|  |
| --- |
| 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 № 79**

Выполнил: студент 2 курса

Группы \_\_\_\_\_\_\_\_\_

шифр \_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
*(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 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 № 79.**

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

**Способ организации линейного связанный список: Очередь**

**Алгоритм сортировки: Пузырковая**

**Теория о сортировках.**

**Сортировка простыми обменами**, **сортировка пузырьком** (англ. *bubble sort*) — простой алгоритм сортировки. Для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов.

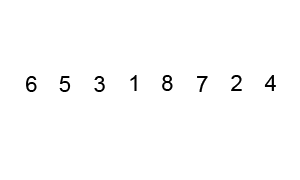


Рисунок 1. Иллюстрация алгоритма сортировки пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов. Проходы по массиву повторяются N − 1 {\displaystyle N-1} раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает - массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде - отсюда и название алгоритма).

**Листинг программы с расчетами.**

from random import randint

from time import time

from queue import SimpleQueue

N\_OP = 0

def print\_queue(*queue*: SimpleQueue):

    elems = []

    for \_ in range(*queue*.qsize()):

        el = *queue*.get()

        elems.append(el)

*queue*.put(el)

    print(", ".join(map(str, elems)))

def rotate(*queue*: SimpleQueue): # 2

    global N\_OP

*queue*.put(*queue*.get()) # 2

    N\_OP += 2

def seek(*queue*: SimpleQueue, *pos*: int): # 2pos + 3 + 2n - 2pos - 2 = 2n + 1

    global N\_OP

    for \_ in range(*pos*): # 2pos

        rotate(*queue*) # 2

    el = *queue*.get() # 2

*queue*.put(el) # 1

    for \_ in range(*queue*.qsize() - *pos* - 1): # 2n - 2pos - 2

        rotate(*queue*) # 2

    N\_OP += *queue*.qsize() \* 2 + 1

    return el

def pop\_by\_pos(*queue*: SimpleQueue, *pos*: int): # 2pos + 2 + 2n - 2pos = 2n + 2

    global N\_OP

    for \_ in range(*pos*): # 2pos

        rotate(*queue*) # 2

    el = *queue*.get() # 2

    for \_ in range(*queue*.qsize() - *pos*): # 2n - 2pos

        rotate(*queue*) # 2

    N\_OP += *queue*.qsize() \* 2 + 2

    return el

def push\_by\_pos(*queue*: SimpleQueue, *el*, *pos*: int): # 2 + 2pos + 1 + 2n - 2pos - 2 = 2n + 1

    global N\_OP

    if *pos* >= *queue*.qsize(): # 2

*queue*.put(*el*) # 1

        N\_OP += 3

        return

    for \_ in range(*pos*): # 2pos

        rotate(*queue*) # 2

*queue*.put(*el*) # 1

    for \_ in range(*queue*.qsize() - *pos* - 1): # 2n - 2pos - 2

        rotate(*queue*) # 2

    N\_OP += *queue*.qsize() \* 2 + 1

def swap(*queue*: SimpleQueue, *pos1*: int, *pos2*: int): # 8n + 7

    temp = pop\_by\_pos(*queue*, *pos1*) # 2n + 3

    push\_by\_pos(*queue*, pop\_by\_pos(*queue*, *pos2* - 1), *pos1*) # 2n + 2 + 2n + 1 = 4n + 3

    push\_by\_pos(*queue*, temp, *pos2*) # 2n + 1

def bubble\_sort(*queue*: SimpleQueue): # 12n^3 + 10n^2

    for i in range(*queue*.qsize()): # Σ (i=0 -> n) ((n - i) \* (12n + 10)) = n^2 \* (12n + 10) = 12n^3 + 10n^2

        for j in range(i + 1, *queue*.qsize() - 1): # Σ(j = i + 1 -> n - 1) (4n + 3 + 8n + 7) = (n - i) \* (12n + 10)

            if seek(*queue*, i) > seek(*queue*, j): # 4n + 3

                swap(*queue*, i, j) # 8n + 7

if \_\_name\_\_ == "\_\_main\_\_":

    step = 100

    tests = 10

    cur\_elems = 0

    for i in range(tests):

        cur\_elems += step

        queue = SimpleQueue()

        N\_OP = 0

        for \_ in range(cur\_elems):

            queue.put(randint(-999, 1000))

        start\_time = time()

        bubble\_sort(queue)

        diff\_time = time() - start\_time

        print(f"Test: {i + 1}\n"

              f"Elems: {cur\_elems}\n"

              f"Time: {diff\_time}\n"

              f"N\_OP: {N\_OP}\n"

              "-------------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 100 | 12100000 | 1000000 | 0,3707 | 7189010 |
| 200 | 96400000 | 8000000 | 2,7159 | 63093690 |
| 300 | 324900000 | 27000000 | 9,3949 | 209847162 |
| 400 | 769600000 | 64000000 | 22,0330 | 497256542 |
| 500 | 1502500000 | 125000000 | 41,7928 | 948459630 |
| 600 | 2595600000 | 216000000 | 74,0148 | 1620895954 |
| 700 | 4120900000 | 343000000 | 118,7271 | 2563356276 |
| 800 | 6150400000 | 512000000 | 172,4658 | 3777664206 |
| 900 | 8756100000 | 729000000 | 247,1642 | 5370923806 |
| 1000 | 12010000000 | 1000000000 | 334,9552 | 7328244858 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 32640949,55 | 2697599,137 | 1,683124658 | 0,139101211 |
| 35494679,48 | 2945616,554 | 1,527886545 | 0,126795564 |
| 34582592,68 | 2873899,669 | 1,548269688 | 0,128665071 |
| 34929424,05 | 2904733,808 | 1,547692056 | 0,1287062 |
| 35951168,62 | 2990945,809 | 1,584147551 | 0,131792642 |
| 35068661,94 | 2918335,252 | 1,60133659 | 0,133259633 |
| 34709009,15 | 2888978,169 | 1,607618901 | 0,133808945 |
| 35661563,05 | 2968704,52 | 1,628096004 | 0,135533486 |
| 35426247,01 | 2949456,272 | 1,630278201 | 0,13573084 |
| 35855541,28 | 2985473,878 | 1,638864453 | 0,136458322 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**



Рисунок 2. Пример очереди после добавления элемента

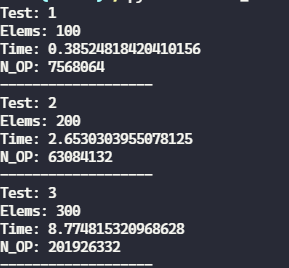


Рисунок 3. Пример вывода программы при выполнении сортировок

**Выводы.**

По результатам экспериментов было установлено, что алгоритм сортировки **пузырьком** для **очереди** имеет **кубическую** зависимость от числа элементов.

**Литература:**

* Левитин А. В. Глава 3. Метод грубой силы: Пузырьковая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.