|  |
| --- |
| 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 № 69**

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 № 69.**

**Реализация связи элементов линейного списка: Массив**

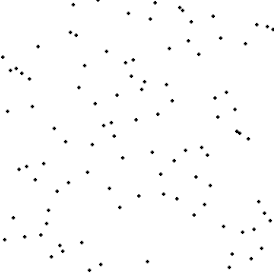
**Способ организации линейного связанный список: Очередь**

**Алгоритм сортировки: Простой выбор**

**Теория о сортировках.**

**Сортировка Простым Выбором.**

Может быть как устойчивый, так и неустойчивый. На массиве из *n* элементов имеет время выполнения в худшем, среднем и лучшем случае Θ(*n*2), предполагая, что сравнения делаются за постоянное время.

Рисунок иллюстрирует сортировку простым выбором.

Шаги алгоритма:

1. Находим номер минимального значения в текущем списке
2. Производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции)
3. Теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы

**Листинг программы с расчетами.**

from typing import Generic, List, TypeVar

VT = TypeVar("VT")

class Queue(Generic[VT]):

    \_queue: List[VT]

    \_array\_size: int

    \_bias: int

    \_size: int

    \_n\_op: int

    def \_\_init\_\_(self) -> None:

        self.\_queue = [0]

        self.\_array\_size = 1

        self.\_size = 0

        self.\_n\_op = 0

        self.\_bias = 0

    def push(self, value: VT) -> None:  # 13

        if self.\_size == self.\_array\_size:  # 3

            self.\_queue += [0] \* self.\_array\_size  # 3

            self.\_array\_size \*= 2  # 2

        self.\_queue[(self.\_size + self.\_bias) % self.\_array\_size] = value  # 3

        self.\_size += 1  # 2

        self.\_n\_op += 13

    def pop(self) -> VT:  # 14

        assert not self.empty, "Can't pop from empty queue!"  # 2

        el = self.\_queue[self.\_bias]  # 4

        self.\_size -= 1  # 2

        self.\_bias = (self.\_bias + 1) % self.\_array\_size  # 6

        self.\_n\_op += 14

        return el

    @property

    def empty(self) -> bool:  # 2

        return self.\_size == 0

    @property

    def size(self) -> int:  # 1

        return self.\_size

    @property

    def tail(self) -> VT:  # 10

        assert not self.empty, "Can't get head from empty dequeue!"  # 2

        self.\_n\_op += 10

        return self.\_queue[(self.\_size - 1 + self.\_bias) % self.\_max\_size]

    @property

    def head(self) -> VT:  # 5

        assert not self.empty, "Can't get tail from empty dequeue!"  # 2

        self.\_n\_op += 5

        return self.\_queue[self.\_bias]

    @property

    def n\_op(self) -> int:

        return self.\_n\_op

def print\_queue(queue: Queue[VT]):

    elements = []

    for \_ in range(queue.size):

        el = queue.pop()

        elements.append(el)

        queue.push(el)

    print("Queue[" + ", ".join(map(str, elements)) + "]")

def rotate(queue: Queue[VT]) -> None:  # 27

    queue.push(queue.pop())  # 27

def seek(queue: Queue[VT], pos: int) -> VT:  # 54\*n + 32

    assert pos <= queue.size and pos >= 0, "Invalid position!"  # 5

    for \_ in range(pos):  # n \* (

        rotate(queue)  # 27

    # ) = 27n

    el = queue.pop()  # 14

    queue.push(el)  # 13

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

        rotate(queue)  # 27

    # ) = 27n

    return el

def pop\_by\_pos(queue: Queue[VT], pos: int) -> VT:  # 54\*n + 19

    assert pos <= queue.size and pos >= 0, "Invalid position!"  # 5

    for \_ in range(pos):  # n \* (

        rotate(queue)  # 27

    # ) = 27n

    el = queue.pop()  # 14

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

        rotate(queue)  # 27

    # ) = 27n

    return el

def push\_by\_pos(queue: Queue[VT], el: VT, pos: int) -> None:  # 54\*n + 18

    assert pos <= queue.size and pos >= 0, "Invalid position!"  # 5

    for \_ in range(pos):  # n \* (

        rotate(queue)  # 27

    # ) = 27n

    queue.push(el)  # 13

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

        rotate(queue)  # 27

    # ) = 27n

def push\_back(queue: Queue[VT], el: VT) -> None:  # 13

    queue.push(el)  # 13

def push\_front(queue: Queue[VT], el: VT) -> None:  # 27\*n + 13

    queue.push(el)  # 13

    for \_ in range(queue.size - 1):  # n \* (

        rotate(queue)  # 27

    # ) = 27n

def pop\_back(queue: Queue[VT]) -> VT:  # 27\*n + 14

    for \_ in range(queue.size - 1):  # n \* (

        rotate(queue)  # 27

    # ) = 27n

    return queue.pop()  # 14

def pop\_front(queue: Queue[VT]) -> VT:  # 14

    return queue.pop()  # 14

def swap(queue: Queue[VT], pos1: int, pos2: int) -> None:  #  216\*n + 74

    temp = pop\_by\_pos(queue, pos1)  # 54n + 19

    push\_by\_pos(queue, pop\_by\_pos(queue, pos2 - 1), pos1)  # 108n + 37

    push\_by\_pos(queue, temp, pos2)  # 54n + 18

def slice\_(queue: Queue[VT], l: int = 0, r: int = -1) -> Queue[VT]:  # 99\*n + 6

