|  |
| --- |
| 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 № 98**

Выполнил: студент 2 курса

группы БИСО-03-19

шифр 19Б1398

Тепсикоев С.А

Проверил:

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

Москва 2020 г.

**Задание на лабораторную работу № 1.**

В рамках лабораторной работы №1 требуется программно реализовать (с помощью указателей (однонаправленных/двунаправленный динамический линейный 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 (конструктор, 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 № 98.**

**Реализация связи элементов линейного списка: Указатели**

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

**Алгоритм сортировки: Пирамидальная сортировка**

**Теория о сортировках**

Пирамидальная сортировка (или сортировка кучей, Heap Sort) — это метод сортировки сравнением, основанный на такой структуре данных как двоичная куча. Она похожа на сортировку выбором, где сначала ищется максимальный элемент и помещается в конец. После этого операция повторяется для следующих элементов.

**Алгоритм пирамидальной сортировки:**

1. Построение кучи из входных данных. После построения максимальный элемент будет храниться в корне кучи.
2. Замена максимального элемента на последний элемент кучи и уменьшение её размера на 1.
3. Преобразование полученного дерева в кучу с новым корнем.
4. Повторение вышеуказанных шагов до тех пор, пока размер кучи больше 1.

**Как построить кучу?**

Процедура преобразования в кучу может быть применена к узлу, только если его дочерние узлы уже преобразованы. Таким образом, преобразование должно выполняться снизу вверх.

Пример:

Входные данные: 4, 10, 3, 5, 1

4(1)

/ \

10(2) 3(3)

/ \

5(4) 1(5)

Числа в скобках представляют индексы в представлении данных в виде массива.

Применение процедуры преобразования в кучу к индексу 2:

4(1)

/ \

10(2) 3(3)

/ \

5(4) 1(5)

Применение процедуры преобразования в кучу к индексу 1:

10(1)

/ \

5(2) 3(3)

/ \

4(4) 1(5)

Процедура преобразования в кучу вызывает себя рекурсивно для создания кучи сверху вниз.

**Листинг программы с расчетами.**

using System; //+1

using System.Diagnostics; //+2

namespace LinkedItem98

{

public class Dequeue

{

public long ellapledTicks;

int count; // количество элементов в списке

Item head; // +6 головной/первый элемент

Item tail; // +6 последний/хвостовой элемент

public ulong n\_op = 16; // учитывая класс Item и все до него 1 + 2 + 6 + 6 + 1

## public void N\_op(ulong x) // фунция для увеличения числа операций + 1

{

n\_op = n\_op + x + 4; // + 3

}

## public int Count { get { N\_op(2); return count; } } // защита от изменения размерности списка

## public void sort(Dequeue ob, int ct)//Сортировка

{

ellapledTicks = DateTime.Now.Ticks; // +3

int n = ct; //+ 1

N\_op(7); // + 1 вызов функции

// Построение кучи

N\_op(5); // +1

for (int i = n / 2 - 1; i >= 0; i--)

{

heapify(ob, n, i); //+1

N\_op(5); //+1

}

N\_op(4);//+1

// Извлекаем элементы

for (int i = n - 1; i >= 0; i--)//24\*n

{

// Перемещаем текущий корень в конец

int temp = ob.GetObj(0).Data;//+4

N\_op(5);//+1

ob.GetObj(0).Data = ob.GetObj(i).Data;//+7

N\_op(8);//+1

ob.GetObj(i).Data = temp;//+4

N\_op(5);//+1

heapify(ob, i, 0); // +1 вызываем процедуру heapify на уменьшенной куче

N\_op(2);//+1

N\_op(4);//+1

}

ellapledTicks = DateTime.Now.Ticks - ellapledTicks; N\_op(2);//+4

N\_op(5);//+1

}

## public void heapify(Dequeue ob, int n, int i) // Процедура для преобразования в двоичную кучу поддерева с корневым узлом i, что является

{

N\_op(4);//+1

int largest = i;//+1

N\_op(2);//+1

// Инициализируем наибольший элемент как корень

int l = 2 \* i + 1; //+3 left = 2\*i + 1

N\_op(4);//+1

int r = 2 \* i + 2; //+3 right = 2\*i + 2

N\_op(4);

