|  |
| --- |
| 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 № 57**

Выполнил: студент 2 курса

группы БИСО-03-19

шифр 19Б1157

Симкин Николай Андреевич  
*(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 № 57.**

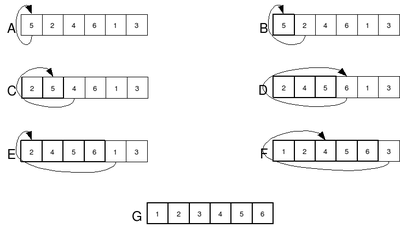
**Реализация связи элементов линейного списка: Указатели**

**Способ организации линейного связанный список: Дек**

**Алгоритм сортировки: Простая вставка**

**Теория о сортировках.**

Сортировка вставками — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Вычислительная сложность — O(n2)



На вход алгоритма подаётся последовательность*n* {\displaystyle n}чисел: a1, a2, … , an{\displaystyle a\_{1},a\_{2},...,a\_{n}}. Сортируемые числа также называют *ключами*. Входная последовательность на практике представляется в виде *n* массива с {\displaystyle n}элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности a1’, a2’, … , an’{\displaystyle a\_{1}^{'},a\_{2}^{'},...,a\_{n}^{'}}, чтобы выполнялось следующее соотношение a1’ ≤ a2’ ≤ … ≤ an’{\displaystyle a\_{1}^{'}\leqslant a\_{2}^{'}\leqslant ...\leqslant a\_{n}^{'}}.

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.

{\displaystyle O(n^{2})}OO

**Листинг программы с расчетами.**

Main.cpp

#include <iostream>

#include <chrono>

#include "MyDeque.h"

long long N\_op;

int main()

{

setlocale(LC\_ALL, "Rus");

for (int i = 1; i <=10; i++)

{

N\_op = 0;

MyDeque deque;

for(int j = 0; j < (i \* 300) / 2; j++)

{

deque.push\_back(std::rand() % 10);

deque.push\_front(std::rand() % 10);

}

auto start = std::chrono::system\_clock::now();

deque.sort();

auto end = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed = end - start;

std::cout << i << ") количество элементов: " << i \* 300 << "; время сортировки: " << elapsed.count() << "; N\_op: "<<N\_op<<std::endl;

}

return 0;

}

MyDeque.h

#ifndef MYDEQUE\_H

#define MYDEQUE\_H

#include <iostream>

struct Element

{

int value;

Element\* nextElement;

Element\* previousElement;

};

class MyDeque

{

private:

Element\* firstElement;

Element\* lastElement;

unsigned int sizeDeque;

bool checkSize(unsigned int);

void set\_front(unsigned int, double);

int get\_front(unsigned int);

public:

MyDeque();

MyDeque(MyDeque\*);

void push\_back(double);

void push\_front(double);

int pop\_back();

int pop\_front();

int back();

int front();

void clear();

void clear\_back(unsigned int);

void clear\_front(unsigned int);

void sort();

unsigned int getSize();

~MyDeque();

};

#endif // MYDEQUE\_H

MyDeque.cpp

#include "MyDeque.h"

bool MyDeque::checkSize(unsigned int index) //2

{

if (sizeDeque >= index) return true; //2

else return false;

}

void MyDeque::set\_front(unsigned int index, double value) //28 + 49n

{

if (checkSize(index)) //2 + 1 + 26n + 14 + 10 + 1 + 23n = 28 + 49n

{

for (unsigned int i = 0; i < index; i++) //1 + sigma(24 + 2) = 1 + 26n

{

push\_back(pop\_front()); //10 + 14 = 24

}

pop\_front(); //14

push\_front(value); //10

for (unsigned int i = 0; i < index; i++) //1 + sigma(21 + 2) = 1 + 23n

{

push\_front(pop\_back()); //10 + 11 = 21

}

}

}

int MyDeque::get\_front(unsigned int index) //9 + 49n

{

int result(0); //1

if (checkSize(index)) //2 + 1 + 26n + 4 + 1 + 23n = 8 + 49n

{

for (unsigned int i = 0; i < index; i++) //1 + sigma(24 + 2) = 1 + 26n

{

push\_back(pop\_front()); //10 + 14 = 24

}

result = front(); //4

for (unsigned int i = 0; i < index; i++) //1 + sigma(21 + 2) = 1 + 23n

{

push\_front(pop\_back()); //10 + 11 = 21

}

}

return result;

