|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-3 «Управление и моделирование систем»

**ОТЧЕТ   
о выполнении лабораторной работы №1**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Программная реализация нелинейных структур»**

**Вариант № \_\_\_**

Выполнил: студент 2 курса

группы \_\_\_БИСО-03-19\_\_\_\_

\_\_\_Хачатрян Евгений Арутюнович\_\_\_\_\_  
*(фио студента)*

Проверил:

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Москва 2020 г.

**Задание на лабораторную работу № 1.**

В рамках лабораторной работы №1 требуется программно реализовать (с помощью указателей (однонаправленных/двунаправленный динамический линенйый связанный список, массива или используя стандартный контейнер библиотеки STL “stack” или «queue» - по варианту) абстрактный тип данных (АТД) в соответствии с заданием (стек, дек, очередь с одной головой, очередь с головой и хвостом).

Абстрактный тип данных должен позволять осуществлять только операции, присущие типу линейного связанного списка:

* получить значение первого элемента (на выходе),
* добавить элемент (на вход),
* удалить элемент из списка (на выходе),
* проверить – список пуст,
* обнулить (проинициализировать) список (конструктур, при необходимости).
* деструктор (при необходимости)

Используя разработанный АТД и указанный набор операций, необходимо реализовать заданный алгоритм сортировки последовательности элементов заданного типа, при этом следует учитывать, что разрешен доступ (чтение/извлечение) только к элементу на выходе.

На основе исходного текста программы получить аналитическую оценку трудоемкости работы алгоритма сортировки, используя О-символику для каждого реализованного метода АТД и сортировки в целом.

**Вариант № 10\_\_\_.**

**Реализация связи элементов линейного списка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Способ организации линейного связанный список: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Алгоритм сортировки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Теория о сортировках.**

Этот метод сортировки Д. Шелл предложил в 1959 г. Он использует минимум памяти и показывает высокие скорости при сортировке. По сути в методе Шелла применяются сравнения и перестановки элементов аналогичные методы вставок, но при этом порядок сравниваемых элементов совершенно другой.

Идея **сортировки методом Шелла** состоит в том, чтобы сортировать элементы отстоящие друг от друга на некотором расстоянии step. Затем сортировка повторяется при меньших значениях step, и в конце процесс сортировки Шелла завершается при step = 1 (а именно обычной сортировкой вставками).

До сих пор продолжает обсуждаться вопрос выбора шага сортировки step. Шелл предложил такую последовательность: N/2, N/4, N/8 …, где N – количество элементов в сортируемом массиве.

Сортировка Шелла требует около log2N проходов для упорядочивания последовательности длиной N.

**Листинг программы с расчетами.**

#include "stdafx.h"

#include <locale.h>

#include <ctime>

#include "windows.h"

#include <iostream>

using namespace std;

struct Node

{

// указатель на следующий элемент в очереди

Node \*next;

// значение элемента в очереди

int value;

};

class Ochered

{

public:

// Колличество элементов в списке

int N;

int i;

int size = 0;

int size\_1 = NULL;

// Счетчик колличества операций

unsigned long long int N\_op = 0;

// указатель на первый элемент

Node \*head;

// указатель на хвост очереди

Node \*tail;

// инициализация очереди

void Init() //2

{

head = NULL; //1

// При инициализации head и tail = NULL

tail = head; //1

}

// добавление элемента в начало очереди, x - значение этого элемента

void Add(int x) //(3+2+2)+(1+2+1)+2=13

{

// создание нового элемента

Node \*node = new Node; //3

// в поле next данного элемента записывается NULL

node->next = NULL; //2

// в поле value записывается значение ячейки очереди int

node->value = x; //2

// после инициализации и при создании первого элемента мы проходим по ветке else

if (tail != NULL) //1

{

/\* добавление элемента через хвост в очереди,

tail указывает на первый элемент в очереди перед добавляемым

в поле next первого элемента записывается адрес нового первого элемента\*/

tail->next = node; //2

// tail начинает указывать на новый элемент в очереди

tail = node; //1

}

else

{

// хвост указывает на первый элемент в очереди

tail = node; //1

// голова указывает на первый элемент в очереди

head = tail; //1

}

size++; //2

}

// проверка очереди на пустоту

bool Isempty() //1

{

//true, если head = NULL; false, если head указывает на какой-либо элемент в очереди

