|  |
| --- |
| 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 № 23**

Выполнил: студент 2 курса

Группы **\_\_\_\_\_\_\_\_\_\_\_**

шифр \_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
*(фио студента)*

Проверил:

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

Москва 2022 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 № 23.**

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

**Способ организации линейного связанный список: Очередь**

**Алгоритм сортировки: Бинарная вставка**

**Теория о сортировках.**

Сортировка вставками (англ. Insertion sort) — квадратичный алгоритм сортировки.

Задача заключается в следующем: есть часть массива, которая уже отсортирована, и требуется вставить остальные элементы массива в отсортированную часть, сохранив при этом упорядоченность. Для этого на каждом шаге алгоритма мы выбираем один из элементов входных данных и вставляем его на нужную позицию в уже отсортированной части массива, до тех пор, пока весь набор входных данных не будет отсортирован. Метод выбора очередного элемента из исходного массива произволен, однако обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве.

Так как в процессе работы алгоритма могут меняться местами только соседние элементы, каждый обмен уменьшает число инверсий на единицу. Следовательно, количество обменов равно количеству инверсий в исходном массиве вне зависимости от реализации сортировки. Максимальное количество инверсий содержится в массиве, элементы которого отсортированы по невозрастанию. Число инверсий в таком массиве

Алгоритм работает за где — число обменов элементов входного массива, равное числу инверсий. В среднем и в худшем случае — за Минимальные оценки встречаются в случае уже упорядоченной исходной последовательности элементов, наихудшие — когда они расположены в обратном порядке.

**Листинг программы с расчетами.**

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 binary\_search\_sort(queue: SimpleQueue) -> SimpleQueue:

    """

    Sort  queue using binary search algorithm

    """

    global N\_OP

    N\_OP = 0

    for i in range(1, queue.qsize()):  # (n - 1) \* (

        el = seek(queue, i)  # 2n + 1

        left = 0  # 1

        right = i - 1  # 1

        N\_OP += 2 + 2 \* queue.qsize() + 1

        while left <= right:  # log2(n - 1) \* (

            mid = (left + right) // 2  # 3

            mid\_el = seek(queue, mid)  # 2n + 1

            N\_OP += 3 + 2 \* queue.qsize() + 1

            if el < mid\_el:  # 1

                right = mid - 1  # 2

            else:  # 1

                left = mid + 1  # 2

            N\_OP += 3

        # ) = log2(n - 1) \* (2n + 7)  = 2nlog(n-1) + 7log(n-1)

        swap(queue, left, i)  # 8n + 7

        N\_OP += 8 \* queue.qsize() + 7

    # ) = 2n^2\*log(n-1) + 7nlog(n-1) - 2nlog(n-1) - 7log(n-1) + 2n^2 + n - 2n - 1

    return queue

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

    import time

    from random import randint

    tests = 10

    step = 300

    for test\_num in range(1, tests + 1):

        queue = SimpleQueue[int]()

        for \_ in range(test\_num \* step):

            queue.put(randint(-10000, 10000))

        start\_time = time.time()

        binary\_search\_sort(queue)

        total\_time = time.time() - start\_time

        print(f"Test: {test\_num}")

        print(f"Elems count: {test\_num \* step}")

        print(f"Total time: {total\_time}")

        print(f"N\_OP: {N\_OP}")

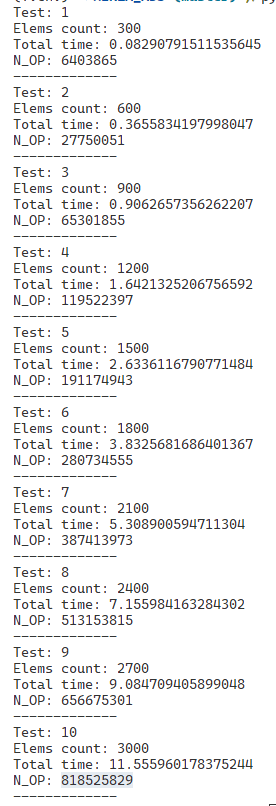
        print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 300 | 1673172 | 740594 | 0,082907915 | 6403865 |
| 600 | 7391770 | 3322375 | 0,36558342 | 27750051 |
| 900 | 17561518 | 7949163 | 0,906265736 | 65301855 |
| 1200 | 32399098 | 14729499 | 1,642132521 | 119522397 |
| 1500 | 52055916 | 23739180 | 2,633611679 | 191174943 |
| 1800 | 76648749 | 35036651 | 3,832568169 | 280734555 |
| 2100 | 106272753 | 48669526 | 5,308900595 | 387413973 |
| 2400 | 141008258 | 64677996 | 7,155984163 | 513153815 |
| 2700 | 180924785 | 83096842 | 9,084709406 | 656675301 |
| 3000 | 226083621 | 103956721 | 11,55596018 | 818525829 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 20181088,72 | 8932725,918 | 0,261275338 | 0,115647922 |
| 20219107,08 | 9087870,37 | 0,266369612 | 0,119724995 |
| 19377890,12 | 8771337,647 | 0,268928315 | 0,121729509 |
| 19729892,52 | 8969738,27 | 0,271071355 | 0,123236308 |
| 19765980,19 | 9013925,802 | 0,272294661 | 0,124175166 |
| 19999317,97 | 9141820,711 | 0,273029266 | 0,124803486 |
| 20017845,68 | 9167533,798 | 0,274313164 | 0,125626666 |
| 19704942,65 | 9038308,943 | 0,274787507 | 0,126040173 |
| 19915307,93 | 9146890,429 | 0,275516355 | 0,12654175 |
| 19564243,73 | 8995939,711 | 0,276208292 | 0,127004814 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**



**Выводы.**

По результатам экспериментов было установлено, что алгоритм сортировки бинарной вставкой для очереди из библиотеки классов имеет зависимость от числа элементов

**Литература:**

1. Д. Кнут. Искусство программирования. Том 3. Сортировка и поиск, 2-е изд.