|  |
| --- |
| 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 № 39**

Выполнил: студент 2 курса

группы БСБО-01-21

шифр 21Б0839

Дроздов Владислав Николаевич  
*(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 № 39.**

**Реализация связи элементов линейного списка:** Библиотека классов

**Способ организации линейного связанный список:** Очередь

**Алгоритм сортировки:** Быстрая сортировка Хоару (с медианным (pivot) элементом)

**Теория о сортировках**

**Быстрая сортировка (метод Хоару)**

Этот алгоритм, чаще называемый просто «быстрая сортировка» (англ. Quicksort), придуман английским ученым Чарльзом Хоаром в 1960 году и использует стратегию **«разделяй и властвуй»**.

Шаги алгоритма таковы:

1. Выбираем в массиве некоторый элемент, который будем называть *опорным элементом*.  
2. Операция разделения массива: реорганизуем массив таким образом, чтобы все элементы, меньшие или равные опорному элементу, оказались слева от него, а все элементы, большие опорного — справа от него. Обычный алгоритм этой операции:

* Два индекса — l и r, приравниваются к минимальному и максимальному индексу разделяемого массива соответственно.
* Вычисляется индекс опорного элемента m.
* Индекс l последовательно увеличивается до m до тех пор, пока l-й элемент не превысит опорный.
* Индекс r последовательно уменьшается до m до тех пор, пока r-й элемент не окажется меньше либо равен опорному.
* Если r = l — найдена середина массива — операция разделения закончена, оба индекса указывают на опорный элемент.
* Если l <r — найденную пару элементов нужно обменять местами и продолжить операцию разделения с тех значений l и r, которые были достигнуты. Следует учесть, что если какая-либо граница (l или r) дошла до опорного элемента, то при обмене значение m изменяется на r-й или l-й элемент соответственно.

3. Рекурсивно упорядочиваем подмассивы, лежащие слева и справа от опорного элемента.  
4. Базой рекурсии являются наборы, состоящие из одного или двух элементов. Первый возвращается в исходном виде, во втором, при необходимости, сортировка сводится к перестановке двух элементов. Все такие отрезки уже упорядочены в процессе разделения.

Поскольку в каждой итерации (на каждом следующем уровне рекурсии) длина обрабатываемого отрезка массива уменьшается, по меньшей мере, на единицу, терминальная ветвь рекурсии будет достигнута всегда, и обработка гарантированно завершится.

**Листинг программы с расчетами.**

**Program.cs**

using Linear\_lists;

namespace Lab\_1

{

class Program

{

static void Main()

{

Random random = new Random();

NAQ<int> students = new NAQ<int>();

int[] tests = new int[10] { 300, 600, 900, 1200, 1500, 1800, 2100, 2400, 2700, 3000 };

for (int i = 0; i < tests.Length; i++)

{

for (int j = 0; j < tests[i]; j++)

{

students.EnQueue(random.Next(10));

}

Sort.Hoar(students);

Console.WriteLine($"Опыт {i+1}:\n \*\*\* Кол-во отсортированных элементов - {tests[i]}\n \*\*\* Время сортировки - {(double)Sort.timer.ElapsedTicks/Sort.freq\*1000} мc\n \*\*\* Кол-во операций - {NAQ<int>.N\_op + Sort.N\_q}\n");

students.ClearAll();

Sort.N\_q = 0;

NAQ<int>.N\_op = 0;

}

}

}

}

**NAQ.cs**

using System;

namespace Linear\_lists

{

public class NAQ<T>

{

private List<T> queue;

public int Count { get; private set; }

static public int N\_op = 0;

public NAQ()

{

queue = new List<T>(); //1

N\_op++;

}

public void ClearAll()

{

queue.Clear();

Count = 0; //1

N\_op += 3;

}

public bool ContainsAtQueue(T key)

{

N\_op += 2;

foreach (T item in queue) //Count итераций - max

{

N\_op += 6;

if (key.Equals(item)) // 3

{

N\_op += 4;

return true; // 1

}

}

N\_op ++;

return false; // 1

}

public void EnQueue(T item)