return head == NULL; //1

}

// удаление элемента из конца очереди

int Del() //1+(2+2+2+1+2+2+2)=14

{

int val = NULL; //1

// если очередь не пуста

if (Isempty() != 1) //2

{

// создаем указатель на структуру

Node \*tmp;

// в указателюь tmp кладем ссылку на next предпоследнего элемента в очереди

tmp = head->next; //2

// в val записываем значение последнего элемента в очереди

val = head->value; //2

// удаляем объект по указателю head (последний элемент в очереди)

delete head;

// head начинает указывать на предпоcледний элемент в очереди (последний удален)

head = tmp; //1

// После удаления последнего элемента из очереди происходит повторная инициализация

if (Isempty() == 1) //2

{

Init(); //2

}

size--; //2

// возвращаем значение последнего элемента в очереди

return val;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

// получение значения из конца очереди

int Value() //3

{

// если очередь не пуста

if (Isempty() != 1) //2

{

// через указатель head получить значение value последнего элемента в очереди

return head->value; //1

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

};

// Класс наследник

class Numbers : public Ochered

{

public:

// получение значения из списка, x - номер элемента в списке

int get(int x) //2+1+1+(1+33n)=5+33n

{

// если очередь не пуста

if (Isempty() != 1) //2

{

size\_1 = size; //1

int val1 = NULL; //1

// Перебор всех значений в списке

for (int i = 1; i < size\_1 + 1; i++) //

{

if (i == x) //1

{

// Когда i доходит до номера получаемого элемента, то в val записывается значение данного элемента

val1 = Value(); //4

}

// Перебор начинается только в случае если x не равен нулю

if (x != 1) //1

{

// Вначале элемент удаляется из конца очереди, и сразу же добавляется в начало очереди, и так в цикле пока не восстановится первоначальное состояние очереди

Add(Del()); //14+13=27

}

else

{

// В противном случае сразу выходим из цикла for

break;

}

}

// Возвращается значение элемента под номером x из очереди

return val1;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

// установка значения элемента в списке, x - номер элемента в списке, y - значение элемента в списке

void set(int x, int y) //2+3+31n=5+31n

{

// если очередь не пуста

if (Isempty() != 1) //2

{

if (x <= size) //1+1+1+31n=3+31n

{

size\_1 = size; //1

for (int i = 1; i < size\_1 + 1; i++) //

{

if (i == x) //1

{

// Запись значения y в список

head->value = y; //2

}

if (x != 1) //1

{

Add(Del()); //14+13=27

}

else

{

break;

}

}

}

// Если элемент устанавливается за пределы изначального списка

else //

{

if (x == size + 1) //2

{

//Если номер элемента в списке на 1 больше размера очереди, значит элемент добавляется в начало очереди

Add(y);

}

else

{

for (i = size + 1; i < x; i++) //

{

// В противном случае очередь заполняется нулями

Add(0); //13

}

Add(y); //13

}

}

}

else { cout << "Очередь пуста!" << endl; }

void sort(int N)

{

int i, j, step;

int tmp;

int n;

for (step = N / 2; step > 0; step /= 2) //

{

for (i = step; i < N; i++)

{

tmp = get(i);

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

{

if (tmp < get(j - step))

{

}

}

}

}

size\_1 = size; N\_op += 2;

for (int i = 1; i <= size\_1; i++)

{

if (i <= N / 2)

{

N\_op += 2;

Add(Del());

}

else

{

N\_op += 2;

Del();

}

}

}

};

}

int main()

