|  |
| --- |
| 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 № 17**

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 № 17.**

**Реализация связи элементов линейного списка: Указатели**

**Способ организации линейного связанный список: Дек**

**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 typing import Generic, TypeVar, Optional

VT = TypeVar("VT")

class Node(Generic[VT]):

    value: VT

    next: Optional["Node"] = None

    prev: Optional["Node"] = None

    def \_\_init\_\_(self, value: VT):

        self.value = value

    def \_\_repr\_\_(self) -> str:

        return str(self.value)

class Dequeue(Generic[VT]):

    \_head: Optional[Node[VT]]

    \_tail: Optional[Node[VT]]

    \_size: int

    \_n\_op: int

    def \_\_init\_\_(self) -> None:

        self.\_head = None

        self.\_tail = None

        self.\_size = 0

        self.\_n\_op = 0

    def is\_empty(self) -> bool:  # 2

        return self.\_size == 0

    def push\_back(self, value: VT) -> None:  # 11

        node = Node[VT](value)  # 2

        if self.is\_empty():  # 2

            self.\_head = node  # 2

            self.\_tail = node  # 2

            self.\_n\_op += 6

        else:

            self.\_tail.next = node  # 3

            node.prev = self.\_tail  # 3

            self.\_tail = node  # 2

            self.\_n\_op += 10

        self.\_size += 1  # 1

        self.\_n\_op += 1

    def push\_front(self, value: VT) -> None:  # 11

        node = Node[VT](value)

        if self.is\_empty():  # 2

            self.\_head = node  # 2

            self.\_tail = node  # 2

            self.\_n\_op += 6

        else:

            self.\_head.prev = node  # 3

            node.next = self.\_head  # 3

            self.\_head = node  # 2

            self.\_n\_op += 10

        self.\_size += 1  # 1

        self.\_n\_op += 1

    def pop\_back(self) -> VT:  # 12

        if self.is\_empty():

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

        node = self.\_tail  # 2

        if self.size == 1:

            self.\_head = None

            self.\_tail = None

        else:

            self.\_tail = node.prev  # 3

            self.\_tail.next = None  # 3

            node.prev = None  # 2

        self.\_size -= 1  # 2

        self.\_n\_op += 11

        return node.value

    def pop\_front(self) -> VT:  # 12

        if self.is\_empty():

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

        node = self.\_head  # 2

        if self.size == 1:

            self.\_head = None

            self.\_tail = None

        else:

            self.\_head = node.next  # 3

            self.\_head.prev = None  # 3

            node.next = None  # 2

        self.\_size -= 1  # 2

        self.\_n\_op += 12

        return node.value

    @property

    def size(self) -> int:  # 1

        return self.\_size

    @property

    def head(self) -> VT:  # 2

        return self.\_head.value

    @property

    def tail(self) -> VT:  # 2

        return self.\_tail.value

    @property

    def n\_op(self) -> int:

        return self.\_n\_op

def print\_dequeue(dequeue: Dequeue[VT]) -> None:

    elements = []

    for \_ in range(dequeue.size):

        elements.append(dequeue.head)

        rotate\_left(dequeue)

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

def rotate\_left(dequeue: Dequeue[VT]) -> None:  # 23

    dequeue.push\_back(dequeue.pop\_front())

def rotate\_right(dequeue: Dequeue[VT]) -> None:  # 23

    dequeue.push\_front(dequeue.pop\_back())

def push\_by\_pos(dequeue: Dequeue[VT], el: VT, pos: int) -> None:  # 24n + 17

    if pos < 0 or pos > dequeue.size:  # 4

        raise Exception("Invalid position argument!")

