|  |
| --- |
| 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 № 65**

Выполнил: студент 2 курса

Группы **БСБО-12-20**

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 № 65.**

**Реализация связи элементов линейного списка: Указатели**

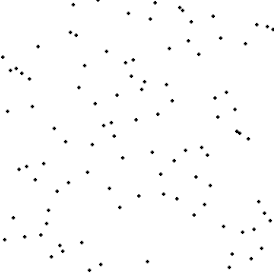
**Способ организации линейного связанный список: Дек**

**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.**

import random

import time

from typing import Generic, TypeVar

VT = TypeVar("VT")

class Node(Generic[VT]):

    value: VT

    next: 'Node' = None

    prev: 'Node' = None

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

*self*.value = *value*

    def \_\_repr\_\_(*self*) -> str:

        return str(*self*.value)

class Dequeue(Generic[VT]):

    \_head: Node[VT] = None

    \_tail: Node[VT] = None

    \_size: int = 0

    \_nop: int = 0

    def is\_empty(*self*) -> bool:

        return *self*.\_size == 0

    def push\_back(*self*, *value*: VT):

        node = Node[VT](*value*)

        if *self*.is\_empty():

*self*.\_head = node # 1

*self*.\_tail = node # 1

*self*.\_nop += 3

        else:

*self*.\_tail.next = node # 2

            node.prev = *self*.\_tail # 2

*self*.\_tail = node # 1

*self*.\_nop += 6

*self*.\_size += 1 # 1

    def push\_front(*self*, *value*: VT):

        node = Node[VT](*value*)

        if *self*.is\_empty():

*self*.\_head = node # 1

*self*.\_tail = node # 1

*self*.\_nop += 3

        else:

*self*.\_head.prev = node # 2

            node.next = *self*.\_head # 2

*self*.\_head = node # 1

*self*.\_nop += 6

*self*.\_size += 1 # 1

    def pop\_back(*self*) -> VT:

        if *self*.is\_empty():

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

        node = *self*.\_tail # 1

*self*.\_tail = node.prev # 2

*self*.\_tail.next = None # 2

        node.prev = None # 2

*self*.\_size -= 1 # 1

*self*.\_nop += 8

        return node.value

    def pop\_front(*self*) -> VT:

        if *self*.is\_empty():

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

        node = *self*.\_head # 1

*self*.\_head = node.next # 2

*self*.\_head.prev = None # 2

        node.next = None # 2

*self*.\_size -= 1 # 1

*self*.\_nop += 8

        return node.value

    def \_\_repr\_\_(*self*) -> str:

        repr\_str = ""

        node = *self*.\_head

        while node is not None:

            repr\_str += str(node)

            if node.next is not None:

                repr\_str += ", "

            node = node.next

        return "dequeue<[" + repr\_str + "]>"

    @property

    def size(*self*) -> int:

        return *self*.\_size

    @property

    def head(*self*) -> VT:

        return *self*.\_head.value

    @property

    def tail(*self*) -> VT:

        return *self*.\_tail.value

    @property

    def nop(*self*) -> int:

        return *self*.\_nop

def rotate\_left(*dequeue*: Dequeue): # 8 + 6 = 14

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

def rotate\_right(*dequeue*: Dequeue): # 8 + 6 = 14

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

def seek(*dequeue*: Dequeue[VT], *index*: int) -> VT: # 14n + 8

    if *dequeue*.is\_empty(): # 1

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

    if *index* >= *dequeue*.size: # 2

        raise Exception("Index out of range!")

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

        for \_ in range(*dequeue*.size - *index* - 1): # 14 \* (n / 2) = 7n

            rotate\_right(*dequeue*)

        node = *dequeue*.tail # 2

        for \_ in range(*dequeue*.size - *index* - 1): # 7n

            rotate\_left(*dequeue*)

        return node

    else:

        for \_ in range(*index*): # 14 \* (n / 2) = 7n

            rotate\_left(*dequeue*)

        node = *dequeue*.head # 2

        for \_ in range(*index*): # 14 \* (n / 2) = 7n

            rotate\_right(*dequeue*)

        return node

def swap(*dequeue*: Dequeue, *pos1*: int, *pos2*: int): # Σ(0, left\_el\_pos)(14) + Σ(0, left\_el\_pos)(14) + Σ(0, right\_el\_pos - left\_el\_pos - 1)(14) + 4 + 3 + 7 + 6 + 6 = 28n + 26

    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): # Σ(0, left\_el\_pos)(14)

            rotate\_left(*dequeue*) # 14

        left\_el = *dequeue*.pop\_front() # 7

        for \_ in range(right\_el\_pos - left\_el\_pos - 1): # Σ(0, right\_el\_pos - left\_el\_pos - 1)(14)

            rotate\_left(*dequeue*) # 14

        right\_el = *dequeue*.pop\_front() # 7

*dequeue*.push\_front(left\_el) # 6

        for \_ in range(right\_el\_pos - left\_el\_pos - 1): # Σ(0, right\_el\_pos - left\_el\_pos - 1)(14)

            rotate\_right(*dequeue*)