{

setlocale(LC\_ALL, "ru");

srand(time(NULL));

// Схема эксперимента

// Инициализация очереди и заполнение хранилища ключей

int i, t\_s, t\_f;

// Хранилище ключей

int Key[3000];

int N = 300;

Numbers list;

list.Init();

for (i = 0; i < 3000; i++)

{

// Заполнение хранилища ключей случайными числами до 1000

Key[i] = rand() % 999;

}

for (i = 0; i < 10; i++)

{

for (int z = N - 300; z < N; z++)

{

list.Add(Key[z]);

}

list.N\_op = 0;

t\_s = GetTickCount();

list.sort(N);

t\_f = GetTickCount();

cout << "Номер сортировки: " << i + 1 << " Колличество отсортированных элементов: " << N << " Время сортировки (ms): " << t\_f - t\_s << " Колличество операций (N\_op): " << list.N\_op << endl; // Шаг в 300 элементов

N = N + 300;

}

return 0;

}

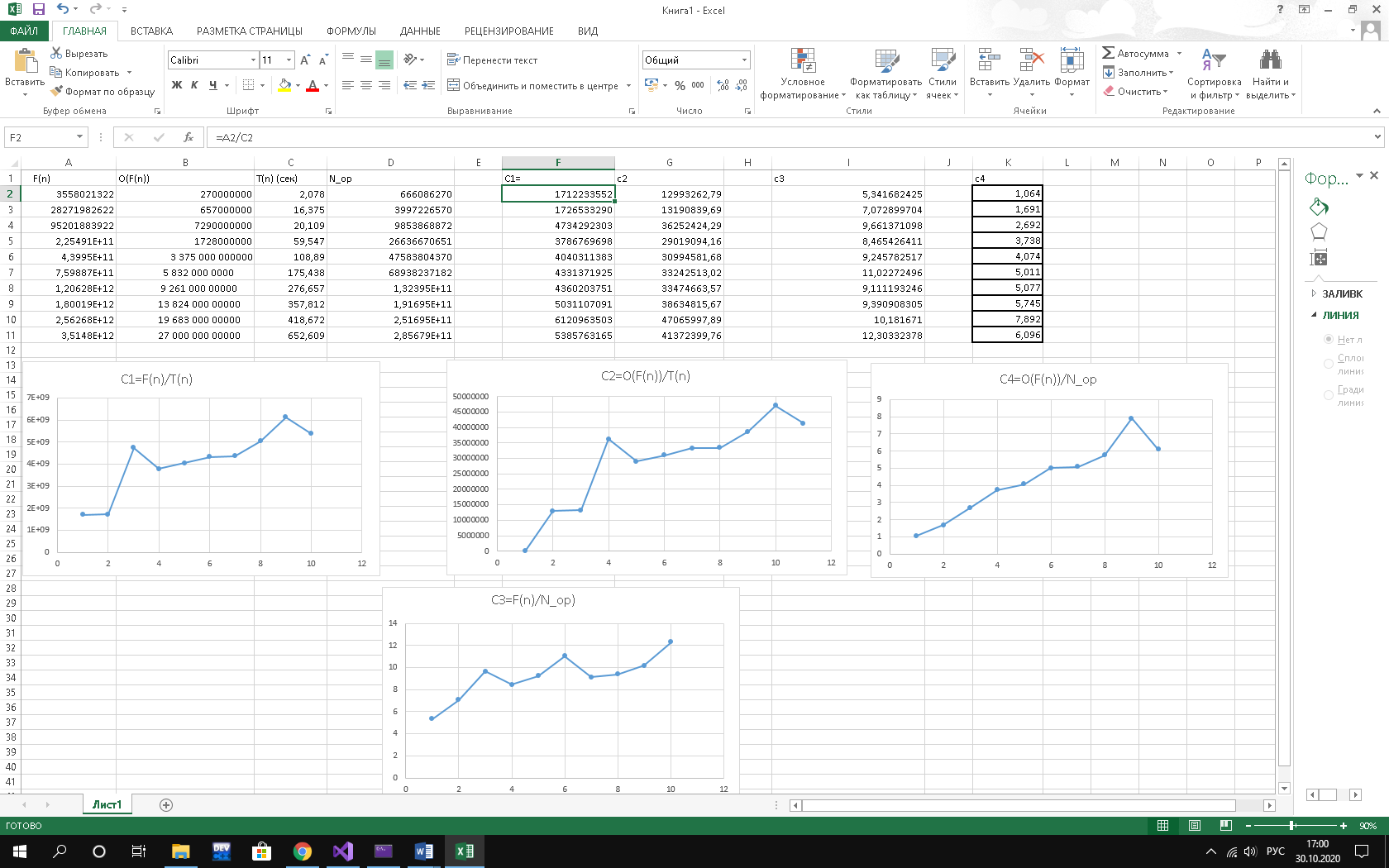
F(n)=1+

O(F(n))=

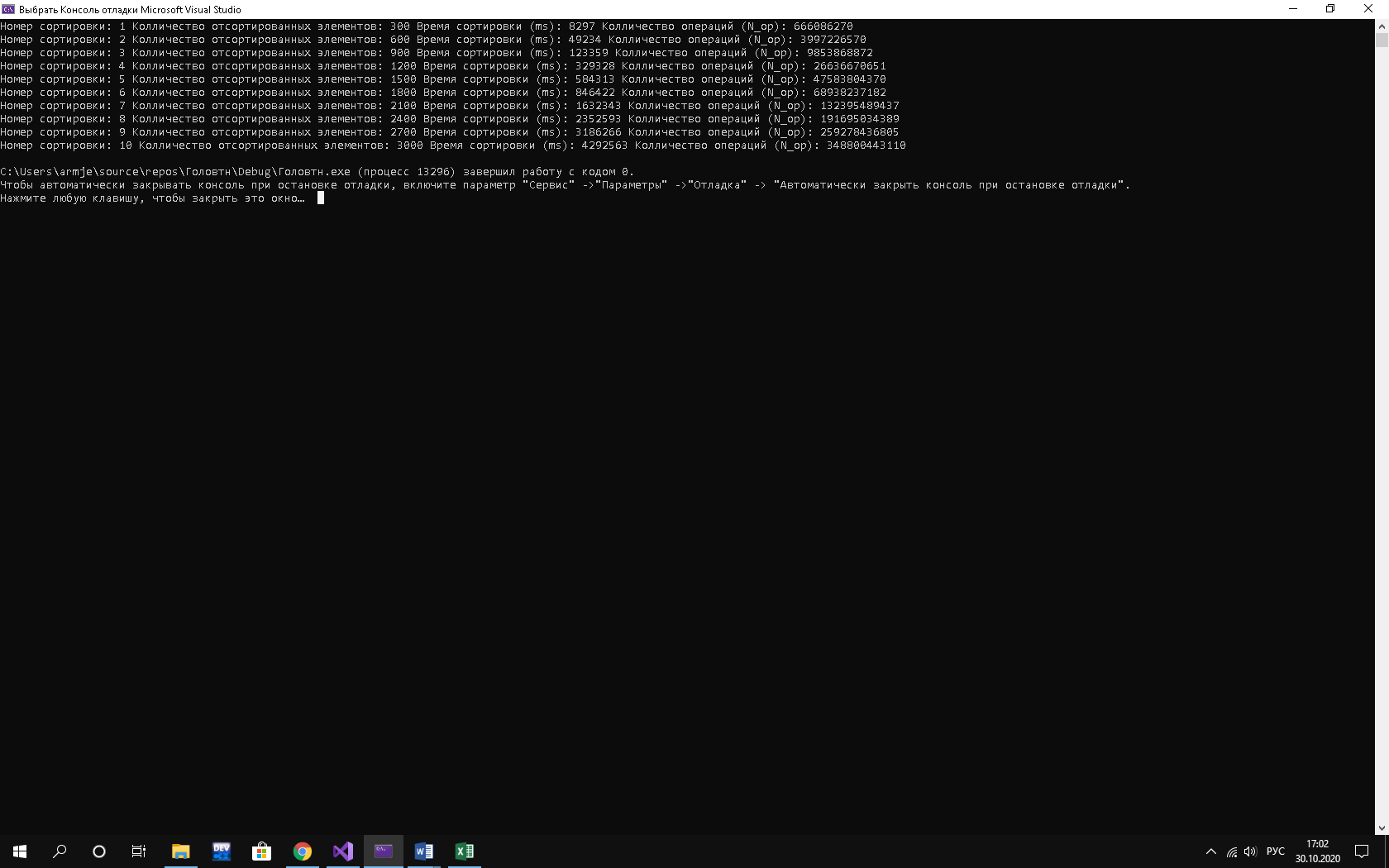
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 | 3558021322 | 270000000 | 2,078 | 666086270 |
| 600 | 28271982622 | 657000000 | 16,375 | 3997226570 |
| 900 | 95201883922 | 7290000000 | 20,109 | 9853868872 |
| 1200 | 225490775222 | 1728000000 | 59,547 | 26636670651 |
| 1500 | 439949506522 | 3 375 000 000000 | 108,859 | 47583804370 |
| 1800 | 759887227822 | 5 832 000 0000 | 175,438 | 68938237182 |
| 2100 | 1206280889122 | 9 261 000 00000 | 276,657 | 132395494394 |
| 2400 | 1800190490422 | 13 824 000 00000 | 357,812 | 1927017797174 |
| 2700 | 2562676031722 | 19 683 000 00000 ‬ | 418,672 | 2528426203692 |
| 3000 | 3514797513022 | 27 000 000 00000 | 652,609 | 2851158710193 |