}

MyDeque::MyDeque() : firstElement(nullptr), lastElement(nullptr), sizeDeque(0)

{ }

MyDeque::MyDeque(MyDeque\* deque) : MyDeque()

{

Element\* element = deque->firstElement;

for (unsigned int i = deque->getSize(); i > 0; --i)

{

push\_back(element->value);

element = element->nextElement;

}

}

void MyDeque::push\_back(double value) //10

{

Element\* element = new Element; //1

element->value = value; //2

sizeDeque++; //1

if (!firstElement) //6

{

firstElement = element;

lastElement = firstElement;

}

else

{

element->previousElement = lastElement; //2

lastElement->nextElement = element; //2

lastElement = element; //1

}

}

void MyDeque::push\_front(double value) //10

{

Element\* element = new Element; //1

element->value = value; //2

sizeDeque++; //1

if (!firstElement) //6

{

firstElement = element;

lastElement = firstElement;

}

else

{

element->nextElement = firstElement; //2

firstElement->previousElement = element; //2

firstElement = element; //1

}

}

int MyDeque::pop\_back() //11

{

double result(0); //1

if (sizeDeque != 0) //10

{

result = lastElement->value; //2

if (sizeDeque > 1) //7

{

Element\* element = lastElement; //1

lastElement = lastElement->previousElement; //2

lastElement->nextElement = nullptr; //2

delete element;

sizeDeque--; //1

}

else

{

firstElement = lastElement = 0;

sizeDeque--;

}

}

return result;

}

int MyDeque::pop\_front() //14

{

double result(0); //1

if (sizeDeque != 0) //13

{

result = firstElement->value; //2

if (firstElement->nextElement) //10

{

Element\* element = firstElement; //1

firstElement = firstElement->nextElement; //2

if (sizeDeque != 1) //3

{

firstElement->previousElement = nullptr; //2

}

delete element;

sizeDeque--; //1

}

else if (firstElement == lastElement)

{

firstElement->nextElement = nullptr;

firstElement = nullptr;

delete firstElement;

sizeDeque = 0;

}

}

return result;

}

int MyDeque::back()

{

if (!sizeDeque == 0) return lastElement->value;

else

{

std::cout << "Error!" << std::endl;

std::exit(-1);

}

}

int MyDeque::front() //

{

if (!sizeDeque == 0) return firstElement->value; //3

else

{

std::cout << "Error!" << std::endl;

std::exit(-1);

}

}

void MyDeque::clear()

{

for (unsigned int i = getSize(); i > 0; --i)

{

lastElement = firstElement->nextElement;

delete firstElement;

firstElement = lastElement;

}

sizeDeque = 0;

}

void MyDeque::clear\_back(unsigned int index)

{

if (checkSize(index))

{

Element\* element;

for (unsigned int i = 0; i < index; i++)

{

element = lastElement;

lastElement = element->previousElement;

delete element;

}

sizeDeque -= index;

}

}

void MyDeque::clear\_front(unsigned int index)

{

if (checkSize(index))

{

Element\* element;

for (unsigned int i = 0; i < index; i++)

{

element = firstElement;

firstElement = element->nextElement;

delete element;

}

sizeDeque -= index;

}

}

void MyDeque::sort() //196n^3 + 128n^2 + 15n + 3

{

double temp(0); //1

int item(0); //1

for (unsigned int counter = 1; counter < getSize(); counter++) //1 + sigma(3 + 10 + 49n + 2 + 79n + 196n^2) = 1 + 15n + 128n^2 + 196n^3

