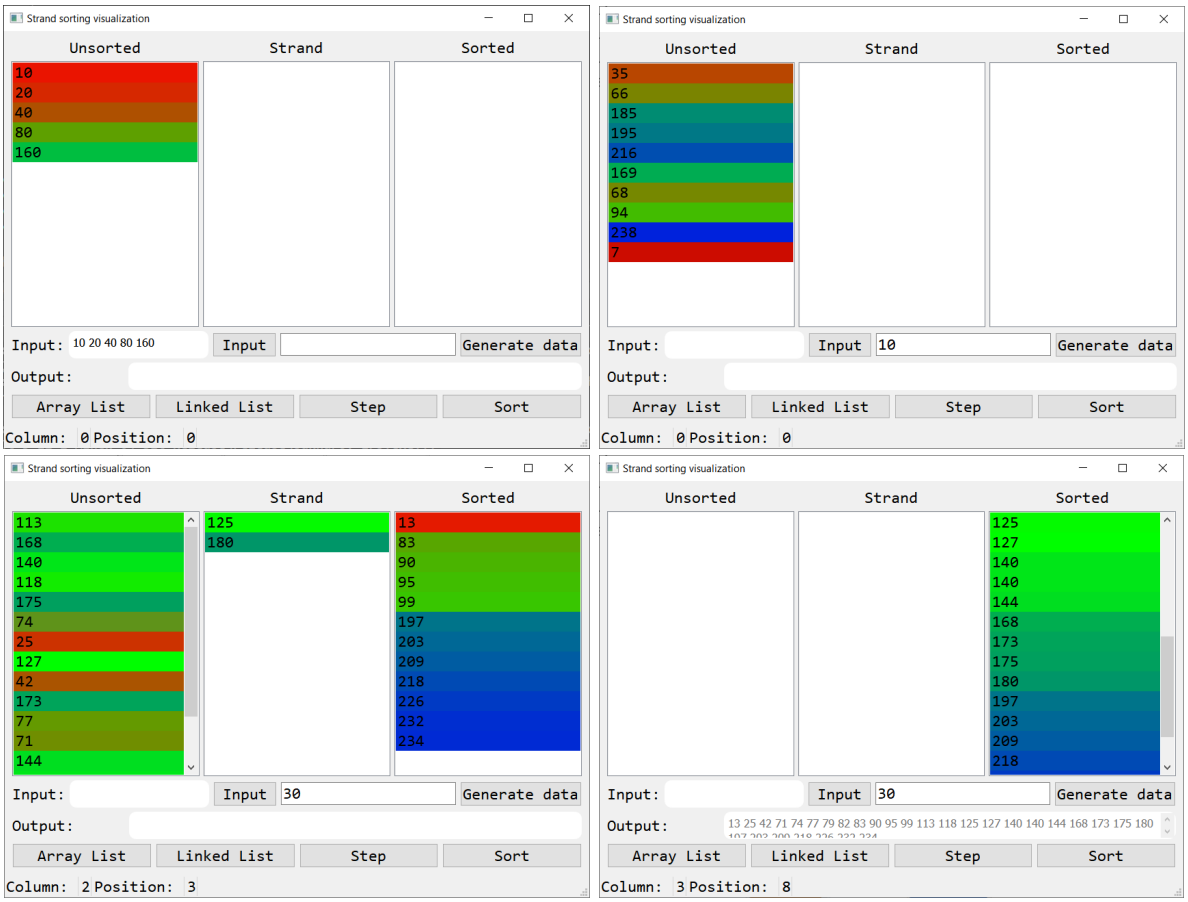
Консольный режим:

```
03:07:28: Запускается D:\sakharov_lr3.exe...
```

```
Executing input...
Input got:
L1: 34 234 54 756 6
L2:
L3:
Executing algorithm...
Result:
L1:
L2:
L3: 6 34 54 234 756
03:07:31:sakharov_lr3.exe завершился с кодом 0
```

Графический интерфейс:

Представлены скриншоты ручного ввода элементов (1), генерации случайного массива из десяти элементов (2), Процесса пошаговой сортировки (3) и результата сортировки (4):



Выводы.

