| 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 № 85**

Выполнил: студент 2 курса

группы БИСО-03-20

шифр 20Б0885

Турбина Виктория Александровна  
*(фио студента)*

Проверил:

*Филатов В.В.*

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 № 85.**

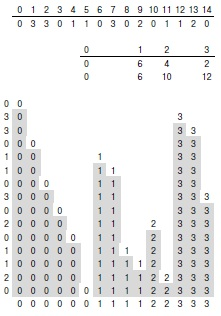
**Реализация связи элементов линейного списка: Массив**

**Способ организации линейного связанный список: Очередь**

**Алгоритм сортировки: Распределяющий подсчет**

**Теория о сортировках.**

**Сортировка распределяющим подсчетом.**

****

Метод распределяющего подсчета представляет собой сортировку с линейным временем выполнения при условии, что диапазон, в котором находятся значения ключей, превышает размер файла не более чем в постоянное количество раз.

Идея распределяющего подсчета состоит в том, чтобы подсчитать количество ключей с каждым конкретным значением, а затем использовать счетчики для перемещения в соответствующие позиции во время второго прохода по сортируемому файлу. Сначала подсчитывается число ключей для каждого значения, а затем вычисляются частичные суммы, эквивалентные количеству ключей, меньших или равных каждому такому значению. Далее, как и в случае двух значений ключа, эти числа используются как индексы при распределении ключей. Для каждого ключа величина связанного с ним счетчика рассматривается как индекс, указывающий на конец блока ключей с тем же значением. Этот индекс используется при размещении ключей в массиве b, после чего выполняется его сдвиг на одну позицию вправо.

**Листинг программы с расчетами.**

using System.Collections.Generic;

using System.IO;

using System;

class Queue

{

private List<int> queue;

public int N\_op = 0;

public Queue() \\1

{

queue = new List<int>(); \\1

}

public int Count() \\2

{

return queue.Count; \\2

}

// Добавляем в конец

public void Push(int newValue) \\3

{

N\_op++; \\2

queue.Add(newValue); \\1

}

// Извлекаем из начала

public int Pop() \\11

{

int result = queue[0]; N\_op += 2; \\4

queue.RemoveAt(0); N\_op++; \\4

N\_op++; \\2

return result; \\1

}

// Показываем значение первого элемента

public int Show() \\4

{

N\_op++; \\2

return queue[0]; \\2

}

// Получение элемента очереди по i-ой позиции

public int Get(int pos) \\16+12\*n

{

N\_op += 2; \\2

for (int i = 0; i < pos; ++i) \\1+(2+2+6+2)=12\*pos+1

{

N\_op++; \\2

Push(Pop()); \\5+1

N\_op += 4; \\2

}

N\_op++; \\2

int result = Show(); N\_op++; \\1+2+2

N\_op += 2; \\2

for (int i = pos; i < queue.Count; ++i) \\1+(2+10)=12\*n-12\*pos+1

{

N\_op++; \\2

Push(Pop()); \\5+1

N\_op += 4; \\2

}

N\_op++; \\2

return result; \\1

}

// Установка нового элемента на i-ую позицию

public void Set(int pos, int newValue) \\12\*n+10

{

N\_op += 2; \\2

for (int i = 0; i < pos; ++i) \\1+(2+10)=12\*pos+1

{

N\_op++; \\2

Push(Pop()); \\5+1

N\_op += 4; \\2

}

queue[0] = newValue; N\_op++; \\4

N\_op += 2; \\2

for (int i = pos; i < queue.Count; ++i) \\1+(2+10)=12\*n-12\*pos+1

{

N\_op++; \\2

Push(Pop()); \\5+1

N\_op += 4; \\2

}

}

// Отображение очереди в консоли

public void Print() \\6

{

N\_op += 2; \\2

foreach (int item in queue)

{

N\_op++; \\2

Console.WriteLine(item);

}

N\_op++; \\2

Console.WriteLine("---------");

}

public static Queue Sort(Queue tmp) \\12\*n^3+18\*n^2+20\*n+17

{

tmp.N\_op += 6; \\2

for (int i = 0; i < tmp.Count() + 1; i++) \\1+(2+4)=1+6\*n+6=6\*n+7

{

counter.Set(i, 0); \\2

tmp.N\_op += 5; \\2

}

for (int i = 0; i < tmp.Count(); i++) \\1+(2+5)=7\*n+1

{

counter.Set(tmp.Get(i), counter.Get(tmp.Get(i))+1); \\3

tmp.N\_op += 15; \\2

}

int x = 0; \\1

for (int i = 0; i < tmp.Count() + 1; i++) \\1+(2+12\*n^2+18\*n+5)=12\*n^3+18\*n^2+7\*n+1