{

temp = get\_front(counter); // 1 + 9 + 49n = 10 + 49n

item = counter - 1; //2

while (get\_front(item) > temp && item >= 0) //sigma(9 + 49n + 3 + 38 + 98n + 28 + 49n + 1) = 79n + 196n^2

{

set\_front(item + 1, get\_front(item)); //28 + 49n + 1 + 9 + 49n = 38 + 98n

set\_front(item, temp); //28 + 49n

item--; //1

}

}

}

unsigned int MyDeque::getSize() //1

{

return sizeDeque; //1

}

MyDeque::~MyDeque()

{

clear();

}

O(F(n)) = 196n3

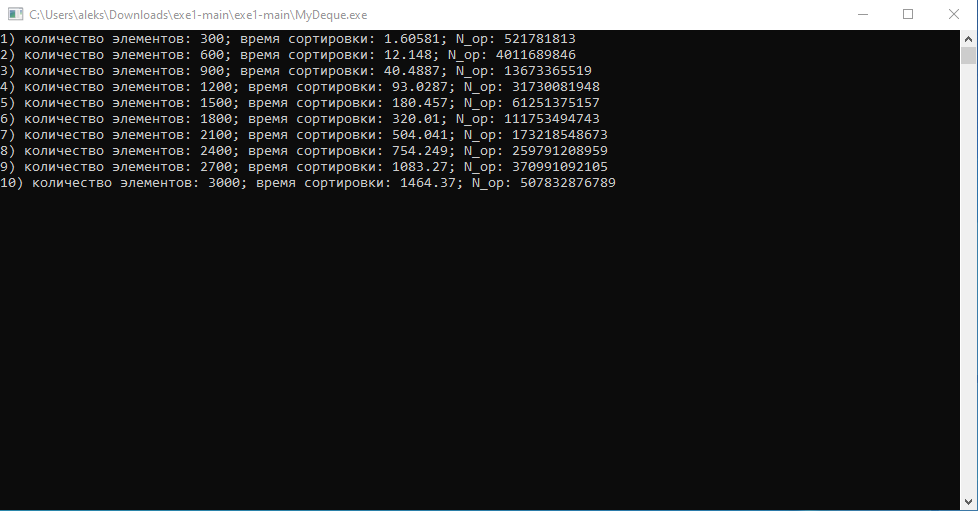
F(n) = 196n3 + 128n2 + 15n + 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 | 5303524503 | 5292000000 | 1,605 | 521781813 |
| 600 | 42382089003 | 42336000000 | 12,148 | 4011689846 |
| 900 | 142987693503 | 142884000000 | 40,488 | 13673365519 |
| 1200 | 338872338003 | 338688000000 | 93,028 | 31730081948 |
| 1500 | 661788022503 | 661500000000 | 180,028 | 62554183040 |
| 1800 | 1143486747003 | 1143072000000 | 320,01 | 109967630608 |
| 2100 | 1815720511503 | 1815156000000 | 504,041 | 178926854152 |
| 2400 | 2710241316003 | 2709504000000 | 754,249 | 261913685051 |
| 2700 | 3858801160503 | 3857868000000 | 1083,27 | 365734898109 |
| 3000 | 5293152045003 | 5292000000000 | 1464,37 | 514742984645 |

**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 |
| 3304376637 | 3297196262 | 10,164 | 10,142 |
| 3488812068 | 3485018110 | 10,564 | 10,553 |
| 3531606735 | 3529045643 | 10,457 | 10,449 |
| 3642691856 | 3640710324 | 10,679 | 10,674 |
| 3676028298 | 3674428422 | 10,804 | 10,799 |
| 3573284419 | 3571988375 | 10,398 | 10,394 |
| 3602327016 | 3601207045 | 10,147 | 10,144 |
| 3593297858 | 3592320308 | 10,347 | 10,345 |
| 3562178552 | 3561317123 | 10,550 | 10,548 |
| 3614627481 | 3613840764 | 10,283 | 10,280 |