В ходе выполнения лабораторной работы была написана программа, сортирующая массива целочисленных элементов. Был реализован алгоритм нитевидной сортировки Strand sort, имеющий худшую сложность $O(n^2)$ и лучшую сложность $O(n)$.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

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

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

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QProgressBar>
#include <QLabel>
#include <QString>
#include <QListWidgetItem>
#include <ctime>
#include "model.h"
#include "ilist.h"

#define M_CHECK if (mainmodel->busy) return; \
                mainmodel->busy = true;

#define M_UNLOCK mainmodel->busy = false;

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
```

```
    void on_GenerateButton_clicked();

    void on_ProceedButon_clicked();

    void on_LinkedListButton_clicked();

    void on_ArrayListButton_clicked();

    void on_AutoButton_clicked();

    void on_SortButton_clicked();

private:
    void syncState();
    void syncListView(QListWidget* view, IList<int>& list);

    logic_model* mainmodel;
    Ui::MainWindow *ui;

    QLabel *statusLabel;
    QLabel *statusLabel2;
    QProgressBar *statusProgressBar;


};

#endif // MAINWINDOW_H
```

Файл ilist.h:

```
#ifndef ILIST_H
#define ILIST_H

template <class T>
struct IList {
    virtual T operator[] (int index) = 0;
    virtual T at (int index) = 0;
    virtual void clean() = 0;
    virtual void insert(int index, T element) = 0;
    virtual T remove(int index) = 0;

    virtual T back() = 0;
    virtual void push_back(T element) = 0;
    virtual T pop_back() = 0;

    virtual T front() = 0;
    virtual void push_front(T element) = 0;
    virtual T pop_front() = 0;

    virtual int size() = 0;
    virtual bool empty() = 0;
    virtual ~IList(){}
};

#endif // ILIST_H
```

Файл utils_linked.h:

```
#ifndef UTILS_LINKED_H
#define UTILS_LINKED_H
#include "ilist.h"

template <class T>
struct node
```

```
{
    node* next;
    T data;
    node* prev;

    node(T d)
    {
        next = nullptr;
        data = d;
        prev = nullptr;
    }
};

template <class T>
class utils_linked : public IList<T>
{
private:
    node<T>* head;
    node<T>* tail;

public:
    utils_linked();
    utils_linked(const utils_linked& copy);
    T operator[] (int index) override;
    T at (int index) override;
    void clean() override;
    void insert(int index, T element) override;
    T remove(int index) override;

    T back() override;
    void push_back(T element) override;
    T pop_back() override;

    T front() override;
    void push_front(T element) override;
    T pop_front() override;

    int size() override;
    bool empty() override;
    ~utils_linked();
};


template<class T>
utils_linked<T>::utils_linked()
{
    head = nullptr;
    tail = nullptr;
}

template<class T>
utils_linked<T>::utils_linked(const utils_linked & copy)
{
//    head = nullptr;
//    tail = nullptr;
//    node<int>* copy_node = copy.head;
//    while (copy_node)
//    {
//        node<int>* t = new node<int>(copy_node->data);
//        head->prev = tail;
//        tail->next = head;
//        t->data
```

```
//    }
//    array = new T[capacity];
//    for (int i = 0; i < count; ++i)
//    {
//        *(array + i) = *(copy.__arr + i);
//    }
}


template <class T>
T utils_linked<T>::operator[] (int index)
{
    node<T>* t = head;
    for (int i = 0; i < index; i++)
    {
        t = t->next;
    }
    return t->data;
}

template <class T>
T utils_linked<T>::at (int index)
{
    return operator[](index);
}

template <class T>
void utils_linked<T>::clean ()
{
    node<T>* t = head;
    while (t)
    {
        delete t;
        t = t->next;
    }
    head = nullptr;
    tail = nullptr;
}

template <class T>
void utils_linked<T>::insert(int index, T element)
{
    node<T>* n = new node<T>(element);
    if (empty())
    {
        head = n;
        tail = n;
    }
    else if (index == 0)
    {
        push_front(element);
    }
    else if (index == size())
    {
        push_back(element);
    }
    else
    {
        node<T>* t = head;
        for (int i = 0; i < index; i++)
        {
            t = t->next;
```

