|  |
| --- |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА – Российский технологический университет"**  **РТУ МИРЭА** |

Институт комплексной безопасности и специального приборостроения

Кафедра КБ-14 «Цифровые технологии обработки данных»

**ОТЧЕТ   
о выполнении лабораторной работы №1**

**«Реализация сортировки линейных структур данных»**

**по дисциплине   
«Программная реализация нелинейных структур»**

**Вариант № 22**

Выполнил: студент 2 курса

группы БСБО-11-21

шифр \_\_\_\_\_\_\_\_\_\_\_

Привалов Тимофей Александрович  
*(фио студента)*

Проверил:

Головин Леонид Олегович

Москва 2022 г.

**Задание на лабораторную работу № 1.**

В рамках лабораторной работы №1 требуется программно реализовать (с помощью массива (однонаправленных/двунаправленный динамический линенйый 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 (конструктур, 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 № 22.**

**Реализация связи элементов линейного списка:** Массив

**Способ организации линейного связанный список:** Стек

**Алгоритм сортировки:** Бинарная Вставка

**Теория о сортировках.**

**Сортировка бинарными вставками.**

На вход алгоритма подаётся последовательность {\displaystyle n}n чисел: a1, a2, …, аn {\displaystyle a\_{1},a\_{2},...,a\_{n}}. Сортируемые числа также называют *ключами*. Входная последовательность на практике представляется в виде массива с n {\displaystyle n} элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности a1’, a2’, …, аn’  {\displaystyle a\_{1}^{'},a\_{2}^{'},...,a\_{n}^{'}}, чтобы выполнялось следующее соотношение a1’ ≤ a2’ ≤ … ≤ аn’   {\displaystyle a\_{1}^{'}\leqslant a\_{2}^{'}\leqslant ...\leqslant a\_{n}^{'}}.В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма. Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей.



Пример работы алгоритма Б приведен на рисунке. Время выполнения алгоритма зависит от входных данных: чем большее множество нужно отсортировать, тем большее время потребуется для выполнения сортировки. Также на время выполнения влияет исходная упорядоченность массива. Время работы алгоритма для различных входных данных одинакового размера зависит от элементарных операций, или шагов, которые потребуется выполнить.

**Листинг программы с расчетами.**

import random

import datetime

class stack:

def \_\_init\_\_(self):

self.spisok = []

def create(self, n = 0):

for i in range(n):

self.spisok.append(random.randrange(1, 101))

def mid(self):

middle = len(self.spisok) // 2

if (len(self.spisok) % 2) != 0:

middle += 1

return middle

def lenght(self):

len\_spisok = 0

for i in self.spisok:

len\_spisok += 1

return len\_spisok

def pop(self, count = 0):# 2

count += 2

return self.spisok.pop()

def append(self, i = None, count = 0):# 1

count += 1

self.spisok.append(i)

return count

stack = stack()

stack\_help1 = []

stack\_help2 = []

stack\_help3 = []

stack\_help4 = []

stack\_help5 = []

count = 0

for n in range(300, 600, 300):

stack.create(n)

print(stack.spisok)

for i in range(stack.lenght() // 2):

stack\_help1.append(stack.pop(count))

for i in range(stack.lenght()):

stack\_help2.append(stack.pop(count))

stack\_help2 = stack\_help2[::-1]

start = datetime.datetime.now()

##################################################################################

for i in range(n-1): # n \* (2 +18 + 9n) = 20n+9n^2

if len(stack\_help3) == 0:

element1 = stack\_help1.pop()

element2 = stack\_help2.pop()

if element1 > element2:

stack\_help3.append(element1)

stack\_help3.append(element2)

else:

stack\_help3.append(element2)

stack\_help3.append(element1)

else:

if len(stack\_help4) == 0:

element1 = stack\_help1.pop()

element2 = stack\_help2.pop()

if element1 > element2:

stack\_help4.append(element1)

stack\_help4.append(element2)

else:

stack\_help4.append(element2)

stack\_help4.append(element1)

else:#2 + 2 \*( 3 + 1+ n/2\*(6)+ 3 + 1+ n/2\*(6 +3) = 2 + 2 \* (8 + 4,5n) = 18 + 9n

while len(stack\_help3) > 0: #2 +2 \*(

element3 = stack\_help3.pop() # 3

count += 1

for i in range(len(stack\_help4)): # 1+ n/2 \* (6

element\_sort = stack\_help4.pop() # 3

count += 1

if(element3 < element\_sort):#1

count += 1

stack\_help4.append(element\_sort) #1

stack\_help4.append(element3) #1

break

elif element3 == element\_sort: #1

count += 1

stack\_help4.append(element\_sort)#1

stack\_help4.append(element3)#1

break

else:

stack\_help5.append(element\_sort)

if len(stack\_help4) == 0:#2

count += 2

stack\_help4.append(element3)#1

while len(stack\_help5) > 0:#2 + n\*(

count += 2

stack\_help4.append(stack\_help5.pop())#3

#########################################################################################################

for i in range(len(stack\_help4)):

stack.append(stack\_help4.pop())

end = datetime.datetime.now()

print()

print('Сортировка №', n // 300)

print('F(n)= ', 20\*n+9\*n\*\*2)

print('O(F(n))= ', n \* log n)

print('T(n)= ', (end - start).microseconds/1000)

print('N\_op= ', count)

print(stack.spisok)

print()

