|  |
| --- |
| 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 практической работы №1**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Программная реализация нелинейных структур»**

**Вариант № 38**

Выполнил: студент 2 курса

группы \_\_\_\_\_\_\_\_\_\_

шифр БСБО-04-19

Шабанова Мария Алексеевна  
*(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 № 37.**

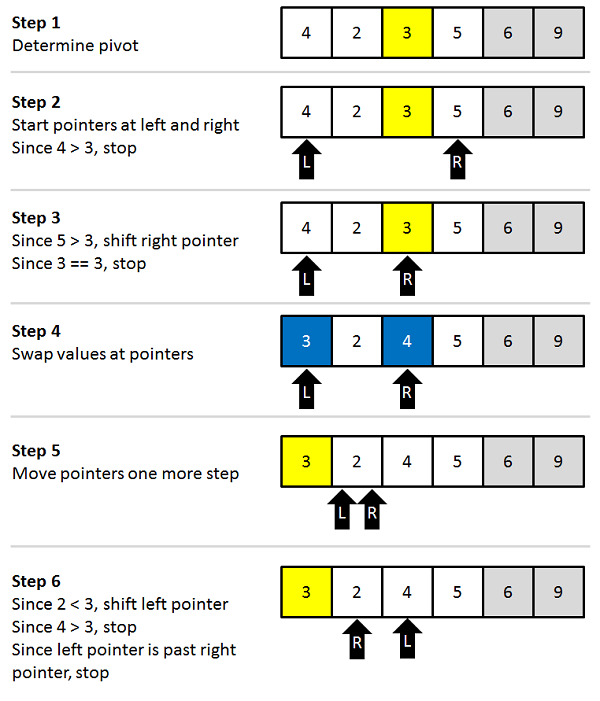
**Реализация связи элементов линейного списка: Массив**

**Способ организации линейного связанный список: Стек**

**Алгоритм сортировки: Быстрая сортировка с медианным элементом**

**Теория о сортировках.**

Быстрая сортировка относится к парадигме «Разделяй и властвуй». Суть сортировки с медианным элементом заключается в следующем: Находится медианный элемент, два цикла меняют местами элементы, которые больше/меньше медианного. Затем оба цикла встречаются в некоей точке, которая далее делит массив на два сабмассива, затем рекурсивно вызываются их сортировки. Деление общей задачи на подзадачи поменьше происходит до того момента, пока сабмассивы не будут состоять из 2-3 элементов. Стек вызовов закрывается и возвращается отсортированный массив.



**Листинг программы с расчетами**

internal class Stack

{

int MAX;

int top;

public int[] stack;

bool IsEmpty()

{

return (top < 0);

}

public Stack(int max)

{

stack = new int[max];

top = -1;

MAX = max;

}

internal void Push(int item) ***// ПОСЧИТАТЬ трудоёмкость***

{

if (top >= MAX)

{

Console.WriteLine("Stack Overflow");

}

else

{

stack[++top] = item;

}

}

internal int Pop() ***// ПОСЧИТАТЬ трудоёмкость***

{

if (top < 0)

{

Console.WriteLine("Stack Underflow");

return 0;

}

else

{

int value = stack[top--];

return value;

}

}

internal void Peek()

{

if (top < 0)

{

Console.WriteLine("Stack Underflow");

return;

}

else

Console.WriteLine("The topmost element of Stack is : {0}", stack[top]);

}

internal int Get(int index) // СЛОЖНОСТЬ – O(2n)

{***// ПОСЧИТАТЬ трудоёмкость***

int toReturn = -1;

Stack tmp\_st = new Stack(top+1);

int top\_ = this.top;

for (int i=0; i < top\_+1; i++) // Сложность N

{

tmp\_st.Push(this.Pop());

}

for (int i=0; i < top\_+1; i++) // Сложность N

{

if (i == index)

{

toReturn = tmp\_st.Pop();

this.Push(toReturn);

}

else

{

this.Push(tmp\_st.Pop());

}

}

return toReturn;

}

internal void Set(int index, int item) // СЛОЖНОСТЬ – O(2n)

{***// ПОСЧИТАТЬ трудоёмкость***

Stack tmp\_st = new Stack(top+1);

int top\_ = this.top;

for (int i=0; i<top\_+1; i++) // Сложность N

{

tmp\_st.Push(this.Pop());

}

for(int i=0; i< top\_+1; i++) // Сложность N

{

if (i == index)

{

int toDelete = tmp\_st.Pop();

this.Push(item);

}

else

{

this.Push(tmp\_st.Pop());

}

}

}

internal void PrintStack()

{

if (top < 0)

{

Console.WriteLine("Stack Underflow");

return;

}

else

{

Console.WriteLine("Items in the Stack are :");

for (int i = 0; i<top+1 ; i++)

{

Console.Write(stack[i]+" ");

}

Console.WriteLine();

}

}

static internal void SwapGS(ref Stack st, int ind1, int ind2) // СЛОЖНОСТЬ – O(8n)

{***// ПОСЧИТАТЬ трудоёмкость***

int val1 = st.Get(ind1); // 2n

int val2 = st.Get(ind2); // 2n

st.Set(ind1, val2); // 2n

st.Set(ind2, val1); // 2n

}

static internal void QSortGS(ref Stack st, int firstIndex = 0, int lastIndex = -1) // СЛОЖНОСТЬ – O(24n) ***// ПОСЧИТАТЬ трудоёмкость***