**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 1712233552 | 12993262,79 | 5,341682425 | 1,064 |
| 1726533290 | 13190839,69 | 7,072899704 | 1,691 |
| 4734292303 | 36252424,29 | 9,661371098 | 2,692 |
| 3786769698 | 29019094,16 | 8,465426411 | 3,738 |
| 4040311383 | 30994581,68 | 9,245782517 | 4,074 |
| 4331371925 | 33242513,02 | 11,02272496 | 5,011 |
| 4360203751 | 33474663,57 | 9,111193246 | 5,077 |
| 5031107091 | 38634815,67 | 9,390908305 | 5,745 |
| 6120963503 | 47065997,89 | 10,181671 | 7,892 |
| 5385763165 | 41372399,76 | 12,30332378 | 6,096 |



**Скриншот работы программы:**



**Выводы.**

По результатам экспериментов было установлено, что графики C1, C2, C3, C4 от N имеют линейную зависимость от количества элементов.

**Литература:**

1. Структуры данных и алгоритмы. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. – М.: Издательский дом «Вильямс», 2000

2. Д. Кнут. Искусство программирования для ЭВМ.

**Приложение 1. Применение счетчика операций N\_op.**

#include "stdafx.h"

#include <locale.h>

#include <ctime>

#include "windows.h"

#include <iostream>

using namespace std;

struct Node

{

// указатель на следующий элемент в очереди

Node \*next;

// значение элемента в очереди

int value;

};

class Ochered

{

public:

// Колличество элементов в списке

int N;

int i;

int size = 0;

int size\_1 = NULL;

// Счетчик колличества операций

unsigned long long int N\_op = 0;

// указатель на первый элемент

Node \*head;

// указатель на хвост очереди

Node \*tail;

// инициализация очереди

void Init()

{

head = NULL; N\_op++;

// При инициализации head и tail = NULL

tail = head; N\_op++;

}

// добавление элемента в начало очереди, x - значение этого элемента

void Add(int x)