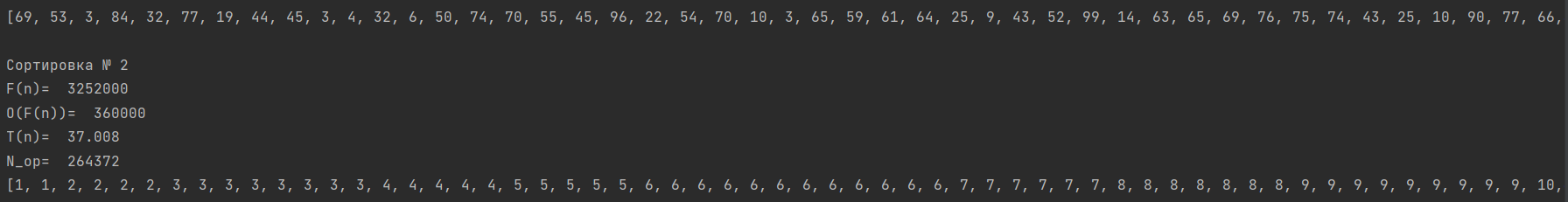
F(n) = 20n+9n^2

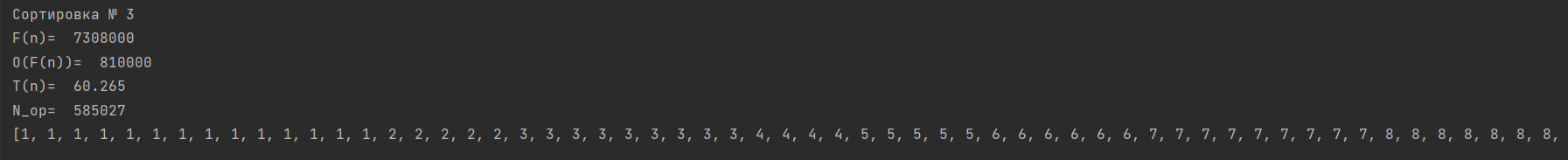
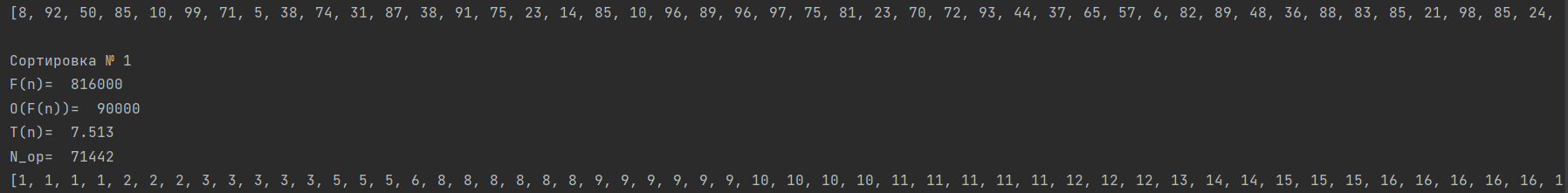
O(F(n)) = n^2

**Таблица результата экспериментов и графики зависимостей**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во элементов | F(n) | O(F(n)) | T(n)(мс) | operation |
| 300,00 | 816000,00 | 90000,00 | 7,03 | 70977,00 |
| 600,00 | 3252000,00 | 360000,00 | 22,31 | 261816,00 |
| 900,00 | 7308000,00 | 810000,00 | 49,14 | 594486,00 |
| 1200,00 | 12984000,00 | 1440000,00 | 84,28 | 1062072,00 |
| 1500,00 | 20280000,00 | 2250000,00 | 127,74 | 1636755,00 |
| 1800,00 | 29196000,00 | 3240000,00 | 191,69 | 2430075,00 |
| 2100,00 | 39732000,00 | 4410000,00 | 305,89 | 3282135,00 |
| 2400,00 | 51888000,00 | 5760000,00 | 363,76 | 4281549,00 |
| 2700,00 | 65664000,00 | 7290000,00 | 415,14 | 5331657,00 |
| 3000,00 | 81060000,00 | 9000000,00 | 563,27 | 6888663,00 |
| 3300,00 | 98076000,00 | 10890000,00 | 748,02 | 8059440,00 |
| 3600,00 | 116712000,00 | 12960000,00 | 777,31 | 9757512,00 |
| 3900,00 | 136968000,00 | 15210000,00 | 864,49 | 11286285,00 |
| 4200,00 | 158844000,00 | 17640000,00 | 954,43 | 13225440,00 |
| 4500,00 | 182340000,00 | 20250000,00 | 1125,99 | 14980224,00 |

