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 № 96

Выполнил: студент 2 курса
группы БСБО-11-21
шифр 21Б0296
Хохлов Дмитрий Владимирович
(фио студента)

Проверил:
Леонид Олегович Головин

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 № 96.

Реализация связи элементов линейного списка: Библиотека классов

Способ организации линейного связанный список: Очередь

Алгоритм сортировки: Фиксированное двухпутевое слияние

Теория о сортировках.

Сортировка простым (фиксированным) двухпутевым слиянием.
В алгоритме Е границы между отрезками полностью определяются "ступеньками вниз". Такой подход обладает тем возможным преимуществом, что исходные файлы с преобладанием возрастающего или *убывающего* расположения элементов могут обрабатываться очень быстро, но при этом замедляется основной цикл вычислений. Вместо проверок ступенек вниз можно принудительно установить длину отрезков, считая, что все отрезки исходного файла имеют длину 1, после первого просмотра все отрезки (кроме, быть может, последнего) имеют длину 2, …, после $k^{го}$ просмотра имеют длину 2k. В отличие от "естественного" слияния в алгоритме Е такой способ называется *простым* (или фиксированным) двухпутевым слиянием.

Сортировка простым двухпутевым слиянием

503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	703	512	677	509	908	426	897	653	275	170	154	612	061	765	087
087	503	703	765	154	170	509	908	897	653	426	275	677	612	512	061
061	087	503	512	612	677	703	765	908	897	653	509	426	275	170	154
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

Пример работы алгоритма Ф приведен на рисунке. Этот метод справедлив и тогда, когда N не является степенью 2, сливаемые отрезки не все имеют длину 2k. Проверки ступенек заменены уменьшением длины переменных q и r и проверкой на равенство нулю. Благодаря этому время асимптотически приближается к $11N\log_{2N}$, что несколько лучше, чем в предыдущем алгоритме.

На практике имеет смысл комбинировать алгоритм Ф с простыми вставками; вместо первых четырех просмотров алгоритма Ф можно простыми вставками отсортировать группы, скажем из 16 элементов, исключив тем самым довольно расточительные вспомогательные операции, связанные со слиянием коротких файлов.

Листинг программы с расчетами.

```
class Queue {
    private _Array: number[][]
    public _F_lim: number
    public _O_F_lim: number
    public _T_lim: number
    public _N_op: number

    constructor(quantity: number) {
        this._Array = []
        this._F_lim = Math.floor(1+( 11+Math.log2(quantity)*16)*(quantity-1))
        this._O_F_lim = Math.floor(Math.log2(quantity)*quantity*) // *=8,62 => 1
        this._T_lim = 0
        this._N_op = 0
    }
    Push(num: number[]){//1
        this._Array.push(num)
    }
    Shift(){//1
        return this._Array.shift()!
    }
    First(){//1
        return this._Array.at(0)!
    }
    Length(){//1
        return this._Array.length
    }
    Merger() {
        const timer = Date.now()
        while (this.Length() != 1){ //2 + ∑_1^(quantity−1)(2 + 2 + 4 + 1 + 1 + 1 +
//log2(quantity) ∗ 16) = 1 + ∑_1^(quantity−1)(11 + log2(quantity) ∗ 16) = 1 + ( 11 + log2(quantity) ∗
//16) ∗ (quantity − 1)
            const A1 = this.Shift()//2

            const A2 = this.Shift()//2

            const limit = A1.length + A2.length//4

            let retA :number[] = []//1

            for (let i = 0; i < limit; i++)// 1 + log2(quantity) ∗ quantity ∗
//(3 + 5 + 2 + 2 + 2 + 2) = 1 + log2(quantity) ∗ quantity ∗ 16
                if(A1.length > 0 && A2.length > 0)//3
```

```
                    A1 > A2 ? retA.push( A2.shift()! ) : retA.push( A1.shift()!)//5

                else if(A2.length === 0)//2
                    retA.push( A1.shift()! )//2

                else if(A1.length === 0)//2
                    retA.push( A2.shift()! )//2

            this.Push( retA.flat())//2
        }
        this._T_lim = Number((Date.now()-timer).toFixed(3))
    }
}

//for (let i = 300; i < 9001; i+=300) {
    let Quantity = 300//i
    let queue = new Queue(Quantity)

    for (let i = 0; i < Quantity; i++)
        queue.Push([Math.random()])

    queue.Merger()
    console.log('Сортировка №' + Quantity/300)
    console.log('F(n)= ' +queue._F_lim)
    console.log('O(F(n))= ' +queue._O_F_lim)
    console.log('T(n)= ' +queue._T_lim)
    console.log('N_op= ' +queue._N_op)
    console.log()
    //console.log(queue)
    // console.log("C1 = " + queue._F_lim / queue._T_lim)
    // console.log("C2 = " + queue._O_F_lim / queue._T_lim)
    // console.log("C3 = " + queue._F_lim / queue._N_op)
    // console.log("C4 = " + queue._O_F_lim / queue._N_op)
    console.log()
//}
```