// Если левый дочерний элемент больше корня

if (l < n && ob.GetObj(l).Data > ob.GetObj(largest).Data)

{

N\_op(2);//+1

largest = l;//+1

}

N\_op(11);//+1

// Если правый дочерний элемент больше, чем самый большой элемент на данный момент

if (r < n && ob.GetObj(r).Data > ob.GetObj(largest).Data)

{

N\_op(2);//+1

largest = r;//+1

}

N\_op(11);//+1

// Если самый большой элемент не корень

if (largest != i)//+1

{

int swap = ob.GetObj(i).Data;//+4

N\_op(5);//+1

ob.GetObj(i).Data = ob.GetObj(largest).Data;//+7

N\_op(8);//+1

ob.GetObj(largest).Data = swap;//+4

N\_op(5);//+1

// Рекурсивно преобразуем в двоичную кучу затронутое поддерево

heapify(ob, n, largest);//+1

N\_op(2);//+1

}

N\_op(2);//+1

}

## public void AddFirst(int data) //30\*n добавление элемента в начало

{

N\_op(2);//+1

Item node = new Item(data); //+8 Item() (+6) new (+1)

N\_op(9);//+1

Item temp = head;//+7

N\_op(8);//+1

node.Next = temp;//+2

N\_op(3);//+1

head = node;//+1

N\_op(2);//+1

N\_op(2);//+1

if (count == 0) //+1

{

tail = head;//+1

N\_op(2);//+1

}

else

{

temp.Previous = node;//+2

N\_op(3);//+1

}

count++;//+2

N\_op(3);//+1

}

## public void AddLast(int data) //24\*n добавление элемента в конец

{

N\_op(2);//+1

Item node = new Item(data);//+8

N\_op(9);//+1

N\_op(2);//+1

if (head == null)//+1

{

head = node;//+1

N\_op(2);//+1

}

else

{

tail.Next = node;//+2

N\_op(3);//+1

node.Previous = tail;//+2

N\_op(3);//+1

}

tail = node;//+1

N\_op(2);//+1

count++;//+2

N\_op(3);//+1

}

## public void RemoveFirst() // удаление первого элемента

{

if (count == 0)//+1

{

Console.WriteLine("Список пуст");//+2

N\_op(3);//+1

}

N\_op(2);//+1

int output = head.Data;//+2

N\_op(3);//+1

N\_op(2);//+1

if (count == 1)//+1

{

head = tail = null;//+2

N\_op(3);//+1

}

else

{

head = head.Next;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

}

count--;//+2

N\_op(3);//+1

}

## public void RemoveLast() //удаление последнего элемента

{

if (count == 0) // +1

{

Console.WriteLine("Список пуст");//+2

N\_op(3);//+1

}

N\_op(2);//+1

int output = tail.Data;//+2

N\_op(3);//+1

N\_op(2);//+1

if (count == 1)//+1

{

head = tail = null;//+2

N\_op(3);//+1

}

else

{

tail = tail.Previous;//+2

N\_op(3);//+1

tail.Next = null;//+2

N\_op(3);//+1

}

count--;//+2

N\_op(3);//+1

}

## public void ShowFirst() // показать первый элемент

{

N\_op(2);//+1

if (Count == 0) //1

{

Console.WriteLine("Список пуст");//+1

N\_op(3);//+1

}

else

{

Console.WriteLine($"Первый элемент списка: {head.Data}");//+3

N\_op(4);//+1

}

}

## public void ShowLast() // показать последний элемент

{

N\_op(2);//+1

if (Count == 0) //+1

{

Console.WriteLine("Список пуст");//+2

N\_op(3);//+1

}

else

{

Console.WriteLine($"Последний элемент списка: { tail.Data}");//+3

N\_op(4);//+1

}

}

## public void isEmpty() // проверка списка на пустоту

{

N\_op(2);//+1

if (Count == 0) //+1

{

Console.WriteLine("Список пуст");//+2

N\_op(3);//+1

}

}

## public void SizeList() // получение размера списка

{

Console.WriteLine($"Размер списка:{Count}");//+3

N\_op(4);//+1

}

## public void Clear() // очистить список

{

if (Count != 0)//+1

{

N\_op(3);//+1

for (int i = Count; i > 0; i--)