|  |  |  |  |
| --- | --- | --- | --- |
| C1=F(n)/T(n) | C2=O(F(n))/T(n) | C3=F(n)/operation | C4= O(F(n))/operation |
| 116107,00 | 12805,92 | 11,50 | 1,27 |
| 145757,70 | 16135,54 | 12,42 | 1,38 |
| 148733,08 | 16485,19 | 12,29 | 1,36 |
| 154052,42 | 17085,30 | 12,23 | 1,36 |
| 158758,74 | 17613,77 | 12,39 | 1,37 |
| 152310,00 | 16902,47 | 12,01 | 1,33 |
| 129891,10 | 14417,09 | 12,11 | 1,34 |
| 142644,68 | 15834,75 | 12,12 | 1,35 |
| 158173,53 | 17560,38 | 12,32 | 1,37 |
| 143910,44 | 15978,21 | 11,77 | 1,31 |
| 131113,79 | 14558,40 | 12,17 | 1,35 |
| 150149,36 | 16672,97 | 11,96 | 1,33 |
| 158438,66 | 17594,27 | 12,14 | 1,35 |
| 166429,00 | 18482,33 | 12,01 | 1,33 |
| 161937,50 | 17984,17 | 12,17 | 1,35 |





**Скриншоты работы программы**

**Выводы.**

По результатам экспериментов было установлено, что графики C1, C2, C3, C4 от N имеют линейную зависимость от количества элементов.

Достоинства данного алгоритма сортировки: Не имеет «трудных» входных данных, сохраняет порядок равных элементов

Недостатки данного алгоритма сортировки: На «почти отсортированных» массивах работает столь же долго, как на хаотичных, плохая оптимизация.

Неплохо работает в [параллельном](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) варианте: легко разбить задачи между процессорами поровну, но трудно сделать так, чтобы другие процессоры взяли на себя работу, в случае если один процессор задержится.

**Литература:**

1. Структуры данных и алгоритмы. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. – М.: Издательский дом «Вильямс», 2000

2. Д. Кнут. Искусство программирования для ЭВМ.

**Приложение 1. Применение счетчика операций N\_op.**

import random

import datetime

class stack:

def \_\_init\_\_(self):

self.spisok = []

def create(self, n = 0):

for i in range(n):

self.spisok.append(random.randrange(1, 101))

def mid(self):

middle = len(self.spisok) // 2

if (len(self.spisok) % 2) != 0:

middle += 1

return middle

def lenght(self):

len\_spisok = 0

for i in self.spisok:

len\_spisok += 1

return len\_spisok

def pop(self, count = 0):# 2

count += 2

return self.spisok.pop()

def append(self, i = None, count = 0):# 1

count += 1

self.spisok.append(i)

return count

stack = stack()

stack\_help1 = []

stack\_help2 = []

stack\_help3 = []

stack\_help4 = []

stack\_help5 = []

count = 0

for n in range(900, 1200, 300):

stack.create(n)

print(stack.spisok)

for i in range(stack.lenght() // 2):

stack\_help1.append(stack.pop(count))

for i in range(stack.lenght()):

stack\_help2.append(stack.pop(count))

stack\_help2 = stack\_help2[::-1]

start = datetime.datetime.now()

##################################################################################

for i in range(n-1): # n \* (2 +18 + 9n) = 20n+9n^2

if len(stack\_help3) == 0:

element1 = stack\_help1.pop()

element2 = stack\_help2.pop()

if element1 > element2:

stack\_help3.append(element1)

stack\_help3.append(element2)

else:

stack\_help3.append(element2)

stack\_help3.append(element1)

else:

if len(stack\_help4) == 0:

element1 = stack\_help1.pop()

element2 = stack\_help2.pop()

if element1 > element2:

stack\_help4.append(element1)

stack\_help4.append(element2)

else:

stack\_help4.append(element2)

stack\_help4.append(element1)

else:#2 + 2 \*( 3 + 1+ n/2\*(6)+ 3 + 1+ n/2\*(6 +3) = 2 + 2 \* (8 + 4,5n) = 18 + 9n

while len(stack\_help3) > 0: #2 +2 \*(

element3 = stack\_help3.pop() # 3

count += 1

for i in range(len(stack\_help4)): # 1+ n/2 \* (6

element\_sort = stack\_help4.pop() # 3

count += 1

if(element3 < element\_sort):#1

count += 1

stack\_help4.append(element\_sort) #1

stack\_help4.append(element3) #1

break

elif element3 == element\_sort: #1

count += 1

stack\_help4.append(element\_sort)#1

stack\_help4.append(element3)#1

break

else:

stack\_help5.append(element\_sort)

if len(stack\_help4) == 0:#2

count += 2

stack\_help4.append(element3)#1

while len(stack\_help5) > 0:#2 + n\*(

count += 2

stack\_help4.append(stack\_help5.pop())#3

#########################################################################################################

for i in range(len(stack\_help4)):

stack.append(stack\_help4.pop())

end = datetime.datetime.now()

print()

print('Сортировка №', n // 300)

print('F(n)= ', 20\*n+9\*n\*\*2)

print('O(F(n))= ', n\*\*2)

print('T(n)= ', (end - start).microseconds/1000)

print('N\_op= ', count)

print(stack.spisok)

print()