{

if (counter.Get(i)!= 0) \\12\*n^2+18\*n+5

{

tmp.N\_op += 1; \\2

for (int j = 0; j < counter.Get(i); j++) \\1+(2+12\*n+16)=12\*n^2+18\*n+1

{

tmp.Set(x, i); \\12\*n+10

x++; tmp.N\_op++; \\4

tmp.N\_op += 15; \\2

}

}

}

return tmp; \\1

}

}

class Program

{

static void Main()

{

Queue tmp = new Queue();

long t\_s; long t\_f;

int[] p = new int[3000];

int k = 300;

var random = new Random();

//Заполнение массива случайными числами

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

{

p[i] = random.Next(1, 300);

}

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

{

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

{

tmp.Push(p[z]);

}

tmp.N\_op = 0;

t\_s = DateTime.Now.Ticks;

tmp = Queue.Sort(tmp);

t\_f = DateTime.Now.Ticks;

Console.WriteLine("Номер сортировки: {0} Колличество отсортированных элементов: {1} Время сортировки(ms): {2} Количество операций: {3}", i + 1, k, t\_f - t\_s, tmp.N\_op);

//tmp.Print();

k += 300;

Console.ReadLine();

}

}

}

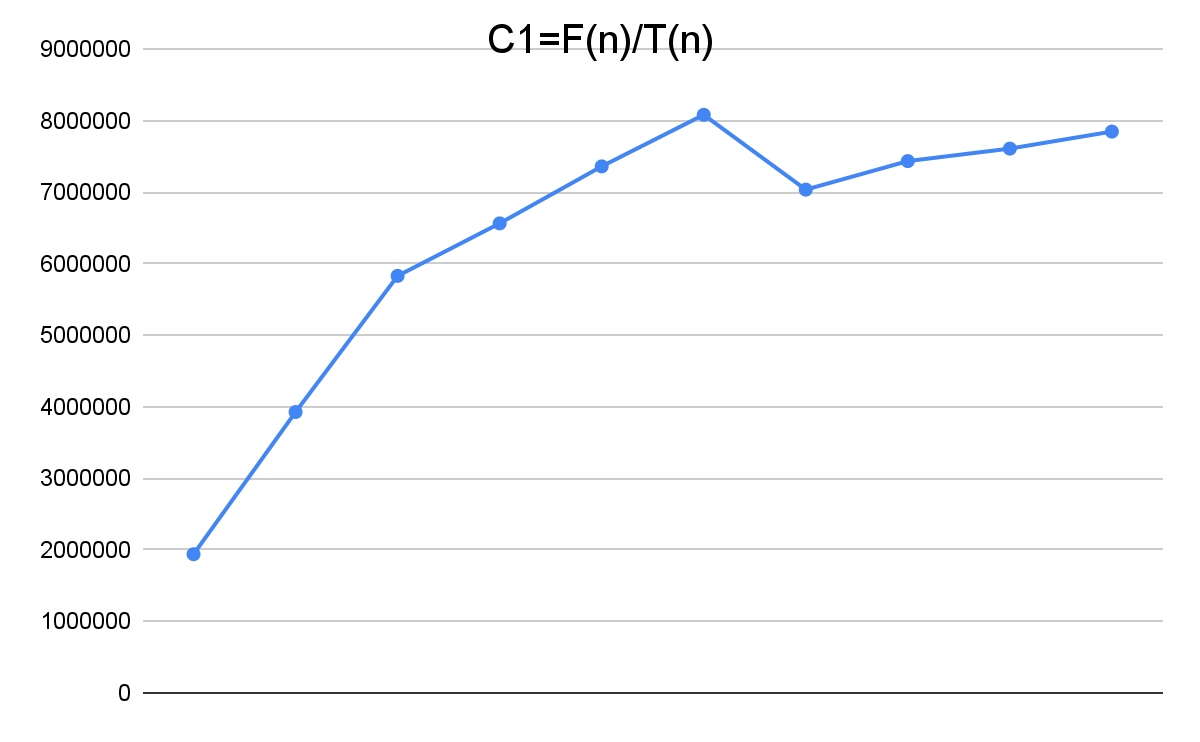
F(n) = 12\*n^3+18\*n^2+44n+70

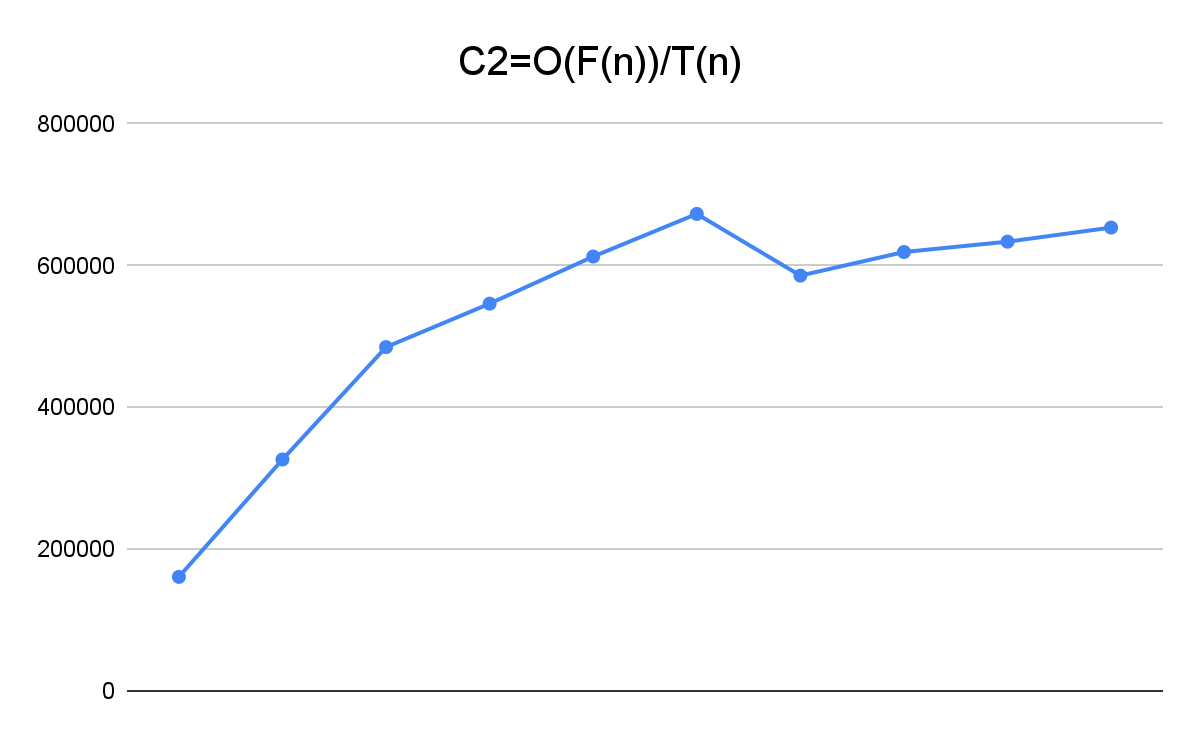
O(F(n)) = n^3

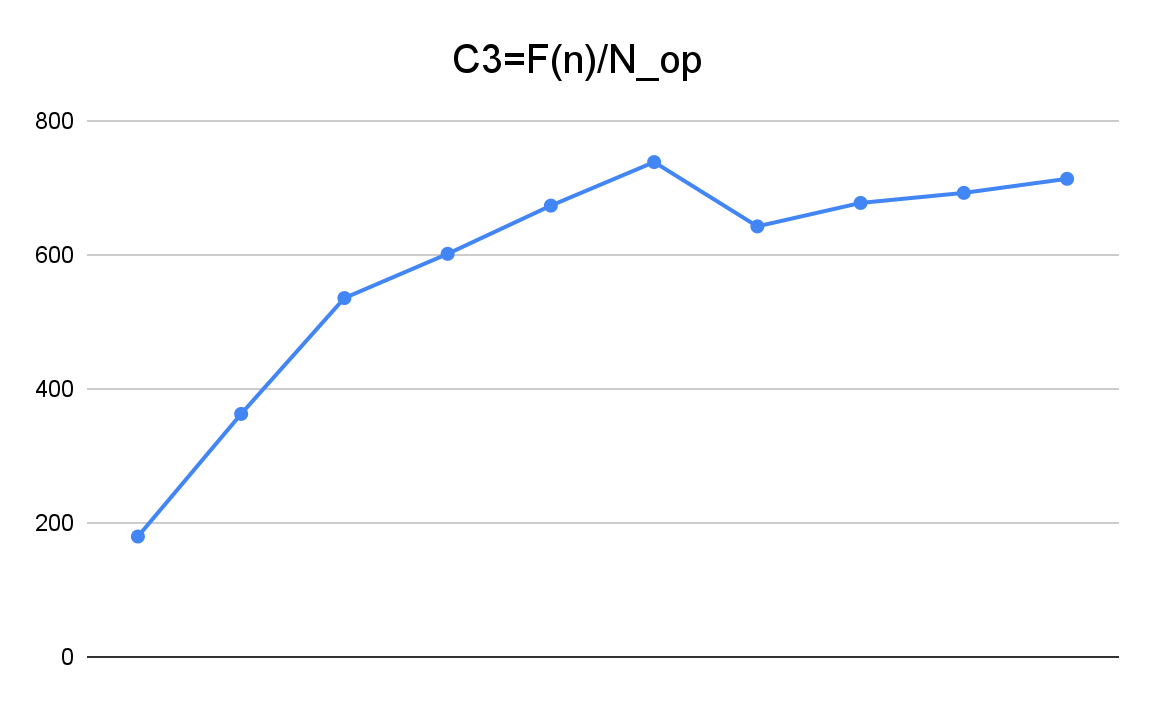
**Таблица результата экспериментов и графики зависимостей**