*dequeue*.push\_front(right\_el) # 6

        for \_ in range(left\_el\_pos): # Σ(0, left\_el\_pos)(14)

            rotate\_right(*dequeue*)

    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 dequeue\_selection\_sort(*dequeue*: Dequeue):

    for start\_pos in range(*dequeue*.size): # Σ(i = 0, n)( Σ(i, n)(14n + 8 + 3) + 14n + 9  + 28n + 26) = 14n^2 + 11n + 32n + 35 = 14n^3 + 43n^2 + 35n

        min\_el\_pos = start\_pos # 1

        min\_el = seek(*dequeue*, min\_el\_pos) # 14n + 8

        for check\_el\_pos in range(start\_pos + 1, *dequeue*.size): # Σ(i, n) (14n + 8 + 3) = 14n^2 + 11n

            check\_el = seek(*dequeue*, check\_el\_pos)  # 14n + 8

            if check\_el < min\_el: # 1

                min\_el = check\_el # 1

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

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

            swap(*dequeue*, start\_pos, min\_el\_pos) #28n + 26

def test\_case(*n*: int):

    dequeue = Dequeue[int]()

    for \_ in range(*n*):

        dequeue.push\_front(random.randint(0, 999))

    print("Sort case")

    print(f"Count: {*n*}")

    start\_time = time.time()

    dequeue\_selection\_sort(dequeue)

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

    print(f"Nop: {dequeue.nop}")

    print(f"Time: {total\_time}")

    print("-------------------------------------------------")

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

    test\_case(100)

    test\_case(200)

    test\_case(300)

    test\_case(400)

    test\_case(500)

    test\_case(600)

    test\_case(700)

    test\_case(800)

    test\_case(900)

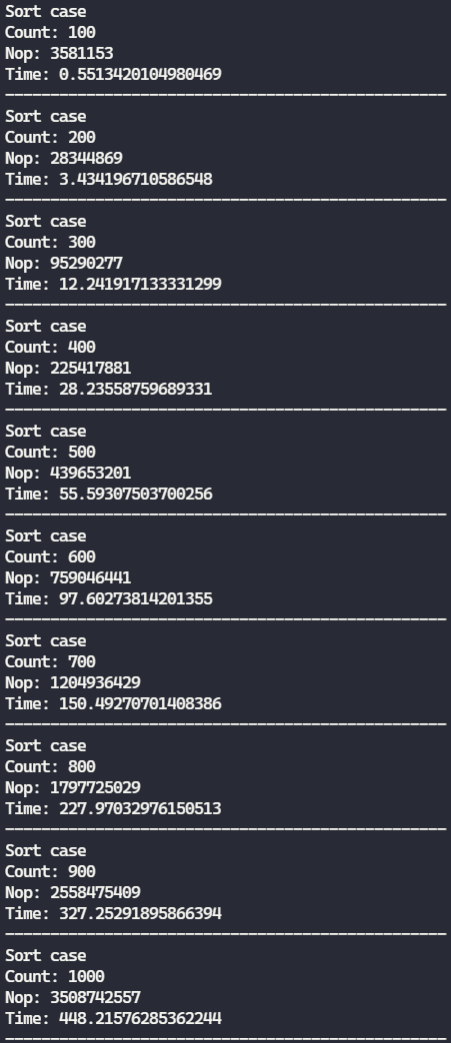
    test\_case(1000)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 100 | 14433500 | 1000000 | 0,5513 | 3581153 |
| 200 | 113727000 | 8000000 | 3,4341 | 28344869 |
| 300 | 381880500 | 27000000 | 12,2419 | 95290277 |
| 400 | 902894000 | 64000000 | 28,2355 | 225417881 |
| 500 | 1760767500 | 125000000 | 55,5930 | 439653201 |
| 600 | 3039501000 | 216000000 | 97,6027 | 759046441 |
| 700 | 4823094500 | 343000000 | 150,4927 | 1204936429 |
| 800 | 7195548000 | 512000000 | 227,9703 | 1797725029 |
| 900 | 10240861500 | 729000000 | 327,2529 | 2558475409 |
| 1000 | 14043035000 | 1000000000 | 448,2157 | 3508742557 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 26180845,27 | 1813894,431 | 4,03040585 | 0,27924 |
| 33116973,88 | 2329576,891 | 4,012260561 | 0,282238 |
| 31194544,96 | 2205539,99 | 4,00754948 | 0,283345 |
| 31977262,67 | 2266650,139 | 4,005423155 | 0,283917 |
| 31672467,76 | 2248484,521 | 4,004900899 | 0,284315 |
| 31141566,78 | 2213053,532 | 4,004367633 | 0,284568 |
| 32048694,06 | 2279180,319 | 4,00277922 | 0,284662 |
| 31563532,62 | 2245906,594 | 4,002585425 | 0,284804 |
| 31293417,11 | 2227634,958 | 4,002720317 | 0,284935 |
| 31330975,24 | 2231068,657 | 4,002298479 | 0,285002 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**



**Выводы.**

По результатам экспериментов было установлено, что алгоритм сортировки вставками для очереди с одной головой имеет кубическую 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