{

N\_op(2);//+1

if (Count == 1)//+1

{

head = tail = null;//+2

N\_op(3);//+1

}

else

{

tail = tail.Previous;//+2

N\_op(3);//+1

tail.Next = null;//+2

N\_op(3);//+1

}

count--;//+2

N\_op(3);//+1

N\_op(4);//+1

}

}

N\_op(2);//+1

Console.WriteLine("Список очищен");//+2

N\_op(3);//+1

}

## public void FrontClear(int index) // очистить список до i-го элемента начиная с головы

{

N\_op(2);//+1

N\_op(3);//+1

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

{

N\_op(2);//+1

if (count == 1) // +1

{

head = tail = null;//+2

N\_op(3);//+1

}

else

{

head = head.Next;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

}

count--;//+2

N\_op(3);//+1

N\_op(4);//+1

}

Console.WriteLine($"Список очищен до {2}-й позиции начиная с головы");//+3

N\_op(4);//+1

}

## public void LastClear(int index) // очистить список до i-го элемента начиная с конца

{

N\_op(2);//+1

N\_op(3);//+1

for (int i = index; i > 0; i--)

{

N\_op(2);//+1

if (count == 1)//+1

{

head = tail = null;//+2

N\_op(3);//+1

}

else

{

tail = tail.Previous;//+2

N\_op(3);//+1

tail.Next = null;//+2

N\_op(3);//+1

}

count--;//+2

N\_op(3);//+1

N\_op(4);//+1

}

Console.WriteLine($"Список очищен до {2}-й позиции начиная с головы");//+3

N\_op(4);//+1

}

## public void AppointFront(int data, int index) // присваиваем новое значение i-му элементу начиная с головы

{

N\_op(3);//+1

N\_op(3);//+1

if (count == 0)//+1

{

Console.WriteLine("Список пуст");//+2

N\_op(3);//+1

}

else

{

Item current;//+6

N\_op(7);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = head;//+1

N\_op(2);//+1

head = head.Next;//+2

N\_op(3);//+1

tail.Next = current;//+2

N\_op(3);//+1

current.Previous = tail;//+2

N\_op(3);//+1

tail = current;//+1

N\_op(3);//+1

current.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

head.Data = data;//+2

N\_op(3);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = tail;//+1

N\_op(2);//+1

tail = tail.Previous;//+2

N\_op(3);//+1

head.Previous = current;//+2

N\_op(3);//+1

current.Next = head;//+2

N\_op(3);//+1

head = current;//+1

N\_op(2);//+1

tail.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

Console.WriteLine($"Значение {4}-го элемента изменено на {7} начиная с головы");//+4

N\_op(5);//+1

}

}

## public void AppointLast(int data, int index)// присваиваем новое значение i-му элементу начиная с конца

{

N\_op(3);//+1

N\_op(2);//+1

if (count == 0)//+1

{

Console.WriteLine("Список пуст");//+2

N\_op(3);//+1

}

else

{

Item current;//+6

N\_op(7);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = tail;//+1

N\_op(2);//+1

tail = tail.Previous;//+2

N\_op(3);//+1

head.Previous = current;//+2

N\_op(3);//+1

current.Next = head;//+2

N\_op(3);//+1

head = current;//+1

N\_op(2);//+1

tail.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

tail.Data = data;//+2

N\_op(3);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = head;//+1

N\_op(2);//+1

head = head.Next;//+2

N\_op(3);//+1

tail.Next = current;//+2

N\_op(3);//+1

current.Previous = tail;//+2

N\_op(3);//+1

tail = current;//+1

N\_op(2);//+1

current.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

Console.WriteLine($"Значение {3}-го элемента изменено на {7} начиная с конца");