{

// создание нового элемента

Node \*node = new Node; N\_op+=3;

// в поле next данного элемента записывается NULL

node->next = NULL; N\_op+=2;

// в поле value записывается значение ячейки очереди int

node->value = x; N\_op+=2;

// после инициализации и при создании первого элемента мы проходим по ветке else

if (tail != NULL)

{ N\_op++;

/\* добавление элемента через хвост в очереди,

tail указывает на первый элемент в очереди перед добавляемым

в поле next первого элемента записывается адрес нового первого элемента\*/

tail->next = node; N\_op+=2;

// tail начинает указывать на новый элемент в очереди

tail = node; N\_op++;

}

else

{ N\_op++;

// хвост указывает на первый элемент в очереди

tail = node; N\_op++;

// голова указывает на первый элемент в очереди

head = tail; N\_op++;

}

size++; N\_op+=2;

}

// проверка очереди на пустоту

bool Isempty()

{ N\_op++;

//true, если head = NULL; false, если head указывает на какой-либо элемент в очереди

return head == NULL;

}

// удаление элемента из конца очереди

int Del()

{

int val = NULL; N\_op++;

// если очередь не пуста

if (Isempty() != 1)

{ N\_op++;

// создаем указатель на структуру

Node \*tmp;

// в указателюь tmp кладем ссылку на next предпоследнего элемента в очереди

tmp = head->next; N\_op+=2;

// в val записываем значение последнего элемента в очереди

val = head->value; N\_op+=2;

// удаляем объект по указателю head (последний элемент в очереди)

delete head;

// head начинает указывать на предпоcледний элемент в очереди (последний удален)

head = tmp; N\_op++;

// После удаления последнего элемента из очереди происходит повторная инициализация

if (Isempty() == 1)

{ N\_op++;

Init();

}

size--; N\_op+=2;

// возвращаем значение последнего элемента в очереди

return val;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

// получение значения из конца очереди

int Value()

{

// если очередь не пуста

if (Isempty() != 1)

{ N\_op+=2;

// через указатель head получить значение value последнего элемента в очереди

return head->value;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

};

// Класс наследник

class Numbers : public Ochered

{

public:

// получение значения из списка, x - номер элемента в списке

int get(int x)

{

// если очередь не пуста

if (Isempty() != 1)

{ N\_op++;

size\_1 = size; N\_op++;

int val1 = NULL; N\_op++;

/\* Перебор всех значений в списке\*/ N\_op++;

for (int i = 1; i < size\_1 + 1; i++)

{

if (i == x)

{ N\_op++;

// Когда i доходит до номера получаемого элемента, то в val записывается значение данного элемента

val1 = Value(); N\_op++;

}

// Перебор начинается только в случае если x не равен нулю

if (x != 1)

{ N\_op++;

// Вначале элемент удаляется из конца очереди, и сразу же добавляется в начало очереди, и так в цикле пока не восстановится первоначальное состояние очереди

Add(Del());

}

else

{ N\_op++;

// В противном случае сразу выходим из цикла for

break;

}

}

// Возвращается значение элемента под номером x из очереди

return val1;

}

else { cout << "Очередь пуста!" << endl; }

return 0;

}

// установка значения элемента в списке, x - номер элемента в списке, y - значение элемента в списке

void set(int x, int y)

{

// если очередь не пуста

if (Isempty() != 1)

{ N\_op++;

if (x <= size)

{ N\_op++;

size\_1 = size; N\_op++;

N\_op++;

for (int i = 1; i < size\_1 + 1; i++)

{

if (i == x)

{ N\_op++;

// Запись значения y в список

head->value = y; N\_op+=2;

}

if (x != 1)

{ N\_op++;

Add(Del());

}

else

{ N\_op++;

break;

}

}

}

// Если элемент устанавливается за пределы изначального списка

else

{ N\_op++;

if (x == size + 1)

{ N\_op+=2;

