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

Институт кибербезопасности и цифровых технологий

Кафедра КБ-3 «Управление и моделирование систем»

**ОТЧЕТ   
о выполнении лабораторной работы №1**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Алгоритмы и структуры данных»**

**Вариант № 2**

Выполнил: студент 2 курса

группы БСБО-05-21

шифр 21Б1002

Филатов Николай Александрович  
 *(фио студента)*

Проверил:

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

Москва 2022 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 № 2.**

**Реализация связи элементов линейного списка:** указатели

**Способ организации линейного связанный список:** очередь с головой и хвостом

**Алгоритм сортировки:** сравнение и подсчет

**Теория о сортировках.**

**Сортировка подсчетом**

Метод сортировки подсчетом основан на том, что jй ключ в окончательно упорядоченной последовательности превышает ровно (j – 1) из остальных ключей. Иначе говоря, если известно, что некоторый ключ превышает ровно 27 других, то после сортировки соответствующая запись должна занять 28е место. Т.о. идея состоит в том, чтобы сравнить попарно все ключи и подсчитать, сколько из них меньше каждого отдельного ключа.

Очевидный способ выполнить сравнения – ((сравнить Кj с К i) при 1 j N) при 1 i N , но легко видеть, что более половины этих действий излишни, поскольку не нужно сравнивать ключ сам с собой, и после сравнения Ка с Кс не надо сравнивать Кс с Ка. Поэтому достаточно ((сравнить К j с К i ) при 1 j i ) при 1 i N .

**Листинг программы с расчетами.**

public class Item {

*// значение элемента в очереди*

public int value;

*// указатель на следующий элемент в очереди*

public Item next;

public Item(int value, Item next){ *//2*

this.value = value; *//1*

this.next = next; *//1*

}

public Item(int value) *//2*

{

this(value, null);

}

}

public class Queue{

public Item head = null;

public Item tail = null;

public Long N\_op = 0L;

public Queue() { }

public void push(Item value){ *//1+(3+1)+2+1+2=10*

if(head == null && tail == null){ *//3*

head = value; *//1*

}

else{

tail.next = value; *//2*

}

tail = value; *//1*

tail.next = null; *//2*

}

public Item pop() throws Exception{ *//1+1+1+1+2=6*

if (head == null){ *//1*

throw new Exception("The queue is empty");

}

if (head == tail){ *//1*

tail = null; *//1*

}

Item result = head; *//1*

head = head.next; *//2*

return result;

}

public int get(int index) throws Exception{ *//1+1+2+18n+2+1+17n=7+35n*

Item oldHead = head; *//1*

for(int i = 0; i < index; i++){ *//2 + 18\*n*

push(pop()); *//18*

}

int result = head.value; *//2*

while(head != oldHead){ *//1 + 17\*n*

push(pop()); *//17*

}

return result;

}

public void set(int index, int value) throws Exception{ *//2+1+2+18n+2+1+17n=8+35n*

Item oldHead = head; *//1*

for(int i = 0; i < index; i++){ *//18\*n*

push(pop()); *//18*

}

head.value = value; *//2*

while(head != oldHead){ *//1 + 17n*

push(pop());

}

}

public void print(){

Item current = head;

while (current != null){

System.*out*.print(current.value + " ");

current = current.next;

}

System.*out*.println();

}

}

public class CountingSort {

public static Queue sort(Queue queue, int min, int max) throws Exception{ *//3+4+2+2+8+5+14n+3n+18nk*

*// Инициализация массива подсчета*

int[] count = new int[max - min + 1]; *//4*

Queue sortedQueue = new Queue(); *//2*

Item oldTail = queue.tail; *//2*

Item currentItem = queue.pop(); *//8*

count[oldTail.value - min] += 1; *//5*

while (currentItem != oldTail){ *//14n*

count[currentItem.value - min] += 1; *//5*

currentItem = queue.pop(); *//8*

}

for (int i = 0; i < count.length; i++){ *//3n + 18nk*

for (int j = 0; j < count[i]; j++){ *//18k*

sortedQueue.push(new Item(i + min)); sortedQueue.N\_op+=6; *//16*

}

}

return sortedQueue;

}

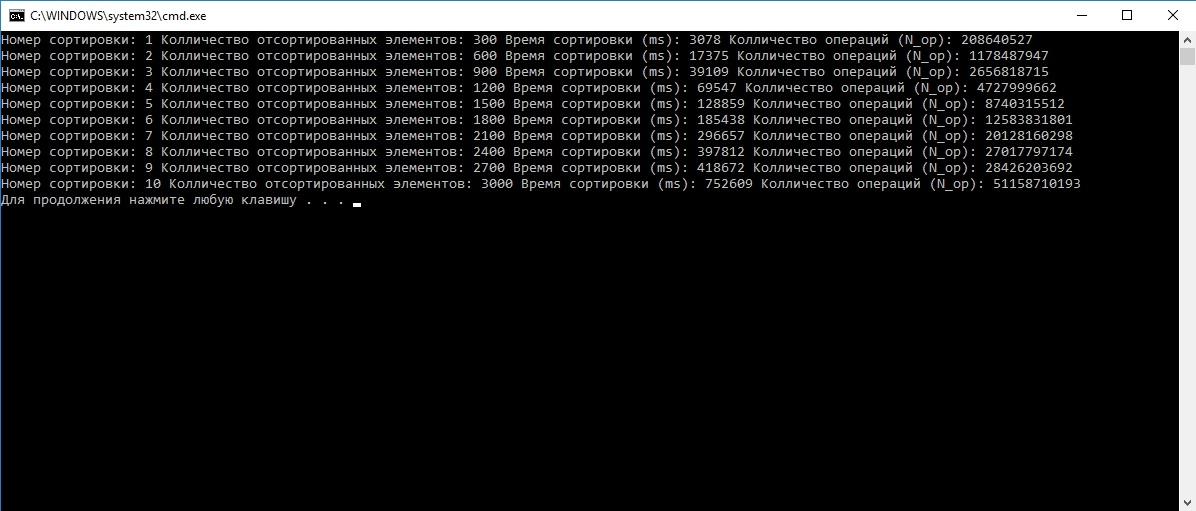
}

F(n)=

O(F(n)=n\*k

**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 |  |  | 0,254 |  |
| 600 |  |  | 0,032 | 1178487947 |
| 900 |  |  | 39,109 | 2656818715 |
| 1200 |  |  | 69,547 | 4727999662 |
| 1500 |  |  | 128,859 | 8740315512 |
| 1800 |  |  | 185,438 | 12583831801 |
| 2100 |  |  | 296,657 | 20128160298 |
| 2400 |  |  | 397,812 | 27017797174 |
| 2700 |  |  | 418,672 | 28426203692 |
| 3000 |  |  |  |  |

**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;

}