N\_op(5);//+1

}

}

## public void GetFront(int index) // получаем значение i-го элемента начиная с головы

{

N\_op(2);//+1

N\_op(2);//+1

if (count == 0)//+1

{

Console.WriteLine("Список пуст");//+2

N\_op(3);//+1

}

else

{

Item current;//+6

N\_op(7);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = head;//+1

N\_op(2);//+1

head = head.Next;//+2

N\_op(3);//+1

tail.Next = current;//+2

N\_op(3);//+1

current.Previous = tail;//+2

N\_op(3);//+1

tail = current;//+1

N\_op(3);//+1

current.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

Console.WriteLine(head.Data);//+3

N\_op(4);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = tail;//+1

N\_op(2);//+1

tail = tail.Previous;//+2

N\_op(3);//+1

head.Previous = current;//+2

N\_op(3);//+1

current.Next = head;//+2

N\_op(3);//+1

head = current;//+1

N\_op(2);//+1

tail.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

}

}

## public void GetLast(int index) // получаем значение i-го элемента начиная с конца

{

N\_op(2);//+1

N\_op(2);//+1

if (count == 0)//+1

{

Console.WriteLine("Список пуст");//+2

N\_op(3);//+1

}

else

{

Item current;//+6

N\_op(7);//+1

int rez;

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = tail;//+1

N\_op(2);//+1

tail = tail.Previous;//+2

N\_op(3);//+1

head.Previous = current;//+2

N\_op(3);//+1

current.Next = head;//+2

N\_op(3);//+1

head = current;//+1

N\_op(2);//+1

tail.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

Console.WriteLine(tail.Data);//+3

N\_op(4);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = head;//+1

N\_op(2);//+1

head = head.Next;//+2

N\_op(3);//+1

tail.Next = current;//+2

N\_op(3);//+1

current.Previous = tail;//+2

N\_op(3);//+1

tail = current;//+1

N\_op(2);//+1

current.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

}

}

## public Item GetObj(int index) //24 + 47n получаем i-й элемент списка для пирамидальной сортировки

{

N\_op(2);//+1

Item current, rez;//+6

N\_op(7);//+1

N\_op(3);//+1

for (int i = 0; i < index; i++)// 16 + 24n

{

current = head;//+1

N\_op(2);//+1

head = head.Next;//+2

N\_op(3);//+1

tail.Next = current;//+2

N\_op(3);//+1

current.Previous = tail;//+2

N\_op(3);//+1

tail = current;//+1

N\_op(3);//+1

current.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(4);//+1

}

rez = head;//+1

N\_op(2);//+1

for (int i = 0; i < index; i++) //8 + 23n

{

current = tail;//+1

N\_op(2);//+1

tail = tail.Previous;//+2

N\_op(3);//+1

head.Previous = current;//+2

N\_op(3);//+1

current.Next = head;//+2

N\_op(3);//+1

head = current;//+1

N\_op(2);//+1

tail.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(4);//+1

}

N\_op(2);//+1

return rez;//+1

}

## public void SetObjFront(int data, int index) // дабавляем новый элемент списка в i-ю позицию начиная с головы

{

N\_op(3);//+1

Item current;//+6

N\_op(7);//+1

Item New = new Item(data);//+8

N\_op(9);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = head;//+1

N\_op(2);//+1

head = head.Next;//+2

N\_op(3);//+1

tail.Next = current;//+2

N\_op(3);//+1

current.Previous = tail;//+2

N\_op(3);//+1

tail = current;//+1

N\_op(3);//+1

current.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

current = head;//+1

N\_op(2);//+1

New.Next = current;//+2

N\_op(3);//+1

head = New;//+1

N\_op(2);//+1

N\_op(2);//+1

if (count == 0)//+1

{

tail = head;//+1

N\_op(2);//+1

}

else

{

current.Previous = New;//+2

N\_op(3);//+1

}

count++;//+2

N\_op(3);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = tail;//+1

N\_op(2);//+1

tail = tail.Previous;//+2

N\_op(3);//+1

head.Previous = current;//+2

N\_op(3);//+1

current.Next = head;//+2

N\_op(3);//+1

head = current;//+1

N\_op(2);//+1

tail.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

}

## public void SetObjLast(int data, int index) // дабавляем новый элемент списка в i-ю позицию начиная с конца

{

N\_op(3);//+1

Item current;//+6

N\_op(7);//+1

Item New = new Item(data);//+8

N\_op(9);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = tail;//+1

N\_op(2);//+1

tail = tail.Previous;//+2

N\_op(3);//+1

head.Previous = current;//+2

N\_op(3);//+1

current.Next = head;//+2

N\_op(3);//+1

head = current;//+1

N\_op(2);//+1

tail.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

current = tail;//+1

N\_op(2);//+1

New.Previous = current;//+2

N\_op(3);//+1

tail = New;//+1

N\_op(2);//+1

N\_op(2);//+1

if (count == 0)//+1

{

head = tail;//+1

N\_op(2);//+1

}

else

{

current.Next = New;//+2

N\_op(3);//+1

}

count++;//+2

N\_op(3);//+1

N\_op(4);//+1

for (int i = 0; i < index - 1; i++)