{

if (lastIndex < 0)

{

lastIndex = st.top;

}

if (firstIndex >= lastIndex)

{

return;

}

int middleIndex = (lastIndex - firstIndex) / 2 + firstIndex, currentIndex = firstIndex;

SwapGS(ref st, firstIndex, middleIndex); // 8n

for (int i = firstIndex + 1; i <= lastIndex; ++i)

{

if (st.Get(i) <= st.Get(firstIndex))

{

SwapGS(ref st, ++currentIndex, i); // 8n

}

}

SwapGS(ref st, firstIndex, currentIndex); // 8n

QSortGS(ref st, firstIndex, currentIndex - 1); // log(2)n

QSortGS(ref st, currentIndex + 1, lastIndex); // log(2)n

}

F(n) =

O(F(n)) = (logn \* 24n) его ебанутый get set сломал всю суть быстрой сортировки

**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (мсек) | N\_op |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Скриншоты работы программы**

**Вывод**

В результате экспериментов стало ясно, что хоть и временная сложность алгоритма быстрой сортировки является O(nlogn), из-за специфической работы стека через методы Get/Set во время сортировки, реальная скорость работы оказывается в разы хуже теоретической, что ставит под сомнение целесообразность использования данного вида сортировки для структур, в которых метод Swap выполняется не за O(1) или O(2), а за O(2-8n) время.

**Литература**

1 Грокаем алгоритмы. Бхаргава Адитья. – М.: Питер, 2019

2. https://en.wikipedia.org/wiki/Quicksort

3. https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/qsort?view=msvc-160

**Приложение 1. Применение счетчика операций N\_op.**

internal class Stack

{

int MAX;

int top;

public int[] stack;

bool IsEmpty()

{

return (top < 0);

}

public Stack(int max)

{

Program.N\_op += 4;

stack = new int[max];

top = -1;

MAX = max;

}

internal void Push(int item)

{

Program.N\_op += 1;

if (top >= MAX)

{

Program.N\_op += 3;

Console.WriteLine("Stack Overflow");

}

else

{

Program.N\_op += 3;

stack[++top] = item;

}

}

internal int Pop()

{

if (top < 0)

{

Program.N\_op += 3;

Console.WriteLine("Stack Underflow");

return 0;

}

else

{

Program.N\_op += 4;

int value = stack[top--];

return value;

}

}

internal void Peek()

{

if (top < 0)

{

Console.WriteLine("Stack Underflow");

return;

}

else

Console.WriteLine("The topmost element of Stack is : {0}", stack[top]);

}

internal int Get(int index)

{

int toReturn = -1;

Stack tmp\_st = new Stack(top+1);

int top\_ = this.top;

Program.N\_op += 8;

for (int i=0; i < top\_+1; i++)

{

Program.N\_op += 5;

tmp\_st.Push(this.Pop());

}

for (int i=0; i < top\_+1; i++)

{

Program.N\_op += 3;

if (i == index)

{

Program.N\_op += 3;

toReturn = tmp\_st.Pop();

this.Push(toReturn);

}

else

{

Program.N\_op += 2;

this.Push(tmp\_st.Pop());

}

}

Program.N\_op += 1;

return toReturn;

}

internal void Set(int index, int item)

{

Stack tmp\_st = new Stack(top+1);

int top\_ = this.top;

Program.N\_op += 8;

for (int i=0; i<top\_+1; i++)

{

Program.N\_op += 4;

tmp\_st.Push(this.Pop());

}

for(int i=0; i< top\_+1; i++)

{

Program.N\_op += 3;

if (i == index)

{

Program.N\_op += 4;

int toDelete = tmp\_st.Pop();

this.Push(item);

}

else

{

Program.N\_op += 2;

this.Push(tmp\_st.Pop());

}

}

}

internal void PrintStack()

{

if (top < 0)

{

Console.WriteLine("Stack Underflow");

return;

}

else

{

Console.WriteLine("Items in the Stack are :");

for (int i = 0; i<top+1 ; i++)

{

Console.Write(stack[i]+" ");

}

Console.WriteLine();

}

}

static internal void SwapGS(ref Stack st, int ind1, int ind2)

{

int val1 = st.Get(ind1);

int val2 = st.Get(ind2);

st.Set(ind1, val2);

st.Set(ind2, val1);

Program.N\_op += 9;

}

static internal void QSortGS(ref Stack st, int firstIndex = 0, int lastIndex = -1)

{

Program.N\_op += 22;

if (lastIndex < 0)

{

lastIndex = st.top;

Program.N\_op += 2;

}

if (firstIndex >= lastIndex)

{

Program.N\_op += 1;

return;

}

int middleIndex = (lastIndex - firstIndex) / 2 + firstIndex, currentIndex = firstIndex;

SwapGS(ref st, firstIndex, middleIndex);

for (int i = firstIndex + 1; i <= lastIndex; ++i)

{

Program.N\_op += 2;

if (st.Get(i) <= st.Get(firstIndex))

{

Program.N\_op += 3;

SwapGS(ref st, ++currentIndex, i);

}

}

SwapGS(ref st, firstIndex, currentIndex);

QSortGS(ref st, firstIndex, currentIndex - 1);

QSortGS(ref st, currentIndex + 1, lastIndex);

}