    buff = Queue[VT]()  # 2

    if r == -1:  # 1

        r = queue.size - 1  # 3

    for \_ in range(l):  # n \* (

        rotate(queue)  # 27

    # ) = 27n

    for \_ in range(r - l + 1):  # n \* (

        buff.push(queue.head)  # 18

        rotate(queue)  # 27

    # ) = 45n

    for \_ in range(queue.size - r - 1):  # n \* (

        rotate(queue)  # 27

    # ) = 27n

    return buff

def selection\_sort(queue: Queue[VT]) -> Queue[VT]:  # 54\*n\*\*3 + 306\*n\*\*2 + 108\*n

    for start\_pos in range(queue.size):  # n \* (

        min\_el\_pos = start\_pos  # 1

        min\_el = seek(queue, min\_el\_pos)  # 54\*n + 33

        for check\_el\_pos in range(start\_pos + 1, queue.size):  # n \* (

            check\_el = seek(queue, check\_el\_pos)  # 54\*n + 33

            if check\_el < min\_el:  # 1

                min\_el = check\_el  # 1

                min\_el\_pos = check\_el\_pos  # 1

        # ) = 54\*n\*\*2 + 36\*n

        if min\_el\_pos != start\_pos:  # 1

            swap(queue, start\_pos, min\_el\_pos)  # 216\*n + 74

    # ) = 54\*n\*\*3 + 306\*n\*\*2 + 108\*n

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

    import sys

    import time

    from random import randint

    if len(sys.argv) < 2 or sys.argv[1] not in ["example", "tests"]:

        print(f"Usage: python3 {sys.argv[0]} [example/tests]")

        exit(1)

    if sys.argv[1] == "example":

        struct = Queue[int]()

        for \_ in range(20):

            struct.push(randint(-10000, 10000))

        selection\_sort(struct)

        print\_queue(struct)

    elif sys.argv[1] == "tests":

        tests = 10

        step = 100

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

            struct = Queue[int]()

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

                struct.push(randint(-10000, 10000))

            start\_time = time.time()

            selection\_sort(struct)

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

            print(f"Test: {test\_num}")

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

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

            print(f"N\_OP: {struct.n\_op}")

            print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 100 | 56070800 | 1000000 | 0,18846941 | 14662732 |
| 200 | 436261600 | 8000000 | 1,475524664 | 112755032 |
| 300 | 1458572400 | 27000000 | 5,013375521 | 375212748 |
| 400 | 3441003200 | 64000000 | 11,90149617 | 883100356 |
| 500 | 6701554000 | 125000000 | 23,66535425 | 1717288580 |
| 600 | 11558224800 | 216000000 | 41,28443527 | 2959294848 |
| 700 | 18329015600 | 343000000 | 64,94263768 | 4688911612 |
| 800 | 27331926400 | 512000000 | 96,39742136 | 6988217036 |
| 900 | 38884957200 | 729000000 | 138,8717504 | 9938761488 |
| 1000 | 53306108000 | 1000000000 | 189,9406435 | 13620541864 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 297506104,7 | 5305900,837 | 3,824034975 | 0,068200114 |
| 295665406,8 | 5421800,256 | 3,869109806 | 0,07095027 |
| 290936195,4 | 5385592,978 | 3,887321014 | 0,071959176 |
| 289123581,6 | 5377475,157 | 3,896503015 | 0,072471945 |
| 283179957 | 5281983,049 | 3,902404103 | 0,072789164 |
| 279965675,3 | 5231995,995 | 3,905736128 | 0,072990361 |
| 282233926 | 5281584,06 | 3,909012819 | 0,073151304 |
| 283533791,8 | 5311345,395 | 3,911144468 | 0,073266185 |
| 280006243,9 | 5249447,768 | 3,912455012 | 0,073349179 |
| 280646137,7 | 5264802,632 | 3,913655457 | 0,073418518 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**

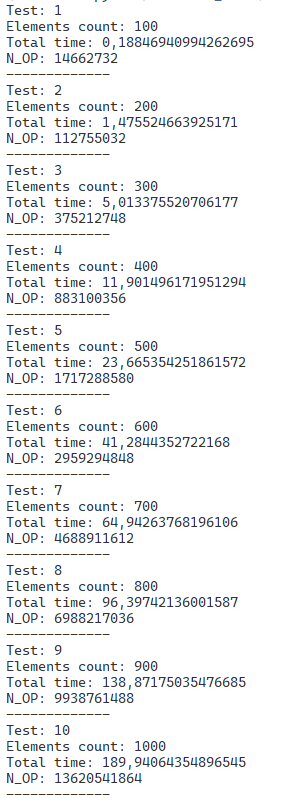


Рисунок 1. Результат работы программы в режиме tests



Рисунок 2. Результат работы программы в режиме examples

**Выводы.**

В данной работе был реализован класс Queue для работы с очередью через массив, так же были реализованы базовые операции над ним.

По результатам экспериментов было установлено, что алгоритм сортировки вставками для очереди через массив имеет зависимость от числа элементов

**Литература:**

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