$F(n) = 1 + (11 + \log2(n) * 16)*(n-1)$

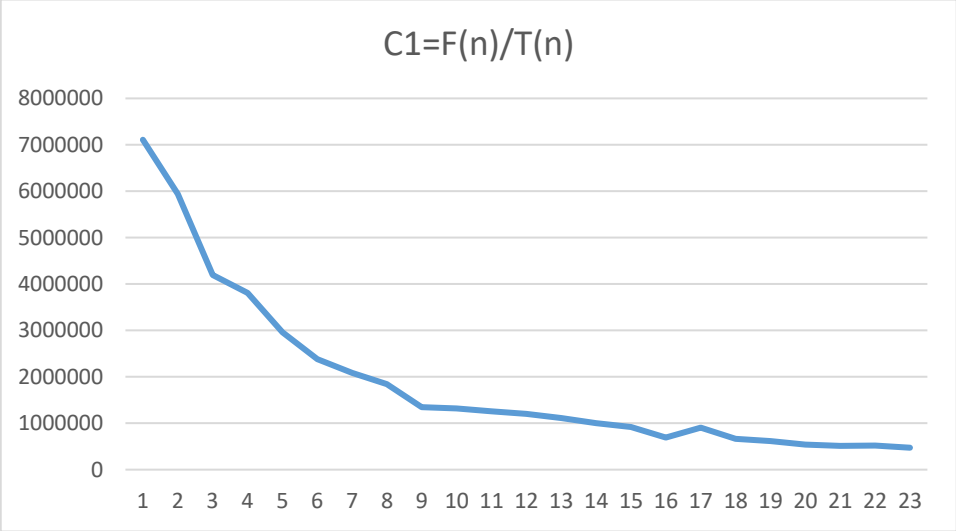
$O(F(n)) = \log2(n) * n$

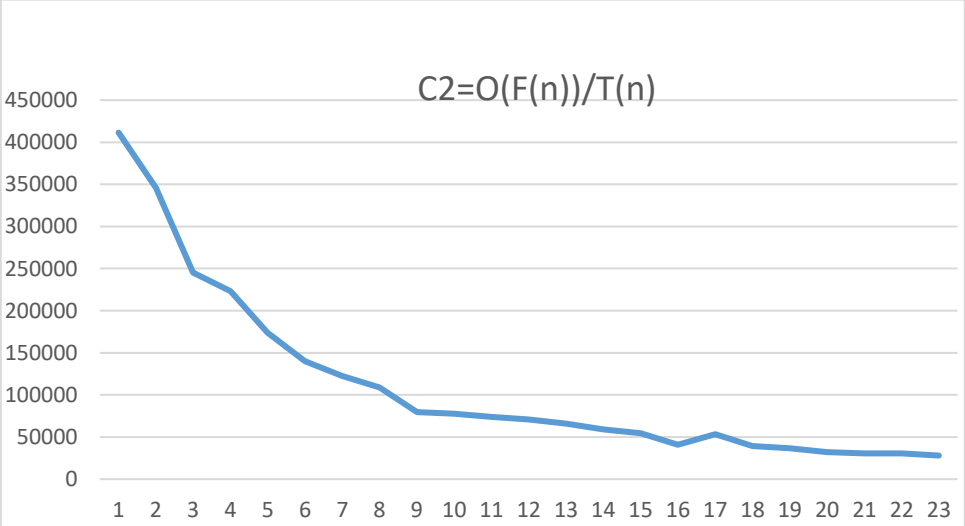
Таблица результата экспериментов и графики зависимостей

