|  |
| --- |
| 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 № 72**

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 № 72.**

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

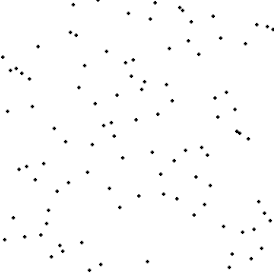
**Способ организации линейного связанный список: Стэк**

**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 List, Generic, TypeVar

VT = TypeVar("VT")

class Stack(Generic[VT]):

    \_stack: List[VT]

    \_n\_op: int

    def \_\_init\_\_(self):

        self.\_stack = []

        self.\_n\_op = 0

    def push(self, el: VT):  # n + 2

        self.\_n\_op += self.size + 2

        self.\_stack = self.\_stack.copy()

        self.\_stack.append(el)

    def pop(self) -> VT:  # n + 2

        self.\_n\_op += self.size + 2

        if self.is\_empty():

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

        el = self.\_stack[-1]

        self.\_stack = self.\_stack[:-1]

        return el

    def is\_empty(self):  # 1

        return len(self.\_stack) == 0

    @property

    def n\_op(self):

        return self.\_n\_op

    @property

    def size(self):  # 1

        return len(self.\_stack)

    @property

    def top(self):  # 1

        return self.\_stack[-1]

def print\_stack(stack: Stack):

    print("Stack[" + ", ".join(map(str, stack.\_stack)) + "]")

def seek(stack: Stack, pos: int):  # ~= 2n^2 + 3n - 8

    swap\_stack = Stack()  # 1

    stack\_size = stack.size  # 2

    for \_ in range(stack\_size - pos - 1):  # (n - pos - 1) \* (

        swap\_stack.push(stack.pop())  # k + 2 + n + 2

    # ) ~= n^2 + 4n - n\*pos - 4pos - n - 4

    el = stack.top  # 2

    for \_ in range(stack\_size - pos - 1):  # (n - pos - 1) \* (

        stack.push(swap\_stack.pop())  # k + 2 + n + 2

    # ) ~= n^2 + 4n - n\*pos - 4pos - n - 4

    return el

def push\_by\_pos(stack: Stack[VT], el: VT, pos: int):  # ~= 2n^2 + 4n + 2

    swap\_stack = Stack()  # 1

    stack\_size = stack.size  # 2

    for \_ in range(stack\_size - pos):  # (n - pos) \* (

        swap\_stack.push(stack.pop())  # k + n + 4

    # ) ~= n^2 + 4n - n \* pos - 4pos

    stack.push(el)  # n + 2

    for \_ in range(stack\_size - pos):  # (n - pos) \* (

        stack.push(swap\_stack.pop())  # k + n + 4

    # ) ~= n^2 + 4n - n \* pos - 4pos

def pop\_by\_pos(stack: Stack[VT], pos: int) -> VT:  # ~= 2n^2 + 3n - 8

    swap\_stack = Stack()  # 1

    stack\_size = stack.size  # 2

    for \_ in range(stack\_size - pos - 1):  # (n - pos - 1) \* (

        swap\_stack.push(stack.pop())  # k + 2 + n + 2

    # ) ~= n^2 + 4n - n\*pos - 4pos - n - 4

    el = stack.pop()  # n + 2

    for \_ in range(stack\_size - pos - 1):  # (n - pos - 1) \* (

        stack.push(swap\_stack.pop())  # k + 2 + n + 2

    # ) ~= n^2 + 4n - n\*pos - 4pos - n - 4

    return el

def swap(stack: Stack, pos1: int, pos2: int):  # 8n^2 + 14n - 12

    temp = pop\_by\_pos(stack, pos1)  # 2n^2 + 3n - 8

    push\_by\_pos(

        stack, pop\_by\_pos(stack, pos2 - 1), pos1

    )  # 2n^2 + 4n + 2 + 2n^2 + 3n - 8

    push\_by\_pos(stack, temp, pos2)  # 2n^2 + 4n + 2

def simple\_choice\_sort(

    stack: Stack,

) -> Stack:

    #  1 + n + log(n)\*4n^3 + log(n)\*6n^2 - log(n)\*16n - log(n)\*4n^2 - log(n)\*6n + log(n)16 + log(n)n - n + 8n^3 + 14n^2 - 12n =

    # = 4log(n)n^3 + 8n^3 + 2log(n)n^2 + 14n^2 - 22log(n)n - 12n + 16log(n) + 1

    """Sorts a list of integers in ascending order using the selection sort algorithm."""

    n = stack.size  # 1

    for i in range(n):  # n \* (

        min\_index = i  # 1

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

            if seek(stack, j) < seek(stack, min\_index):  # 4n^2 + 6n - 16

                min\_index = j  # 1

        swap(stack, i, min\_index)  # 8n^2 + 14n - 12

    return stack

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

    import time

    from random import randint

    tests = 10

    step = 50

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

        stack = Stack[int]()

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

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

        start\_time = time.time()

        stack = simple\_choice\_sort(stack)

        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: {stack.n\_op}")

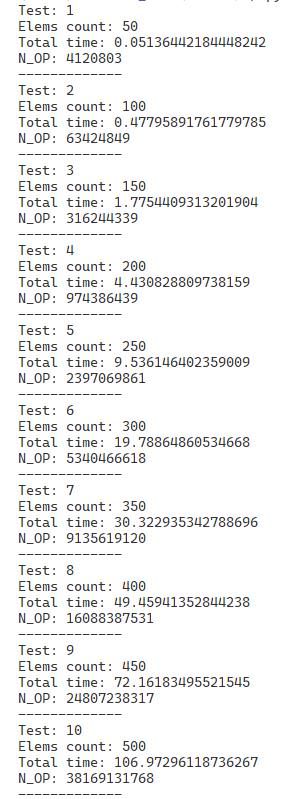
        print("-------------")

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | Т(n) (сек) | N\_op |
| 50 | 3878428 | 705482 | 0,0513 | 4120803 |
| 100 | 34832591 | 6643856 | 0,4779 | 63424849 |
| 150 | 125203809 | 24397263 | 1,7754 | 316244339 |
| 200 | 309738995 | 61150850 | 4,4308 | 974386439 |
| 250 | 624685555 | 124465379 | 9,5361 | 2397069861 |
| 300 | 1107395826 | 222178105 | 19,7886 | 5340466618 |
| 350 | 1796099112 | 362345676 | 30,3229 | 9135619120 |
| 400 | 2729752490 | 553206796 | 49,4594 | 16088387531 |
| 450 | 3947935309 | 803155811 | 72,1618 | 24807238317 |
| 500 | 5490770553 | 1120723036 | 106,9729 | 38169131768 |

|  |  |  |  |
| --- | --- | --- | --- |
| С1=F(n)/T(n) | С2=O(F(n))/T(n) | С3=F(n)/N\_op | С4=O(F(n))/N\_op |
| 75602893,48 | 13752086,23 | 0,941182686 | 0,171200133 |
| 72886776,94 | 13902189,14 | 0,549194697 | 0,104751628 |
| 70521464,87 | 13741840,19 | 0,395908458 | 0,077146877 |
| 69905884,92 | 13801311,17 | 0,317881061 | 0,062758313 |
| 65507445,96 | 13052021,21 | 0,260603817 | 0,051923968 |
| 55961302,28 | 11227580,76 | 0,207359376 | 0,041602751 |
| 59232432 | 11949571,99 | 0,196603984 | 0,039662958 |
| 55191783,36 | 11185068,89 | 0,169672224 | 0,034385472 |
| 54709490,47 | 11129930,39 | 0,159144491 | 0,032375865 |
| 51328612,7 | 10476700,51 | 0,143853693 | 0,029362026 |

**Таблица результата экспериментов и графики зависимостей**

**Скриншот работы программы:**



**Выводы.**

По результатам экспериментов было установлено, что алгоритм сортировки вставками для стека имеет n^3log(3) 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