{

current = head;//+1

N\_op(2);//+1

head = head.Next;//+2

N\_op(3);//+1

tail.Next = current;//+2

N\_op(3);//+1

current.Previous = tail;//+2

N\_op(3);//+1

tail = current;//+1

N\_op(3);//+1

current.Next = null;//+2

N\_op(3);//+1

head.Previous = null;//+2

N\_op(3);//+1

N\_op(5);//+1

}

}

}

## public class Item // создаем элемент двусвязного списка

{

public int Data;

public Item Next { get; set; }

public Item Previous { get; set; }

public Item(int data)

{

Data = data;

}

}

## class Program

{

static void Main(string[] args)

{

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 1 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 300 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

for (int i = 0; i < 300; i++)

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 300 элементов

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 2 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 600 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

for (int i = 0; i < 600; i++)

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 600 элементов

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 3 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 900 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

for (int i = 0; i < 900; i++)

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 900 элементов

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 4 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 1200 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

for (int i = 0; i < 1200; i++)

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 1200 элементов

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 5 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 1500 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

for (int i = 0; i < 1500; i++)

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 1500 элементов

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 6 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 1800 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

for (int i = 0; i < 1800; i++)

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 1800 элементов

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 7 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 2100 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

for (int i = 0; i < 2100; i++)

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 2100 элементов

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 8 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 2400 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

for (int i = 0; i < 2400; i++)

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 2400 элементов

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 9 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 2700 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

for (int i = 0; i < 2700; i++)

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 2700 элементов

## {

Dequeue linkedList = new Dequeue();//+2

linkedList.N\_op(5);//+2

Console.Write("Номер сортировки: 10 ");//+2

linkedList.N\_op(4);//+2

Console.Write("Кол-во отсортированных элементов: 3000 ");//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(4);//+2

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

{

linkedList.AddLast(i);//+2

linkedList.N\_op(4);//+2

linkedList.AddFirst(i);//+2

linkedList.N\_op(4);//+2

linkedList.N\_op(5);//+2

}

linkedList.sort(linkedList, linkedList.Count);//+5

linkedList.N\_op(7);//+2

Console.Write("Время сортировки (такты): " + linkedList.ellapledTicks + " ");//+5