```
        }
        n->next = t;
        n->prev = t->prev;
        t->prev->next= n;
        t->prev = n;
    }
}

template<class T>
T utils_linked<T>::remove(int index)
{
    T res = at(index);
    if (index == 0)
    {
        pop_front();
    }
    else if (index == size() - 1)
    {
        pop_back();
    }
    else {
        node<T> *t = head;
        for (int i = 0; i < index; i++) {
            t = t->next;
        }
        t->prev->next = t->next;
        t->next->prev = t->prev;
        delete t;
    }
    return res;
}

template<class T>
T utils_linked<T>::back()
{
    return tail->data;
}

template<class T>
void utils_linked<T>::push_back(T element)
{
    node<T>* n = new node<T>(element);
    if (empty())
    {
        head = n;
        tail = n;
    }
    else
    {
        tail->next = n;
        n->prev = tail;
        tail = n;
    }
}

template<class T>
T utils_linked<T>::pop_back()
{
    T data;
    if (size() == 1) {
        if (head != nullptr) {
            data = head->data;
```

```
            delete head;
            head = nullptr;
        } else if (tail != nullptr) {
            data = tail->data;
            delete tail;
            tail == nullptr;
        }
    }
    else {
        node<T> *n = tail;
        tail = tail->prev;
        tail->next = nullptr;
        data = n->data;
        delete n;
    }
    return data;
}

template<class T>
T utils_linked<T>::front()
{
    return head->data;
}

template<class T>
void utils_linked<T>::push_front(T element)
{
    node<T>* n = new node<T>(element);
    if (empty())
    {
        head = n;
        tail = n;
    }
    else
    {
        head->prev = n;
        n->next = head;
        head = n;
    }
}

template<class T>
T utils_linked<T>::pop_front()
{
    T data;
    if (size() == 1) {
        if (head != nullptr) {
            data = head->data;
            delete head;
            head = nullptr;
        } else if (tail != nullptr) {
            data = tail->data;
            delete tail;
            tail == nullptr;
        }
    }
    else {
        node<T> *n = head;
        head = head->next;
        head->prev = nullptr;
        data = n->data;
        delete n;
```

```
    }
    return data;
}


template<class T>
int utils_linked<T>::size()
{
    int i = 0;
    node<T>* t = head;
    while (t)
    {
        t = t->next;
        i++;
    }
    return i;
}

template<class T>
bool utils_linked<T>::empty()
{
    return !size();
}

template<class T>
utils_linked<T>::~utils_linked()
{
    node<T>* t = head;
    while (t)
    {
        delete t;
        t = t->next;
    }
}