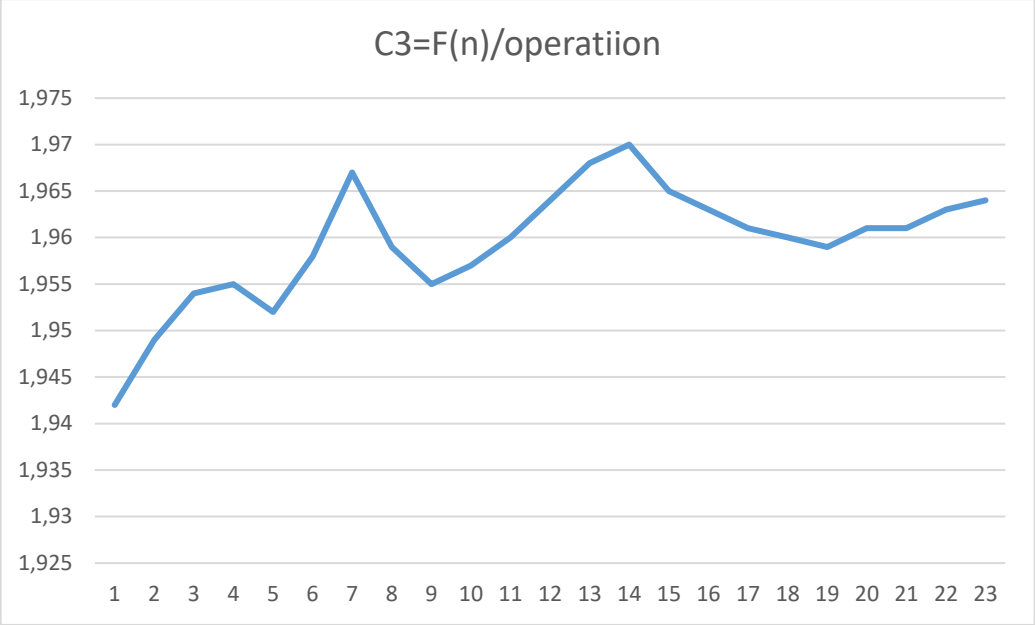
Кол-во элементов	F(n)	O(F(n))	T(n)(c)	operatiion
300	42656,669	2468,646	0,006	21966
600	95038,998	5537,291	0,016	48758
900	151051,429	8832,403	0,036	77310
1200	209419,658	12274,582	0,055	107094
1500	269539,111	15826,12	0,091	138086
1800	331053,878	19464,806	0,139	169086
2100	393728,855	23175,965	0,189	200166
2400	457396,977	26949,165	0,248	233538
2700	521933,348	30776,608	0,387	266910
3000	587241,034	34652,24	0,446	300078
3300	653242,604	38571,226	0,521	333318
3600	719874,776	42529,612	0,6	366598
3900	787084,857	46524,108	0,708	399998
4200	854828,288	50551,929	0,856	433882
4500	923066,897	54610,692	1,003	469714
4800	991767,614	58698,33	1,434	505246
5100	1060901,512	62813,036	1,174	541018
5400	1130443,075	66953,216	1,707	576710
5700	1200369,626	71117,453	1,941	612658
6000	1270660,879	75304,481	2,348	648034
6300	1341298,582	79513,158	2,609	683846
6600	1412266,221	83742,452	2,729	719422
6900	1483548,785	87991,426	3,141	755406