{

N\_op += 4;

queue.Add(item); // 3

Count++; // 1

}

public T DeQueue()

{

T c = queue[0]; // 1

queue.Remove(c); // 3

Count--; // 1

N\_op += 6;

return c; // 1

}

public void DeQueueAll()

{

int count = Count; // 1

N\_op += 3;

for (int i = 0; i < count; i++) // Count итераций

{

N\_op += 5;

Console.Write(DeQueue() + " "); // 3

}

N\_op += 2;

Console.WriteLine(); // 2

}

public T[] CopyToArray()

{

int count = Count; // 1

T[] array = new T[count]; // 1

N\_op += 4;

for (int i = 0; i < count; i++) // Count итераций

{

N\_op += 5;

array[i] = queue[i]; // 3

}

N\_op ++;

return array; // 1

}

public void CopyFromArray(T[] array)

{

N\_op += 4;

for (int i = 0; i < array.Length; i++) // Count итераций

{

N\_op += 7;

queue[i] = array[i]; // 3

}

Count = array.Length; // 3

N\_op += 3;

}

}

}

**Sort.cs**

using Linear\_lists;

using System.Diagnostics;

namespace Lab\_1

{

class Sort

{

static public int N\_q;

static public Stopwatch timer = new Stopwatch();

static public long freq = Stopwatch.Frequency;

public static void Hoar(NAQ<int> nq)

{

timer.Start();

int[] naq = nq.CopyToArray(); // 3

Quicksort(naq, 0, naq.Length - 1); // 5

nq.CopyFromArray(naq); // 4

N\_q = N\_q + 11;

}

private static int Partition(int[] arr, int start, int end)

{

int key = start; // 1

N\_q++;

N\_q = N\_q + 2;

for (int i = start; i <= end; i++) // end - start + 1 итераций

{

N\_q += 2;

if (arr[i] <= arr[end]) // 3

{

N\_q += 11;

int temp = arr[key]; // 2

arr[key] = arr[i]; // 3

arr[i] = temp; // 2

key++; // 1

}

}

N\_q += 5;

return key - 1; // 2

}

private static void Quicksort(int[] array, int start, int end)

{

N\_q += 1;

if (start >= end) // 2 в любом случае

{

N\_q += 2;

return;

}

int pivot = Partition(array, start, end); // 6

Quicksort(array, start, pivot-1); // 6

Quicksort(array, pivot + 1, end); // 6

N\_q += 18;

timer.Stop();