#endif // UTILS_LINKED_H
```

Файл utils_vector.h:

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

template <class T>
class utils_vector : public IList<T>
{
private:
    T* array;
    int capacity;
    int count;
    void resize(int new_capacity);

public:
    utils_vector(int start_capacity = 4);
    utils_vector(const utils_vector& copy);
    T operator[] (int index) override;
    T at (int index) override;
    void clean() override;
    void insert(int index, T element) override;
    T remove(int index) override;

    T back() override;
    void push_back(T element) override;
```

```
    T pop_back() override;

    T front() override;
    void push_front(T element) override;
    T pop_front() override;

    int size() override;
    bool empty() override;
    ~utils_vector();
};

template<class T>
void utils_vector<T>::resize(int new_capacity)
{
    auto *arr = new T[count];
    for (int i = 0; i < count; ++i)
    {
        arr[i] = array[i];
    }
    delete [] array;
    array = new T[new_capacity];
    for (int i = 0 ; i < count; ++i)
    {
        array[i] = arr[i];
    }
    delete [] arr;
    capacity = new_capacity;
}

template<class T>
utils_vector<T>::utils_vector(int start_capacity)
{
    capacity = start_capacity;
    count = 0;
    array = new T[capacity];
}

template<class T>
utils_vector<T>::utils_vector(const utils_vector & copy) :
    count(copy.size),
    capacity(copy.capacity)
{
    array = new T[capacity];
    for (int i = 0; i < count; ++i)
    {
        *(array + i) = *(copy.__arr + i);
    }
}

template <class T>
T utils_vector<T>::operator[] (int index)
{
    return array[index];
}

template <class T>
T utils_vector<T>::at (int index)
{
    return operator[](index);
}

template <class T>
```

```
void utils_vector<T>::clean ()
{
    count = 0;
}

template <class T>
void utils_vector<T>::insert(int index, T 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;
}

template<class T>
T utils_vector<T>::remove(int index)
{
    auto temp = array[index];
    for (int i = index; i < count - 1; i++)
    {
        array[i] = array[i + 1];
    }
    count--;
    return temp;
}

template<class T>
T utils_vector<T>::back()
{
    return array[count - 1];
}

template<class T>
void utils_vector<T>::push_back(T element)
{
    if (capacity == count)
    {
        resize(count + 8);
    }
    array[count] = element;
    count++;
}

template<class T>
T utils_vector<T>::pop_back()
{
    return array[--count];
}

template<class T>
T utils_vector<T>::front()
{
    return *array;
}
```

```
template<class T>
void utils_vector<T>::push_front(T element)
{
    insert(0, element);
}

template<class T>
T utils_vector<T>::pop_front()
{
    return remove(0);
}

template<class T>
int utils_vector<T>::size()
{
    return count;
}

template<class T>
bool utils_vector<T>::empty()
{
    return !count;
}

template<class T>
utils_vector<T>::~utils_vector()
{
    delete [] array;
}

#endif //VECTOR_VECTOR_H
```

Файл utils_strandsort.h:

```
#ifndef UTILS_STRANDSORT_H
#define UTILS_STRANDSORT_H

#include "ilist.h"
#include "state.h"

class utils_strandsort
{
public:
    static void sort (state& s);
    static void step (state& s);
};

#endif // UTILS_STRANDSORT_H
```

Файл utils_cli.h:

```
#ifndef UTILS_CLI_H
#define UTILS_CLI_H

#include <iostream>
#include <cstdlib>
#include "model.h"

using namespace std;

class utils_cli
{
public:
```

```
    static int exec(int argc, char *argv[]);
};

#endif // UTILS_CLI_H
```

Файл state.h:

```
#ifndef STATE_H
#define STATE_H

#include "ilist.h"

struct state
{
    int column;
    int position;
    IList<int> *l1;
    IList<int> *l2;
    IList<int> *l3;

    state ()
    {
        column = 0;
        position = 0;
    }
};

#endif // STATE_H
```

Файл model.h:

```
#ifndef MODEL_H
#define MODEL_H

#include <cstdlib>
#include <ctime>
#include <string>
#include "state.h"
#include "utils_vector.h"
#include "utils_linked.h"
#include "utils_strandsort.h"

class logic_model
{
public:
    logic_model();
    void Clear();
    void SetArr();
    void SetLinked();
    void Generate(int count = 15, int maxnum = 100);
    void Sort();
    void SortStep();
    std::string ToString();

    bool busy;
    state current;
};

#endif // MODEL_H
```

Файл mainwindow.cpp:

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

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    mainmodel(new logic_model()),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    QLabel* textLabel = new QLabel(this);
    textLabel->setText("Column: ");
    statusLabel = new QLabel(this);
    QLabel* textLabel2 = new QLabel(this);
    textLabel2->setText("Position: ");
    statusLabel2 = new QLabel(this);
    //statusProgressBar = new QProgressBar(this);

    //statusProgressBar->setTextVisible(false);
    // addPermanentWidget(label) would add it to the first empty spot from the
bottom right
    // If you're just wanting to show a message, you could use: statusBar()-
>showMessage(tr("Message Here"))
    ui->statusBar->addWidget(textLabel);
    ui->statusBar->addWidget(statusLabel);
    ui->statusBar->addWidget(textLabel2);
    ui->statusBar->addWidget(statusLabel2);
    //ui->statusBar->addWidget(statusProgressBar,1);
}

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

void MainWindow::syncState()
{
    ui->List1->clear();
    ui->List2->clear();
    ui->List3->clear();
    state s = mainmodel->current;
    statusLabel->setText(QString::number(s.column));
    statusLabel2->setText(QString::number(s.position));
    syncListView(ui->List1, *s.l1);
    syncListView(ui->List2, *s.l2);
    syncListView(ui->List3, *s.l3);
    if (s.column == 3)
    {
        QString result = "";
        for (int i = 0; i < s.l3->size(); i++)
        {
            result += QString::number(s.l3->at(i)) + " ";
        }
        ui->Output->setText(result);
    }
    else
    {
        ui->Output->setText("");
    }
    mainmodel->busy = false;
}

void MainWindow::syncListView(QListWidget* view, IList<int>& list)
{
    for (int i = 0; i < list.size(); i++)
```

```
    {
        int v = list[i];
        QListWidgetItem* item = new QListWidgetItem(QString::number(list[i]));
        QColor color;
        int r = v < 128 ? (127 - v) * 2 : 0;
        int g = v < 128 ? v * 2 : (255 - v) * 2;
        int b = v <= 128 ? 0 : (v - 128) * 2;
        color.setRgb(r, g, b);
        item->setBackgroundColor(color);
        view->addItem(item);
    }
}

void MainWindow::on_GenerateButton_clicked()
{
    M_CHECK
    mainmodel->Generate(ui->RandomCount->text().isEmpty() ? 20 : ui->Random-
Count->text().toInt(), 255);
    syncState();
    M_UNLOCK
}

void MainWindow::on_ProceedButon_clicked()
{
    M_CHECK
    if (!ui->Input->toPlainText().isEmpty())
    {
        mainmodel->Clear();
        for (auto i : ui->Input->toPlainText().split(' '))
        {
            mainmodel->current.l1->push_back(i.toInt());
        }
        syncState();
    }
    M_UNLOCK
}

void MainWindow::on_LinkedListButton_clicked()
{
    M_CHECK
    mainmodel->SetLinked();
    syncState();
    M_UNLOCK
}

void MainWindow::on_ArrayListButton_clicked()
{
    M_CHECK
    mainmodel->SetArr();
    syncState();
    M_UNLOCK
}

void MainWindow::on_AutoButton_clicked()
{
    M_CHECK
    mainmodel->SortStep();
    syncState();
    M_UNLOCK
}
```

```
void MainWindow::on_SortButton_clicked()
{
    M_CHECK
    mainmodel->Sort();
    syncState();
    M_UNLOCK
}
```

Файл model.cpp:

```
#include "model.h"

logic_model::logic_model()
{
    std::srand(static_cast<unsigned>(std::time(nullptr)));
    busy = false;
    current = state();
    SetArr();
}

void logic_model::Clear()
{
    current.column = 0;
    current.position = 0;
    current.l1->clean();
    current.l2->clean();
    current.l3->clean();
}

void logic_model::SetArr()
{
    current.l1 = new utils_vector<int>();
    current.l2 = new utils_vector<int>();
    current.l3 = new utils_vector<int>();
    Clear();
}

void logic_model::SetLinked()
{
    current.l1 = new utils_linked<int>();
    current.l2 = new utils_linked<int>();
    current.l3 = new utils_linked<int>();
    Clear();
}

void logic_model::Generate(int count, int maxnum)
{
    Clear();
    for (int i = 0; i < count; i++)
    {
        current.l1->push_back(rand() % maxnum);
    }
}

void logic_model::Sort()
{
    if (current.l2->empty() && current.l3->empty())
    {
        utils_strandsort::sort(current);
    }
    else
    {
```