    if pos == dequeue.size:  # 2

        dequeue.push\_back(el)

    elif pos == 0:  # 1

        dequeue.push\_front(el)

    else:

        if pos < dequeue.size // 2:  # 3

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

                rotate\_left(dequeue)  # 23

            # ) = 12n

            dequeue.push\_front(el)  # 11

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

                rotate\_right(dequeue)  # 23

            # ) = 12n

        else:

            for \_ in range(dequeue.size - pos):

                rotate\_right(dequeue)

            dequeue.push\_back(el)

            for \_ in range(dequeue.size - pos):

                rotate\_left(dequeue)

def pop\_by\_pos(dequeue: Dequeue[VT], i: int) -> VT:  # 24n + 17

    if i < 0 or i >= dequeue.size:

        raise Exception("Invalid position argument!")

    if i == dequeue.size - 1:

        return dequeue.pop\_back()

    elif i == 0:

        return dequeue.pop\_front()

    else:

        if i < dequeue.size // 2:

            for \_ in range(i):

                rotate\_left(dequeue)

            v = dequeue.pop\_front()

            for \_ in range(i):

                rotate\_right(dequeue)

        else:

            for \_ in range(dequeue.size - i - 1):

                rotate\_right(dequeue)

            v = dequeue.pop\_back()

            for \_ in range(dequeue.size - i - 1):

                rotate\_left(dequeue)

        return v

def seek(dequeue: Dequeue[VT], index: int) -> VT:  # 48n + 12

    if dequeue.is\_empty():  # 2

        raise Exception("Can't seek empty dequeue!")

    if index >= dequeue.size:  # 3

        raise Exception("Index out of range!")

    if index >= dequeue.size // 2:  # 3

        for \_ in range(dequeue.size - index - 1):  # (n / 2) \* (

            rotate\_right(dequeue)  # 23

        # ) = 12n

        node = dequeue.tail  # 2

        for \_ in range(dequeue.size - index - 1):  # (n / 2) \* (

            rotate\_left(dequeue)  # 23

        # ) = 12n

        return node

    else:

        for \_ in range(index):  # (n / 2) \* (

            rotate\_left(dequeue)  # 23

        # ) = 12n

        node = dequeue.head  # 2

        for \_ in range(index):  # (n / 2) \* (

            rotate\_right(dequeue)  # 23

        # ) = 12n

        return node

def swap(dequeue: Dequeue, pos1: int, pos2: int) -> None:  # 92n + 10

    left\_el\_pos, right\_el\_pos = min(pos1, pos2), max(pos1, pos2)  # 4

    if left\_el\_pos < 0 or right\_el\_pos >= dequeue.size:  # 3

        raise Exception("Invalid position argument!")

    if left\_el\_pos < dequeue.size // 2:  # 3

        for \_ in range(left\_el\_pos):  #  n \* (

            rotate\_left(dequeue)  # 23

        # ) = 23n

        left\_el = dequeue.pop\_front()  # 12

        for \_ in range(right\_el\_pos - left\_el\_pos - 1):  # n \* (

            rotate\_left(dequeue)  # 23

        # ) = 23n

        right\_el = dequeue.pop\_front()  # 12

        dequeue.push\_front(left\_el)  # 11

        for \_ in range(right\_el\_pos - left\_el\_pos - 1):  # n \* (

            rotate\_right(dequeue)  # 23

        # ) = 23n

        dequeue.push\_front(right\_el)  # 11

        for \_ in range(left\_el\_pos):  # n \* (

            rotate\_right(dequeue)  # 23

        # ) = 23n

    else:

        for \_ in range(dequeue.size - right\_el\_pos - 1):

            rotate\_right(dequeue)

        left\_el = dequeue.pop\_back()

        for \_ in range(right\_el\_pos - left\_el\_pos - 1):

            rotate\_right(dequeue)

        right\_el = dequeue.pop\_back()

        dequeue.push\_back(left\_el)

        for \_ in range(right\_el\_pos - left\_el\_pos - 1):

            rotate\_left(dequeue)

        dequeue.push\_back(right\_el)

        for \_ in range(dequeue.size - right\_el\_pos - 1):

            rotate\_left(dequeue)

def slice\_(dequeue: Dequeue[VT], l: int = 0, r: int = -1) -> Dequeue[VT]:  # 82n + 6