//Если номер элемента в списке на 1 больше размера очереди, значит элемент добавляется в начало очереди

Add(y);

}

else

{ N\_op+=2; N\_op++;

for (i = size + 1; i < x; i++)

{

// В противном случае очередь заполняется нулями

Add(0);

}

Add(y);

}

}

}

else { cout << "Очередь пуста!" << endl; }

}

void sort(int N)

{

int s, f, d, i, j, g, k, p;

s = 1; N\_op++;

do

{

s = 1 - s; N\_op+=2;

d = 1; N\_op++;

f = 1; N\_op++;

if (s == 0)

{ N\_op++;

i = 1; N\_op++;

j = N; N\_op++;

k = N + 1; N\_op+=2;

g = 2 \* N; N\_op+=2;

}

else

{ N\_op++;

i = N + 1; N\_op+=2;

j = 2 \* N; N\_op+=2;

k = 1; N\_op++;

g = N; N\_op++;

}

while (i != j)//

{ N\_op++;

// if(K[i]<K[j])

if (get(i) > get(j))

{ N\_op++;

// R[k] = R[j];

set(k, get(j));

k = k + d; N\_op+=2;

j = j - 1; N\_op+=2;

// if (!(K[j+1]<=K[j]))

if (get(j + 1) <= get(j))

{

N\_op+=2;

}

else

{ N\_op+=2;

do

{

// R[k]=R[i];

set(k, get(i));

k = k + d; N\_op+=2;

i = i + 1; N\_op+=2; N\_op+=2;

}

// while(!K[i-1]>=K[i]);

while (get(i - 1) < get(i));

f = 0; N\_op++;

d = -d; N\_op++;

p = k; N\_op++;

k = g; N\_op++;

g = p; N\_op++;

}

}

else

{ N\_op++;

// R[k]=R[i];

set(k, get(i));

k = k + d; N\_op+=2;

i = i + 1; N\_op+=2;

// if(K[i-1]<=K[i])

if (get(i - 1) <= get(i))

{

N\_op+=2;

}

else

{ N\_op+=2;

do

{

// R[k] = R[j];

set(k, get(j));

k = k + d; N\_op+=2;

j = j - 1; N\_op+=2; N\_op+=2;

}

// while(K[j+1]>=K[j]);

while (get(j + 1) < get(j));

f = 0; N\_op++;

d = -d; N\_op++;

p = k; N\_op++;

k = g; N\_op++;

g = p; N\_op++;

}

}

}

//R[k]=R[i];

set(k, get(i)); N\_op++;

} while (f != 1);

if (s == 0)

{ N\_op+=2;

for (int i = 1; i <= N; i++)

{ N\_op++;

// R[i] = R[i+n];

set(i, get(i + N));

}

}

size\_1 = size; N\_op+=2;

for (int i = 1; i <= size\_1; i++)

{

if (i <= size / 2)

{ N\_op+=2;

Add(Del());

}

else

{ N\_op+=2;

Del();

}

}

}

};

int main()

{

setlocale(LC\_ALL, "ru");

srand(time(NULL));

// Схема эксперимента

// Инициализация очереди и заполнение хранилища ключей

int i, t\_s, t\_f;

// Хранилище ключей

int Key[3000];

int N = 300;

Numbers list;

list.Init();

for (i = 0; i < 3000; i++)

{

// Заполнение хранилища ключей случайными числами до 1000

Key[i] = rand() % 999;

}

for (i = 0; i < 10; i++)

{

for (int z = N - 300; z < N; z++)

{

list.Add(Key[z]);

}

list.N\_op = 0;

t\_s = GetTickCount();

list.sort(N);

t\_f = GetTickCount();

cout << "Номер сортировки: " << i + 1 << " Колличество отсортированных элементов: " << N << " Время сортировки (ms): " << t\_f - t\_s << " Колличество операций (N\_op): " << list.N\_op << endl;

// Шаг в 300 элементов

N = N + 300;

}

return 0;

}