|  |
| --- |
| 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 № 45**

Выполнил: студент 2 курса

Группы **\_\_\_\_\_\_\_\_\_\_\_**

шифр \_\_\_\_\_\_\_\_

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

Проверил:

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

Москва 2021 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 № 45.**

**Реализация связи элементов линейного списка: Массив**

**Способ организации линейного связанный список: Очередь**

**Алгоритм сортировки: Естественное двухпутевое слияние**

**Теория о сортировках.**

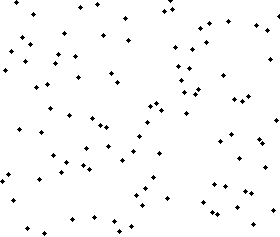
**Сортировка слиянием** (англ. *merge sort*) — алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

Рисунок иллюстрирует сортировку слиянием.

**Алгоритм**

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.  
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по возрастанию подмассива. Тогда:  
3.2. Слияние двух подмассивов в третий результирующий массив.  
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.  
3.3. «Прицепление» остатка.  
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

**Листинг программы с расчетами.**

from typing import Generic, TypeVar, List

VT = TypeVar("VT")

class Queue(Generic[VT]):

    \_queue: List[VT]

    \_nop: int

    def \_\_init\_\_(self):

        self.\_queue = []

        self.\_nop = 0

    def push(self, value: VT): # n + 2

        self.\_nop += self.size + 2

        self.\_queue = self.\_queue.copy()

        self.\_queue.append(0)

        # Имитация работы со списком как с массивом.

        # Создание массива на элемент больше и копирование элементов

        self.\_queue[-1] = value

    def pop(self) -> VT: # n + 2

        self.\_nop += self.size + 2

        if self.is\_empty():

            raise Exception("Can't pop from empty queue!")

        el = self.\_queue[0]

        self.\_queue = self.\_queue[1:] # создание нового списка на элемент меньше

        return el

    def is\_empty(self) -> bool: # 1

        self.\_nop += 1

        return len(self.\_queue) == 0

    @property

    def size(self) -> int: # 1

        self.\_nop += 1

        return len(self.\_queue)

    @property

    def tail(self) -> VT: # 1

        self.\_nop += 1

        return self.\_queue[-1]

    @property

    def head(self) -> VT: # 1

        self.\_nop += 1

        return self.\_queue[0]

    @property

    def n\_op(self) -> int:

        return self.\_nop

def print\_queue(queue: Queue):

    elems = []

    for \_ in range(queue.size):

        el = queue.pop()

        elems.append(el)

        queue.push(el)

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

def rotate(queue: Queue): # 2n + 4

    queue.push(queue.pop())

def seek(queue: Queue, pos: int): # 2n \* pos + 4pos + 4n + 8 + 2n^2 + 8n - 2n \* pos - 4pos - 2n - 4 = 2n^2

    for \_ in range(pos): # pos \* (2n + 4) = 2n \* pos + 4pos

        rotate(queue)

    el = queue.pop() # n + 2

    queue.push(el) # n + 2

    for \_ in range(queue.size - pos - 1): # (n - pos - 1) \* (2n + 4) = 2n^2 + 8n - 2n \* pos - 4pos - 2n - 4

        rotate(queue)

    return el

def pop\_by\_pos(queue: Queue, pos: int): # 2n^2 + 3n - 2

    for \_ in range(pos): # 2n \* pos + 4pos

        rotate(queue)

    el = queue.pop() # n + 2

    for \_ in range(queue.size - pos): # (n - pos - 1) \* (2n + 4)

        rotate(queue)

    return el

def push\_by\_pos(queue: Queue, el, pos: int): # 2n^2 + 3n - 2

    if pos >= queue.size: # 1

        queue.push(el) # n + 2

        return

    for \_ in range(pos): # 2n \* pos + 4pos

        rotate(queue)

    queue.push(el) # n + 2

    for \_ in range(queue.size - pos - 1): # (n - pos - 1) \* (2n + 4)

        rotate(queue)

def swap(queue: Queue, pos1: int, pos2: int): # 6n^2 + 9n - 7

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

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

    push\_by\_pos(queue, temp, pos2) # 2n^2 + 3n - 2

def slice\_queue(queue: Queue[VT], l: int = 0, r: int = -1) -> Queue[VT]: # 1 + 1 + 3 + 2n + 4 + 2n + 6 + 2n^2 - 4 = 2n^2 + 4n + 15

    q = Queue[VT]() # 1

    if r == -1: # 1

        r = queue.size - 1 # 3

    for \_ in range(l): # l \* (

        rotate(queue) # 2n + 4

    # ) = 2ln + 4l ~= 2n + 4

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

        q.push(queue.head) # k + 2

        rotate(queue) # 2n + 4

    # ) = (r - l + 1) \* (k + 6 + 2n) = (rk - lk + k + 2rn - 2ln + 2n + 6r - 6l + 6) ~= 2n + 6

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

        rotate(queue) # 2n + 4

    # ) = (2n^2 - 2rn - 2n + 4n - 4r - 4) ~= 2n^2 - 4

    return q

def merge\_sort(queue: Queue): # 4(n^2)log(n) + 8n + 2log(nlog(n)) + 13

    def \_merge(left: Queue, right: Queue) -> Queue: # 1 + 2 + k^3 + m^3 + k^3 + 2k + m^3 + 2m = 2k^3 + 2m^3 + 2k + 2m + 3