    buffer = Dequeue[VT]()  # 2

    if r == -1:  # 1

        r = dequeue.size - 1  # 3

    for \_ in range(l):  # n \* (

        rotate\_left(dequeue)  # 23

    # ) = 23n

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

        buffer.push\_back(dequeue.head)  # 13

        rotate\_left(dequeue)  # 23

    # ) = 36n

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

        rotate\_left(dequeue)  # 23

    # ) = 23n

    return buffer

def insertion\_sort(

    dequeue: Dequeue[VT],

) -> Dequeue[VT]:  # 188n^2 \* log(n) + 34n \* log(n)

    for i in range(1, dequeue.size):  # n \* (

        for j in range(i, 0, -1):  # log(n) \* (

            if seek(dequeue, j - 1) > seek(dequeue, j):  # 96n + 24

                swap(dequeue, j - 1, j)  # 92n + 10

            else:

                break

        # ) = 188n \* log(n) + 34 \* log(n)

    # ) = 188n^2 \* log(n) + 34n \* log(n)

    return dequeue

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 = Dequeue[int]()

        for \_ in range(20):

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

        insertion\_sort(struct)

        print\_dequeue(struct)

    elif sys.argv[1] == "tests":

        tests = 10

        step = 50

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

            struct = Dequeue[int]()

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

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

            start\_time = time.time()

            insertion\_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}")

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

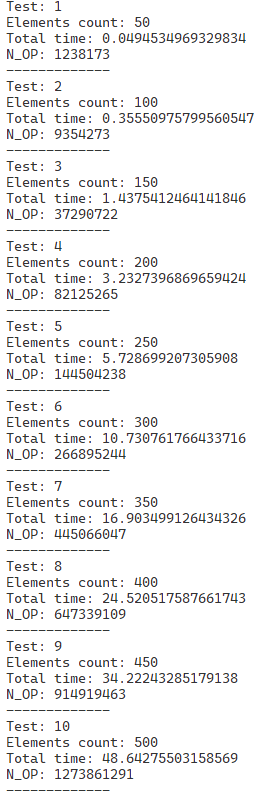
            print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 50 | 2662207 | 14110 | 0,049453497 | 1238173 |
| 100 | 12513039 | 66439 | 0,355509758 | 9354273 |
| 150 | 30614770 | 162648 | 1,437541246 | 37290722 |
| 200 | 57533777 | 305754 | 3,232739687 | 82125265 |
| 250 | 93665675 | 497862 | 5,728699207 | 144504238 |
| 300 | 139315546 | 740594 | 10,73076177 | 266895244 |
| 350 | 194731961 | 1035273 | 16,90349913 | 445066047 |
| 400 | 260124751 | 1383017 | 24,52051759 | 647339109 |
| 450 | 335675501 | 1784791 | 34,22243285 | 914919463 |
| 500 | 421544280 | 2241446 | 48,64275503 | 1273861291 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 53832532,17 | 285311,2793 | 2,150109043 | 0,011395532 |
| 35197455,11 | 186882,5269 | 1,337681587 | 0,007102483 |
| 21296620,26 | 113143,4809 | 0,820975524 | 0,004361632 |
| 17797219,18 | 94580,53453 | 0,700561231 | 0,003723023 |
| 16350251,8 | 86906,55588 | 0,648186349 | 0,003445307 |
| 12982819,79 | 69015,9467 | 0,52198587 | 0,002774848 |
| 11520216,01 | 61246,09783 | 0,437534974 | 0,002326112 |
| 10608452,68 | 56402,43871 | 0,401836915 | 0,002136464 |
| 9808639,329 | 52152,65376 | 0,366890764 | 0,001950763 |
| 8666126,732 | 46079,75164 | 0,330918509 | 0,001759568 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**



**Выводы.**

В данной работе был реализован класс Dequeue предоставляющий интерфейс работы с деком на указателях и базовые операции над ним. Так же были реализованы вспомогательные операции для сортировки.

По результатам экспериментов было установлено, что алгоритм сортировки бинарной вставкой для дека на указателях классов имеет зависимость от числа элементов

**Литература:**

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