**Скриншот работы программы:**



**Выводы.**

По результатам экспериментов было установлено, что графики C1, C2, C3, C4 от N имеют нелинейную 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.**

**Main.cpp**

#include <iostream>

#include <chrono>

#include "MyDeque.h"

long long N\_op;

int main()

{

setlocale(LC\_ALL, "Rus");

for (int i = 1; i <= 10; i++)

{

N\_op = 0;

MyDeque deque;

for (int j = 0; j < (i \* 300) / 2; j++)

{

deque.push\_back(std::rand() % 10);

deque.push\_front(std::rand() % 10);

}

auto start = std::chrono::system\_clock::now();

deque.sort();

auto end = std::chrono::system\_clock::now();

std::chrono::duration<double> elapsed = end - start;

std::cout << i << ") количество элементов: " << i \* 300 << "; время сортировки: " << elapsed.count() << "; N\_op: " << N\_op << std::endl;

}

int x;

std::cin >> x;

system("pause");

return 0;

}

**MyDeque.h**

#ifndef MYDEQUE\_H

#define MYDEQUE\_H

#include <iostream>

struct Element

{

int value;

Element\* nextElement;

Element\* previousElement;

};

class MyDeque

{

private:

Element\* firstElement;

Element\* lastElement;

unsigned int sizeDeque;

bool checkSize(unsigned int);

void set\_front(unsigned int, double);

int get\_front(unsigned int);

public:

MyDeque();

MyDeque(MyDeque\*);

void push\_back(double);

void push\_front(double);

int pop\_back();

int pop\_front();

int back();

int front();

void clear();

void clear\_back(unsigned int);

void clear\_front(unsigned int);

void sort();

unsigned int getSize();

~MyDeque();

};

#endif // MYDEQUE\_H

**MyDeque.cpp**

#include "MyDeque.h"

extern long long N\_op;

bool MyDeque::checkSize(unsigned int index)

{

if (sizeDeque >= index)

{

N\_op++;

return true;

}

else

{

N\_op++;

return false;

}

}

void MyDeque::set\_front(unsigned int index, double value)

{

if (checkSize(index))

{

N\_op++;

for (unsigned int i = 0; i < index; i++)

{

N\_op += 2;

push\_back(pop\_front());

}

pop\_front();

push\_front(value);

N\_op++;

for (unsigned int i = 0; i < index; i++)

{

N\_op += 2;

push\_front(pop\_back());

}

}

}

int MyDeque::get\_front(unsigned int index)

{

int result(0); N\_op++;

if (checkSize(index))

{

N\_op++;

for (unsigned int i = 0; i < index; i++)

{

N\_op += 2;

push\_back(pop\_front());

}

result = front(); N\_op++;

N\_op++;

for (unsigned int i = 0; i < index; i++)

{

N\_op += 2;

push\_front(pop\_back());

}

}

return result;

}

MyDeque::MyDeque() : firstElement(nullptr), lastElement(nullptr), sizeDeque(0)

{

N\_op += 3;

}

MyDeque::MyDeque(MyDeque\* deque) : MyDeque()

{

Element\* element = deque->firstElement; N\_op += 2;

N\_op++;

for (unsigned int i = deque->getSize(); i > 0; --i)

{

N\_op += 2;

push\_back(element->value); N\_op += 2;

element = element->nextElement; N\_op += 2;

}

}

void MyDeque::push\_back(double value)

{

Element\* element = new Element; N\_op++;

element->value = value; N\_op += 2;

sizeDeque++; N\_op++;

if (!firstElement)

{

N\_op++;

firstElement = element; N\_op++;

lastElement = firstElement; N\_op++;

}

else

{

N\_op++;

element->previousElement = lastElement; N\_op += 2;

lastElement->nextElement = element; N\_op += 2;

lastElement = element; N\_op++;