}

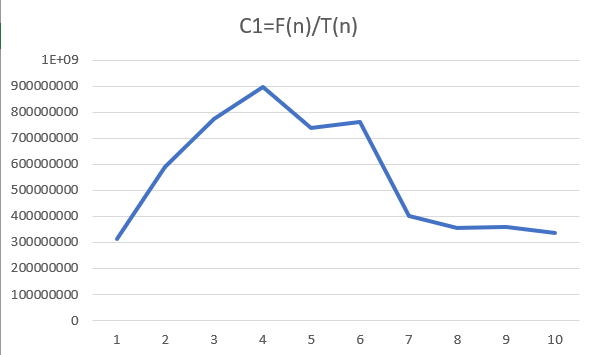
}

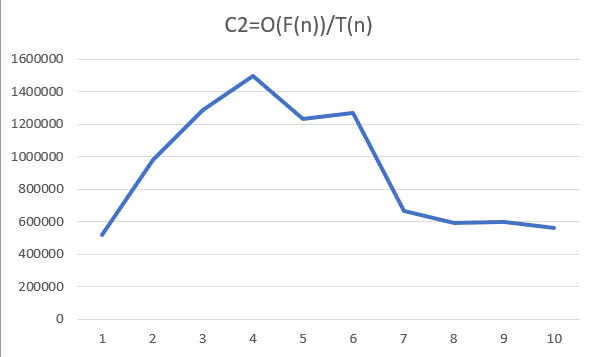
}

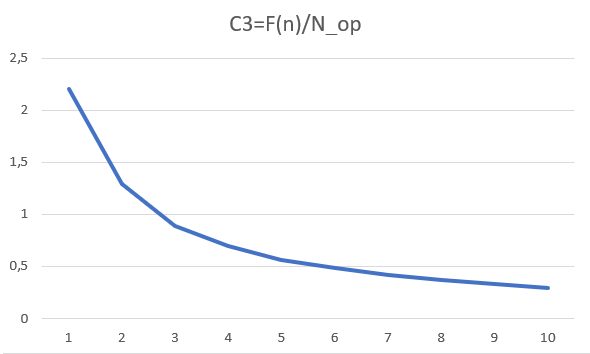
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 | 180817 | 300 | 0,0005784 | 82103 |
| 600 | 360817 | 600 | 0,0006136 | 279288 |
| 900 | 540817 | 900 | 0,0006991 | 605987 |
| 1200 | 720817 | 1200 | 0,000802 | 1030165 |
| 1500 | 900817 | 1500 | 0,0012173 | 1609680 |
| 1800 | 1080817 | 1800 | 0,001417 | 2246965 |
| 2100 | 1260817 | 2100 | 0,0031394 | 3037480 |
| 2400 | 1440817 | 2400 | 0,0040586 | 3937087 |
| 2700 | 1620817 | 2700 | 0,0045317 | 4950089 |
| 3000 | 1800817 | 3000 | 0,0053293 | 6195561 |

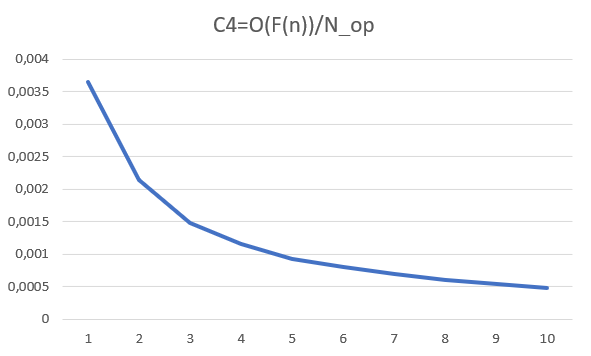
**Таблица результата экспериментов и графики зависимости**

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 312615837 | 518672,1992 | 2,202319038 | 0,003653947 |
| 588032920 | 977835,7236 | 1,291917304 | 0,00214832 |
| 773590330 | 1287369,475 | 0,892456439 | 0,00148518 |
| 898774314 | 1496259,352 | 0,699710241 | 0,001164862 |
| 740012322 | 1232235,275 | 0,559624894 | 0,000931862 |
| 762750176 | 1270289,344 | 0,481011943 | 0,000801081 |
| 401610817 | 668917,6276 | 0,415086519 | 0,000691363 |
| 355003449 | 591336,9142 | 0,365960163 | 0,000609588 |
| 357662025 | 595802,8996 | 0,327431891 | 0,000545445 |
| 337908731 | 562925,7126 | 0,29066246 | 0,000484218 |