        result = Queue() # 1

        i, j = 0, 0 # 2

        while i < left.size and j < right.size: # ((k + m) / 2) \* (

            left\_v = seek(left, i) # 2k^2

            right\_v = seek(right, j) # 2m^2

            if left\_v < right\_v: # 1

                result.push(left\_v) # s + 2

                i += 1 # 1

            else:

                result.push(right\_v) # s + 2

                j += 1 # 1

        # ) = k^3 + km^2 + ks + 3k + mk^2 + m^3 + ms + 3m ~= k^3 + m^3

        while i < left.size: # (k / 2) \* (

            result.push(seek(left, i)) # s + 2 + 2k^2

            i += 1 # 1

        # ) = sk / 2 + k + k^3 ~= k^3 + 2k

        while j < right.size: # (m / 2) \* (

            result.push(seek(right, j)) # s + 2 + 2m^2

            j += 1 # 1

        # ) = sm / 2 + m + m^3 ~= m^3 + 2m

        return result

    if queue.size < 2: # 2

        return queue

    middle = queue.size // 2 # 3

    left\_slice = slice\_queue(queue, r = middle - 1)# 2n^2 + 4n + 15

    right\_slice = slice\_queue(queue, l = middle)# 2n^2 + 4n + 15 +

    left = merge\_sort(left\_slice)# 2log(nlog(n))

    right = merge\_sort(right\_slice)# 2log(nlog(n))

    sorted\_queue = \_merge(left, right) # ~= (n^2)log(n)

    sorted\_queue.\_nop += left\_slice.n\_op + right\_slice.n\_op + left.n\_op + right.n\_op

    return sorted\_queue

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

    import time

    from random import randint

    tests = 10

    step = 200

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

        queue = Queue[int]()

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

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

        start\_time = time.time()

        queue = merge\_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}")

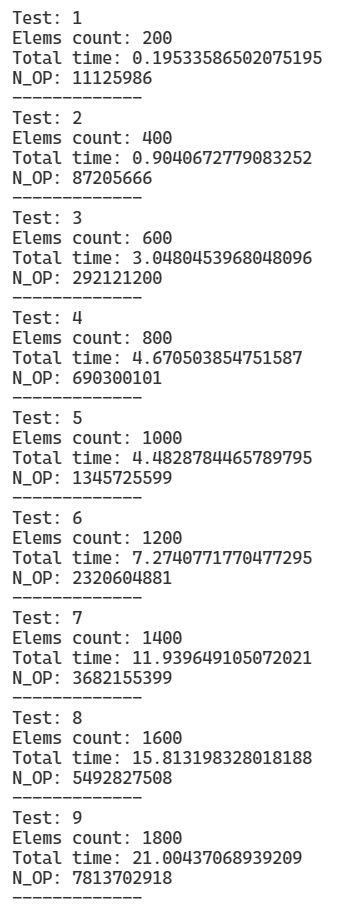
        print(f"N\_OP: {queue.n\_op}")

        print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 200 | 1224651,1467 | 305754,2476 | 0,1962 | 11125986 |
| 400 | 5535304,4725 | 1383016,99 | 0,9106 | 87205666 |
| 600 | 13294336,7842 | 3322374,729 | 2,2590 | 292121200 |
| 800 | 24694710,6728 | 6172067,961 | 5,2003 | 690300101 |
| 1000 | 39871176,7042 | 9965784,285 | 4,7665 | 1345725599 |
| 1200 | 58927635,8240 | 14729498,91 | 7,6013 | 2320604881 |
| 1400 | 81948735,7904 | 20484373,78 | 11,3025 | 3682155399 |
| 1600 | 109005928,4949 | 27248271,85 | 16,6415 | 5492827508 |
| 1800 | 140161045,7353 | 35036651,06 | 21,0811 | 7813702918 |
| 2000 | 175468590,3960 | 43863137,14 | 27,6889 | 10694675708 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 6241850,9 | 1558380,467 | 0,110071246 | 0,027481092 |
| 6078744,204 | 1518797,486 | 0,063474138 | 0,015859256 |
| 5885053,911 | 1470728,078 | 0,045509661 | 0,011373275 |
| 4748708,858 | 1186867,673 | 0,035773877 | 0,008941137 |
| 8364875,004 | 2090797,081 | 0,029628014 | 0,00740551 |
| 7752310,24 | 1937760,503 | 0,025393222 | 0,006347267 |
| 7250496,42 | 1812375,473 | 0,022255643 | 0,005563148 |
| 6550246,582 | 1637368,738 | 0,01984514 | 0,0049607 |
| 6648659,023 | 1661993,495 | 0,017937852 | 0,004484001 |
| 6337145,585 | 1584141,556 | 0,016407098 | 0,004101399 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**

**Выводы.**

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

**Литература:**

1. Левитин А. В. Глава 6. Метод декомпозиции: Сортировка слиянием // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.