linkedList.N\_op(7);//+2

Console.WriteLine($"Колличество операций (N\_op): {linkedList.n\_op}");//+4

linkedList.N\_op(6);//+2

} // 3000 элементов

## Console.ReadLine();

}

}

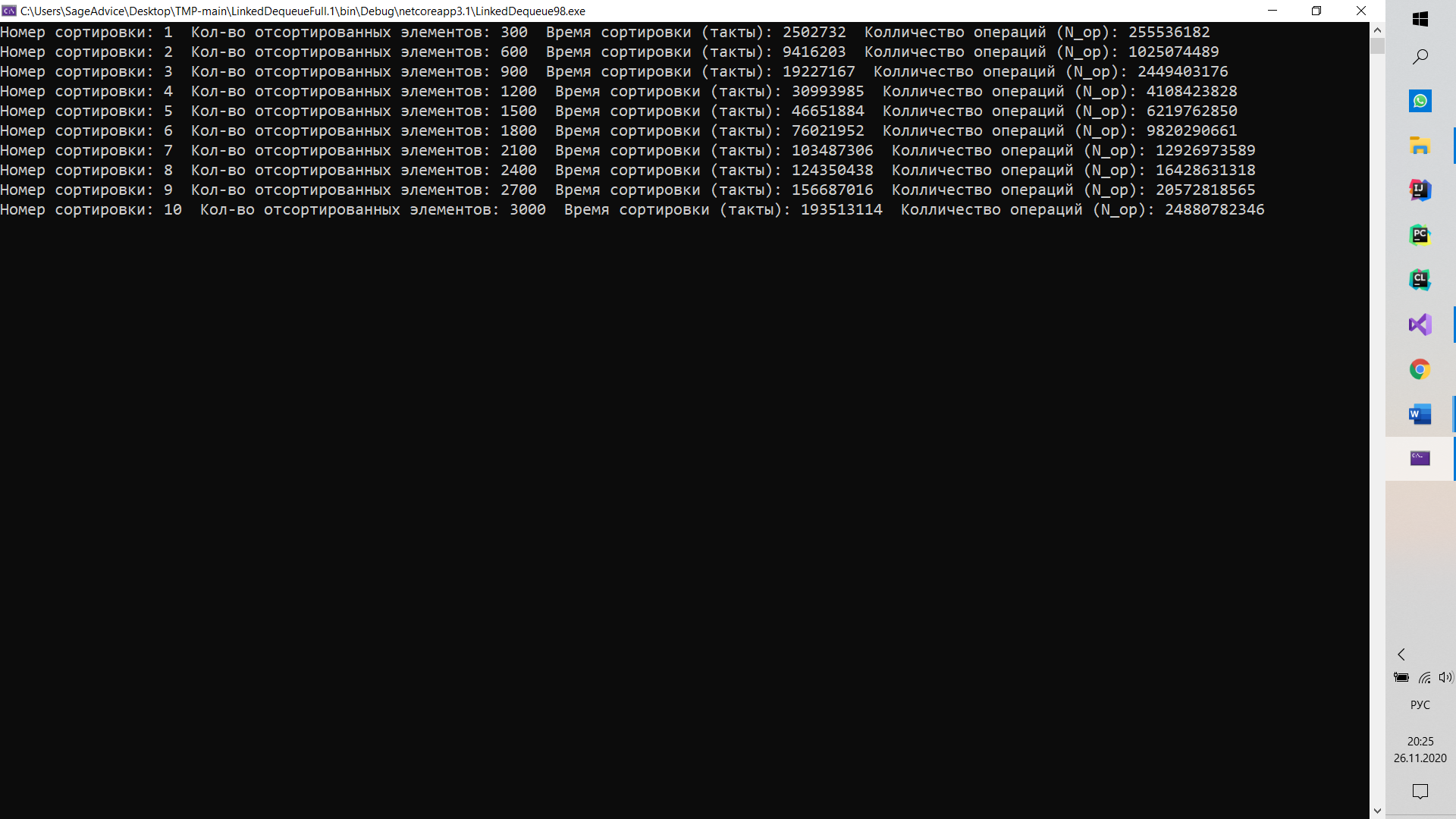
}

## **Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (такты) | N\_op |
| 300 | 44047142 | 740593 | 2502732 | 255536182 |
| 600 | 175615423 | 3322374 | 9416203 | 1025074489 |
| 900 | 394908711 | 7949162 | 19227167 | 2449403176 |
| 1200 | 702035547 | 14729498 | 30993985 | 4108423828 |
| 1500 | 1097071729 | 23739180 | 46651884 | 6219762850 |
| 1800 | 1580075700 | 35036651 | 76021952 | 9820290661 |
| 2100 | 2151095074 | 48669525 | 103487306 | 12926973589 |
| 2400 | 2810170044 | 64677995 | 124350438 | 16428631318 |
| 2700 | 3557335390 | 83096841 | 156687016 | 20572818565 |
| 3000 | 4392621770 | 103956721 | 193513114 | 24880782346 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 17,599 | 0,295 | 0,172 | 0,0024 |
| 18,650 | 0,352 | 0,171 | 0,0032 |
| 20,539 | 0,413 | 0,161 | 0,0032 |
| 22,650 | 0,475 | 0,170 | 0,0035 |
| 23,516 | 0,508 | 0,176 | 0,0038 |
| 20,784 | 0,460 | 0,160 | 0,0035 |
| 20,786 | 0,470 | 0,166 | 0,0037 |
| 22,598 | 0,520 | 0,171 | 0,0039 |
| 22,703 | 0,530 | 0,172 | 0,0040 |
| 22,699 | 0,537 | 0,176 | 0,0041 |

**Скриншот работы программы:**



**Вывод:**

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

**Источники:**

https://habr.com/ru/company/otus/blog/460087/