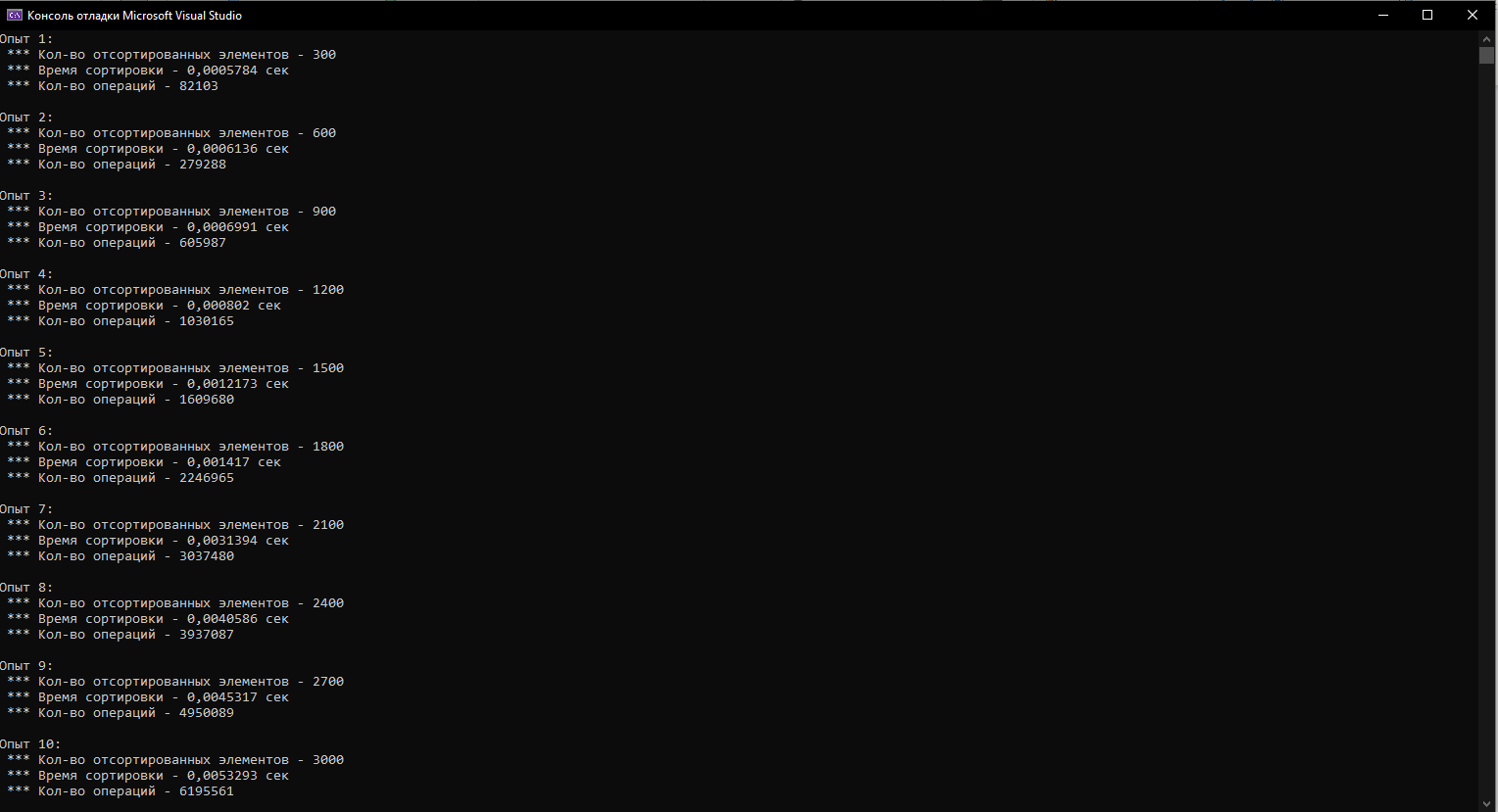








**Скриншот работы программы**



**Выводы**

По результатам экспериментов было установлено, что графики С1 и С2, С3 и С4 попарно линейно зависимы от количество обрабатываемых элементов.

**Литература:**

1. Структуры данных и алгоритмы. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. – М.: Издательский дом «Вильямс», 2000
2. Алгоритмы. Самый краткий и понятный курс / Панос Луридас ; [перевод с английского М. А. Райтмана]. — Москва: Эксмо, 2022. — 192 с. — (Библиотека MIT).
3. Быстрая сортировка / Хабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/company/otus/blog/524948/ (Дата обращения 17.10.2022).