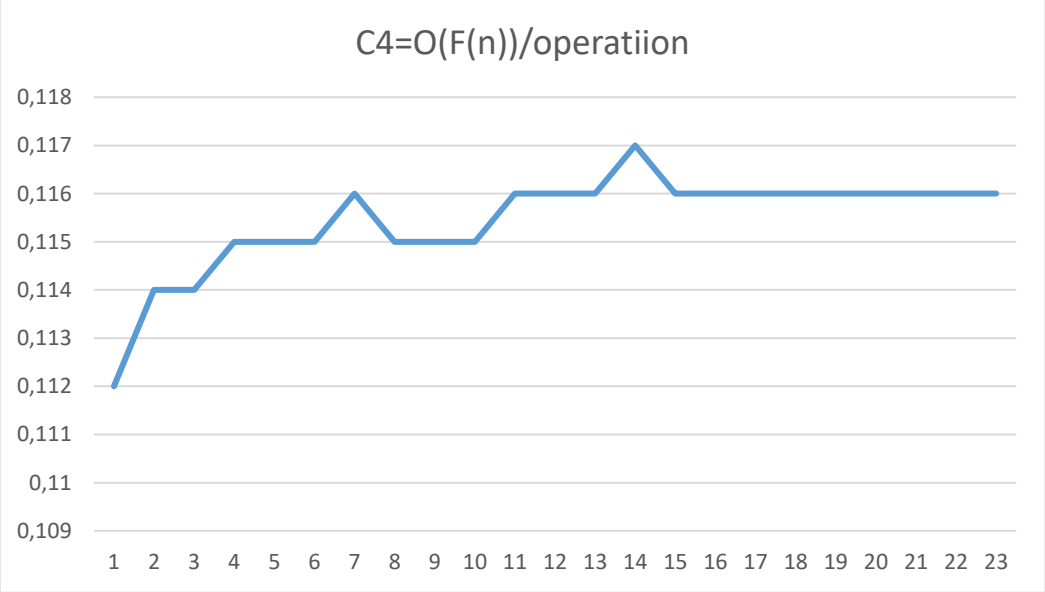
C1=F(n)/T(n)	C2=O(F(n))/T(n)	C3=F(n)/operatiion	C4=O(F(n))/operatiion
7109444,833	411441	1,942	0,112
5939937,375	346080,688	1,949	0,114
4195873,028	245344,528	1,954	0,114
3807630,145	223174,218	1,955	0,115
2961968,253	173913,407	1,952	0,115
2381682,576	140034,576	1,958	0,115
2083221,455	122624,153	1,967	0,116
1844342,649	108665,988	1,959	0,115
1348664,982	79526,119	1,955	0,115
1316683,933	77695,605	1,957	0,115
1253824,576	74033,063	1,96	0,116

C1=F(n)/T(n)	C2=O(F(n))/T(n)	C3=F(n)/operatiion	C4=O(F(n))/operatiion
1199791,293	70882,687	1,964	0,116
1111701,775	65712,017	1,968	0,116
998631,178	59055,992	1,97	0,117
920305,979	54447,35	1,965	0,116
691609,215	40933,285	1,963	0,116
903663,98	53503,438	1,961	0,116
662239,646	39222,739	1,96	0,116
618428,452	36639,595	1,959	0,116
541167,325	32071,755	1,961	0,116
514104,478	30476,488	1,961	0,116
517503,196	30686,131	1,963	0,116
472317,346	28013,826	1,964	0,116






C3=F(n)/operatiion
1,975
1,97
1,965
1,96
1,955
1,95
1,945
1,94
1,935
1,93
1,925
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23


C4=O(F(n))/operatiion
0,118
0,117
0,116
0,115
0,114
0,113
0,112
0,111
0,11
0,109
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

```
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node dist/labaNop.js`
Сортировка №1
F(n)= 42656
O(F(n))= 2468
T(n)= 6
N_op= 21966


Сортировка №2
F(n)= 95038
O(F(n))= 5537
T(n)= 16
N_op= 48758


Сортировка №3
F(n)= 151051
O(F(n))= 8832
T(n)= 36
N_op= 77310


Сортировка №4
F(n)= 209419
O(F(n))= 12274
T(n)= 55
N_op= 107094


Сортировка №5
F(n)= 269539
O(F(n))= 15826
T(n)= 91
N_op= 138086


Сортировка №6
F(n)= 331053
O(F(n))= 19464
T(n)= 139
N_op= 169086


Сортировка №7
F(n)= 393728
O(F(n))= 23175
T(n)= 189
N_op= 200166


Сортировка №8
F(n)= 457396
O(F(n))= 26949
T(n)= 248
N_op= 233538
```

```
Не россортированное
Queue {
  _Array: [
    [ 0.544562133899342 ],
    [ 0.9497831072351426 ],
    [ 0.0534024787619114 ],
    [ 0.06585752140974055 ],
    [ 0.34114900703256335 ],
    [ 0.010146173214366616 ],
    [ 0.6710647335416198 ],
    [ 0.21992523275415587 ]
  ]
}
Россортированное
Queue {
  _Array: [
    [
      0.010146173214366616,
      0.0534024787619114,
      0.06585752140974055,
      0.21992523275415587,
      0.34114900703256335,
      0.544562133899342,
      0.6710647335416198,
      0.9497831072351426
    ]
  ]
}

Не россортированное
Queue {
  _Array: [
    [ 0.8722112536462812 ],
    [ 0.3046836457141733 ],
    [ 0.7625768564447961 ],
    [ 0.06532533396771778 ],
    [ 0.8626644682788669 ],
    [ 0.7030776141077684 ],
    [ 0.35321202374937277 ],
    [ 0.4517398598649398 ],
    [ 0.615449393297473 ]
  ]
}
Россортированное
Queue {
  _Array: [
    [
      0.06532533396771778,
      0.3046836457141733,
      0.35321202374937277,
      0.4517398598649398,
      0.615449393297473,
      0.7030776141077684,
      0.7625768564447961,
      0.8626644682788669,
      0.8722112536462812
    ]
  ]
}
```