}

}

void MyDeque::push\_front(double value)

{

Element\* element = new Element; N\_op++;

element->value = value; N\_op += 2;

sizeDeque++; N\_op++;

if (!firstElement)

{

N\_op++;

firstElement = element; N\_op++;

lastElement = firstElement; N\_op++;

}

else

{

N\_op++;

element->nextElement = firstElement; N\_op += 2;

firstElement->previousElement = element; N\_op += 2;

firstElement = element; N\_op++;

}

}

int MyDeque::pop\_back()

{

double result(0); N\_op++;

if (sizeDeque != 0)

{

N\_op++;

result = lastElement->value; N\_op += 2;

if (sizeDeque > 1)

{

N\_op++;

Element\* element = lastElement; N\_op++;

lastElement = lastElement->previousElement; N\_op += 2;

lastElement->nextElement = nullptr; N\_op += 2;

delete element;

sizeDeque--; N\_op++;

}

else

{

N\_op++;

firstElement = lastElement = 0; N\_op += 2;

sizeDeque--; N\_op++;

}

}

return result;

}

int MyDeque::pop\_front()

{

double result(0); N\_op++;

if (sizeDeque != 0)

{

N\_op++;

result = firstElement->value; N\_op += 2;

if (firstElement->nextElement)

{

N\_op++;

Element\* element = firstElement; N\_op++;

firstElement = firstElement->nextElement; N\_op += 2;

if (sizeDeque != 1)

{

N\_op++;

firstElement->previousElement = nullptr; N\_op++;

}

delete element;

sizeDeque--; N\_op++;

}

else if (firstElement == lastElement)

{

N\_op++;

firstElement->nextElement = nullptr; N\_op += 2;

firstElement = nullptr; N\_op++;

delete firstElement;

sizeDeque = 0; N\_op++;

}

}

return result;

}

int MyDeque::back()

{

if (!sizeDeque == 0)

{

N\_op += 3;

return lastElement->value;

}

else

{

std::cout << "Error!" << std::endl;

std::exit(-1);

}

}

int MyDeque::front()

{

if (!sizeDeque == 0)

{

N\_op += 3;

return firstElement->value;

}

else

{

std::cout << "Error!" << std::endl;

std::exit(-1);

}

}

void MyDeque::clear()

{

N\_op++;

for (unsigned int i = getSize(); i > 0; --i)

{

N\_op += 2;

lastElement = firstElement->nextElement; N\_op += 2;

delete firstElement;

firstElement = lastElement; N\_op++;

}

sizeDeque = 0; N\_op++;

}

void MyDeque::clear\_back(unsigned int index)

{

if (checkSize(index))

{

Element\* element;

N\_op++;

for (unsigned int i = 0; i < index; i++)

{

N\_op += 2;

element = lastElement; N\_op++;

lastElement = element->previousElement; N\_op += 2;

delete element;

}

sizeDeque -= index; N\_op++;

}

}

void MyDeque::clear\_front(unsigned int index)

{

if (checkSize(index))

{

Element\* element;

N\_op++;

for (unsigned int i = 0; i < index; i++)

{

N\_op += 2;

element = firstElement; N\_op++;

firstElement = element->nextElement; N\_op += 2;

delete element;

}

sizeDeque -= index; N\_op++;

}

}

void MyDeque::sort()

{

double temp(0); N\_op++;

int item(0); N\_op++;

N\_op++;

for (unsigned int counter = 1; counter < getSize(); counter++)

{

N\_op += 2;

temp = get\_front(counter); N\_op++;

item = counter - 1; N\_op += 2;

while (get\_front(item) > temp && item >= 0)

{

N\_op += 4;

set\_front(item + 1, get\_front(item));

set\_front(item, temp);

item--; N\_op++;

}

}

}

unsigned int MyDeque::getSize()

{

return sizeDeque;

}

MyDeque::~MyDeque()

{

clear();

}