**Приложение 1. Программный код Program.cs**

using Linear\_lists;

namespace Lab\_1

{

class Program

{

static void Main()

{

Random random = new Random();

NAQ<int> students = new NAQ<int>();

int[] tests = new int[10] { 300, 600, 900, 1200, 1500, 1800, 2100, 2400, 2700, 3000 };

for (int i = 0; i < tests.Length; i++)

{

for (int j = 0; j < tests[i]; j++)

{

students.EnQueue(random.Next(10));

}

Sort.Hoar(students);

Console.WriteLine($"Опыт {i+1}:\n \*\*\* Кол-во отсортированных элементов - {tests[i]}\n \*\*\* Время сортировки - {(double)Sort.timer.ElapsedTicks/Sort.freq} cек\n \*\*\* Кол-во операций - {NAQ<int>.N\_op + Sort.N\_q}\n");

students.ClearAll();

Sort.N\_q = 0;

NAQ<int>.N\_op = 0;

}

}

}

}

**Приложение 2. Программный код NAQ.cs**

using System;

namespace Linear\_lists

{

public class NAQ<T>

{

private List<T> queue;

public int Count { get; private set; }

static public int N\_op = 0;

public NAQ()

{

queue = new List<T>(); //1

N\_op++;

}

public void ClearAll()

{

queue.Clear();

Count = 0; //1

N\_op += 3;

}

public bool ContainsAtQueue(T key)

{

N\_op += 2;

foreach (T item in queue) //Count итераций - max

{

N\_op += 6;

if (key.Equals(item)) // 3

{

N\_op += 4;

return true; // 1

}

}

N\_op ++;

return false; // 1

}

public void EnQueue(T item)

{

N\_op += 4;

queue.Add(item); // 3

Count++; // 1

}

public T DeQueue()

{

T c = queue[0]; // 1

queue.Remove(c); // 3

Count--; // 1

N\_op += 6;

return c; // 1

}

public void DeQueueAll()

{

int count = Count; // 1

N\_op += 3;

for (int i = 0; i < count; i++) // Count итераций

{

N\_op += 5;

Console.Write(DeQueue() + " "); // 3

}

N\_op += 2;

Console.WriteLine(); // 2

}

public T[] CopyToArray()

{

int count = Count; // 1

T[] array = new T[count]; // 1

N\_op += 4;

for (int i = 0; i < count; i++) // Count итераций

{

N\_op += 5;

array[i] = queue[i]; // 3

}

N\_op ++;

return array; // 1

}

public void CopyFromArray(T[] array)

{

N\_op += 4;

for (int i = 0; i < array.Length; i++) // Count итераций

{

N\_op += 7;

queue[i] = array[i]; // 3

}

Count = array.Length; // 3

N\_op += 3;

}

}

}

**Приложение 3. Программный код Sort.cs**

using Linear\_lists;

using System.Diagnostics;

namespace Lab\_1

{

class Sort

{

static public int N\_q;

static public Stopwatch timer = new Stopwatch();

static public long freq = Stopwatch.Frequency;

public static void Hoar(NAQ<int> nq)

{

timer.Start();

int[] naq = nq.CopyToArray(); // 3

Quicksort(naq, 0, naq.Length - 1); // 5

nq.CopyFromArray(naq); // 4

N\_q = N\_q + 11;

}

private static int Partition(int[] arr, int start, int end)

{

int key = start; // 1

N\_q++;

N\_q = N\_q + 2;

for (int i = start; i <= end; i++) // end - start + 1 итераций

{

N\_q += 2;

if (arr[i] <= arr[end]) // 3

{

N\_q += 11;

int temp = arr[key]; // 2

arr[key] = arr[i]; // 3

arr[i] = temp; // 2

key++; // 1

}

}

N\_q += 5;

return key - 1; // 2

}

private static void Quicksort(int[] array, int start, int end)

{

N\_q += 1;

if (start >= end) // 2 в любом случае

{

N\_q += 2;

return;

}

int pivot = Partition(array, start, end); // 6

Quicksort(array, start, pivot-1); // 6

Quicksort(array, pivot + 1, end); // 6

N\_q += 18;

timer.Stop();

}

}

}