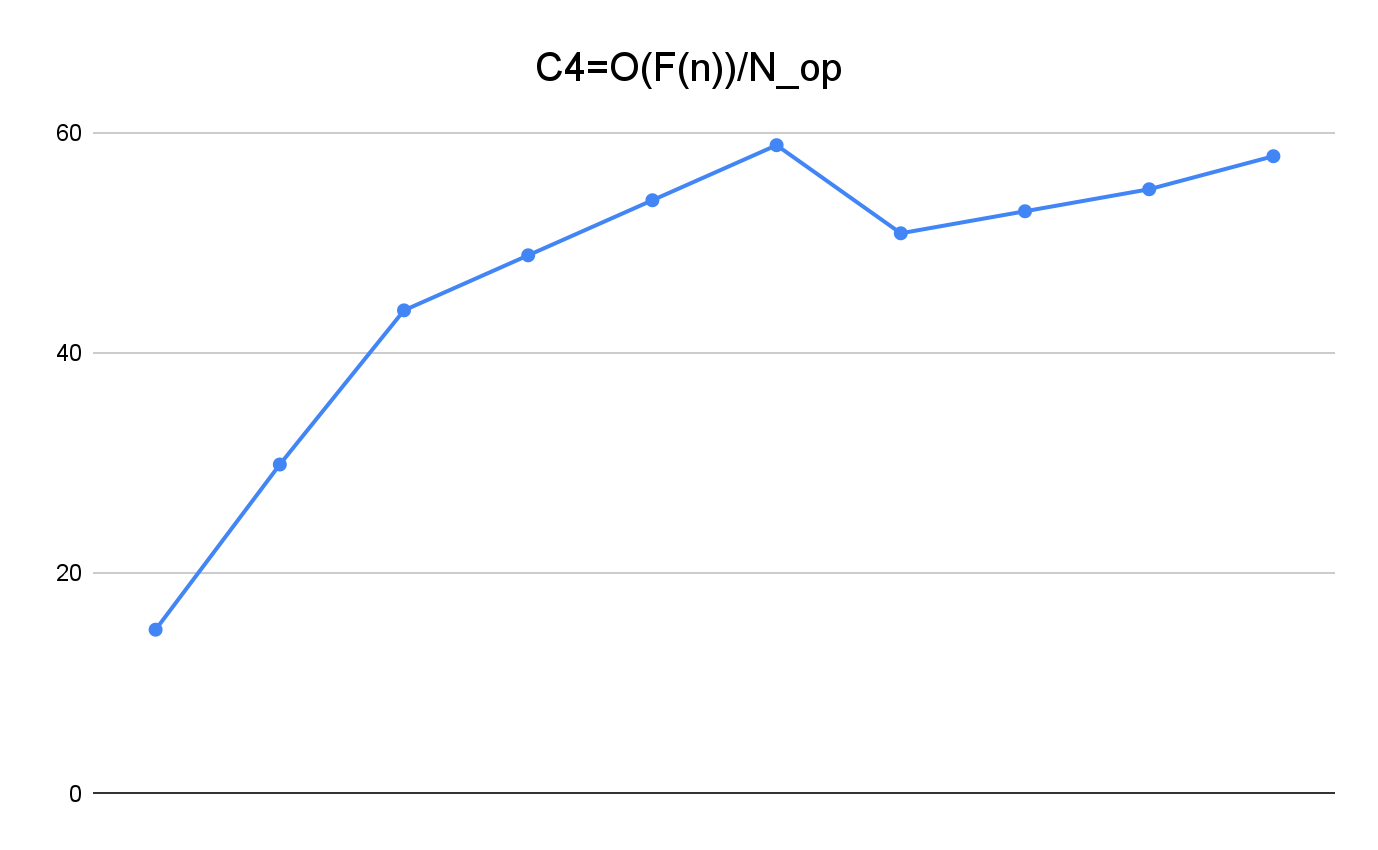
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| --- | --- | --- | --- | --- |
| 300 | 325633270 | 27000000 | 167,8628 | 1814911 |
| 600 | 2598506470 | 216000000 | 661,6980 | 7229669 |
| 900 | 8762619670 | 729000000 | 1503,043 | 16244390 |
| 1200 | 20761972870 | 1728000000 | 3163,027 | 28859101 |
| 1500 | 40540566070 | 3375000000 | 5507,745 | 45073807 |
| 1800 | 70042399270 | 5832000000 | 8666,908 | 64888508 |
| 2100 | 111211472470 | 9261000000 | 15806,727 | 88303210 |
| 2400 | 165991785670 | 13824000000 | 22326,839 | 115317910 |
| 2700 | 236327338870 | 19683000000 | 31057,977 | 145932610 |
| 3000 | 324162132070 | 27000000000 | 41309,337 | 180147310 |

| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| --- | --- | --- | --- |
| 1939877,510 | 160845,643 | 179,421 | 14,877 |
| 3927027,844 | 326432,904 | 362,423 | 29,877 |
| 5829919,483 | 485016,064 | 535,424 | 43,877 |
| 6563956,890 | 546312,124 | 601,425 | 48,877 |
| 7360646,884 | 612773,467 | 673,426 | 53,877 |
| 8081590,259 | 672904,339 | 738,427 | 58,877 |
| 7035705,271 | 585889,792 | 642,427 | 50,877 |
| 7434629,939 | 619165,122 | 677,428 | 52,877 |
| 7609231,563 | 633750,228 | 692,428 | 54,877 |
| 7847187,963 | 653605,261 | 713,428 | 57,877 |

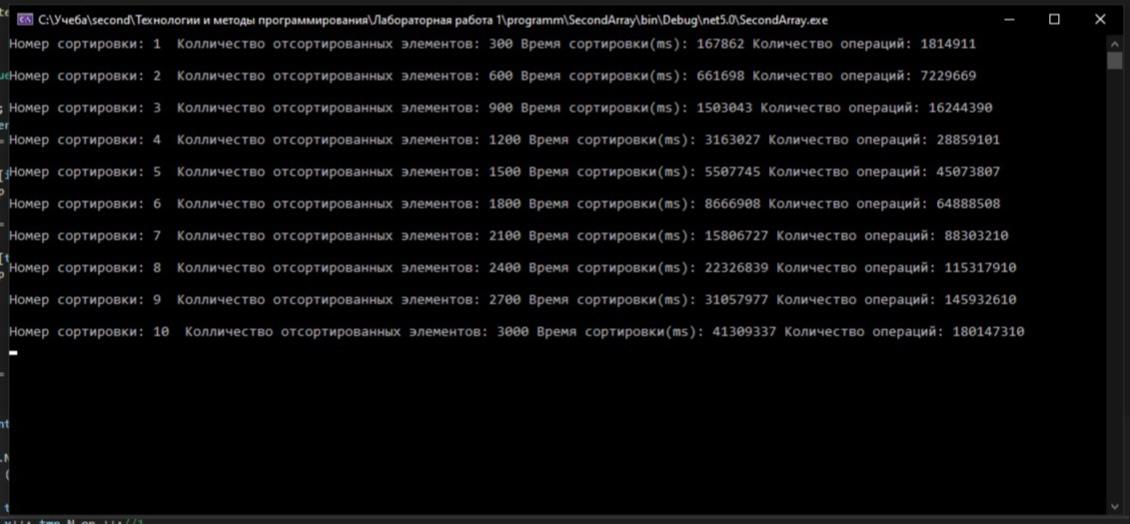








**Скриншот работы программы:**

****

**Выводы.**

По результатам проверки удалось установить, что графики C1, C2, C3, C4 от N на следующем шаге станут константными значениями. Следовательно, работа по выявлению экспериментальных значений трудоемкости функции сортировки линейной структуры была проведена и реализована. Резкое падение на графиках зависимостей появилось из-за рандомного подбора элементов очереди, и при увеличении числа элементов соответствующее соотношение приближается к желаемому результату. После написания алгоритма сортировки была посчитана сложность алгоритма - она равно .

**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.