Скриншоты работы программы

Выводы.

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

Достоинства данного алгоритма сортировки: Не имеет «трудных» входных данных, сохраняет порядок равных элементов

Недостатки данного алгоритма сортировки: На «почти отсортированных» массивах работает столь же долго, как на хаотичных.

Неплохо работает в параллельном варианте: легко разбить задачи между процессорами поровну, но трудно сделать так, чтобы другие процессоры взяли на себя работу, в случае если один процессор задержится.

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.

```
class Queue {
    private _Array: number[][]
    public _F_lim: number
    public _O_F_lim: number
    public _T_lim: number
    public _N_op: number

    constructor(quantity: number) {
        this._Array = []
        this._F_lim = Math.floor(1+( 11+Math.log2(quantity)*16)*(quantity-1))
        this._O_F_lim = Math.floor(Math.log2(quantity)*quantity) // *=8,62 => 1
        this._T_lim = 0
        this._N_op = 0
    }
```

```
Push(num: number[]){
    this._N_op++
    this._Array.push(num)
}
Shift(){
    this._N_op++
    return this._Array.shift()!
}
First(){
    this._N_op++
    return this._Array.at(0)!
}
Length(){
    this._N_op++
    return this._Array.length
}
Merger() {
    const timer = Date.now()
    while (this.Length() != 1){

        const A1 = this.Shift()
        this._N_op+=1

        const A2 = this.Shift()
        this._N_op+=1

        const limit = A1.length + A2.length
        this._N_op+=3

        let retA :number[] = []
        this._N_op+=1

        for (let i = 0; i < limit; i++){
            if(A1.length > 0 && A2.length > 0){

                this._N_op+=8
                A1 > A2 ? retA.push( A2.shift()! ) : retA.push( A1.shift()! )
            }
            else if(A2.length === 0){

                this._N_op+=4
                retA.push( A1.shift()! )
            }
            else if(A1.length === 0){

                this._N_op+=4
                retA.push( A2.shift()! )
```

```
                }
            }

            this._N_op+=1
            this.Push( retA.flat())
        }
        this._T_lim = Number((Date.now()-timer).toFixed(3))
    }
}

//for (let i = 300; i < 9001; i+=300) {
    let Quantity = 300//i
    let queue = new Queue(Quantity)

    for (let i = 0; i < Quantity; i++)
        queue.Push([Math.random()])

    queue.Merger()
    console.log('Сортировка №' + Quantity/300)
    console.log('F(n)= ' +queue._F_lim)
    console.log('O(F(n))= ' +queue._O_F_lim)
    console.log('T(n)= ' +queue._T_lim)
    console.log('N_op= ' +queue._N_op)
    console.log()
    //console.log(queue)
    // console.log("C1 = " + queue._F_lim / queue._T_lim)
    // console.log("C2 = " + queue._O_F_lim / queue._T_lim)
    // console.log("C3 = " + queue._F_lim / queue._N_op)
    // console.log("C4 = " + queue._O_F_lim / queue._N_op)
    console.log()
//}
```