```
        while (!current.l1->empty() || !current.l2->empty())
utils_strandsort::step(current);
    }
    current.column = 3;
}

void logic_model::SortStep()
{
    utils_strandsort::step(current);
}

std::string logic_model::ToString()
{
    std::string res = "L1: ";
    for (int i = 0; i < current.l1->size(); i++)
    {
        res += std::to_string(current.l1->at(i)) + " ";
    }
    res += "\nL2: ";
    for (int i = 0; i < current.l2->size(); i++)
    {
        res += std::to_string(current.l2->at(i)) + " ";
    }
    res += "\nL3: ";
    for (int i = 0; i < current.l3->size(); i++)
    {
        res += std::to_string(current.l3->at(i)) + " ";
    }
    return res;
}
```

Файл utils_cli.cpp:

```
#include "utils_cli.h"

int utils_cli::exec(int argc, char *argv[])
{
    argc = 5; //STUB
    char* t[] = {"34", "234", "54", "756", "6"}; //STUB
    argv = t;
    logic_model model = logic_model();
    cout << "Executing input..." << endl << "Input got: " << endl;
    for (int i = 0; i < argc; i++)
    {
        model.current.l1->push_back(atoi(argv[i]));
    }
    cout << model.ToString();
    cout << endl << "Executing algorithm..." << endl;
    model.Sort();
    cout << endl << "Result: " << endl << model.ToString();
    return 0;
}
```

Файл utils_strandsort.cpp:

```
#include "utils_strandsort.h"

void utils_strandsort::sort(state& s)
{
    while(!s.l1->empty())
    {
        s.l2->clean();
        s.l2->push_back(s.l1->pop_front());
```

```
        for(int i = 0; i < s.l1->size(); i++)
        {
            if(s.l1->at(i) > s.l2->back())
            {
                s.l2->push_back(s.l1->remove(i));
                i--;
            }
        }
        int j = 0;
        while (!s.l2->empty())
        {
            bool spliced = false;
            for(;j < s.l3->size(); j++)
            {
                if(s.l2->front() < s.l3->at(j))
                {
                    s.l3->insert(j, s.l2->pop_front());
                    j++;
                    spliced = true;
                    break;
                }
            }
            if(!spliced)
            {
                s.l3->push_back(s.l2->pop_front());
            }
        }
    }
}

void utils_strandsort::step(state& s)
{
    switch (s.column) {
    case 0:
        if (s.l1->empty() && s.l2->empty()) return;
        s.l2->clean();
        s.l2->push_back(s.l1->pop_front());
        s.column = 1;
        s.position = 0;
        break;
    case 1:
        if (s.position >= s.l1->size())
        {
            s.column = 2;
            s.position = 0;
            return;
        }
        while (s.position < s.l1->size())
        {
            if (s.l1->at(s.position) > s.l2->back())
            {
                s.l2->push_back(s.l1->remove(s.position));
                break;
            }
            s.position++;
        }
        break;
    case 2:
        if (s.l2->empty())
        {
            s.column = 0;
            s.position = 0;
```

```
            return;
        }
        bool spliced = false;
        for(;s.position < s.l3->size(); s.position++)
        {
            if(s.l2->front() < s.l3->at(s.position))
            {
                s.l3->insert(s.position, s.l2->pop_front());
                s.position++;
                spliced = true;
                break;
            }
        }
        if(!spliced)
        {
            s.l3->push_back(s.l2->pop_front());
        }
        break;
    }

}
```