**“Санкт-Петербургский политехнический университет Петра Великого”**

Институт компьютерных наук и технологий

**ОТЧЕТ**

**По Расчетно-графической работе №2**

по дисциплине «Теория и технологии программирования»

**Выполнил:**

студент группы з3530902/00002 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Загидулин М.В.

**Проверил:**

Доцент, кандидат технических наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Хлопин С.В.

Санкт-Петербург 2021 г.

**Задание**

В соответствии с индивидуальным заданием необходимо:

1. Создать программу для сортировки массива случайных данных заданными методами сортировки.
2. Провести серию экспериментов с разным размером сортируемых данных. Провести анализ изменения зависимости числа сравнений и перестановок элементов при сортировках от размера сортируемого массива. Построить таблицы и графики изменения числа сравнений и перестановок от размеров массивов сортировки. Сравнение производить по абсолютным значения и по нормированным или приведенным к логарифмическому виду (в соответствии с персональным заданием) величинам.
3. Построить блок-схему алгоритма сортировки, соответствующую алгоритму кода программы.

Индивидуальное задание:

1. Сортировки: сортировка «пузырьковая», сортировка «отбор (линейная)».
2. Размеры массивов для экспериментов: 25;43;79;160;362;909;2457.
3. Блок-схема: сортировка пузырьковая.

**Исходный код программы**

На рисунке 1-2 представлен исходный код программы.

from sorts.sorts import SortClass  
  
sort = SortClass([25, 43, 79, 160, 362, 909, 245, 725, 43, 79, 160, 362, 909, 2457])  
  
# Bubble sort  
sort.bubble\_sort()  
  
# Discarded Sort (Linear Sort)  
sort.linear\_sort()

Рисунок 1 – Исходный код программы – Исполняемый файл

from random import randint  
  
  
class SortClass:  
 arrays\_size = []  
 # arrays\_size = [25, 79, 160, 245, 362, 725, 909, 2457]  
 results = {}  
  
 def \_\_init\_\_(self, arrays\_size):  
 self.arrays\_size = arrays\_size  
 print('Starting program...')  
  
 def arrays\_generator(self):  
 raw\_array = []  
 for i in range(len(self.arrays\_size)):  
 raw\_array.append([randint(1, 99999) for \_ in range(self.arrays\_size[i])])  
 return raw\_array  
  
 def sort\_init(self, sort\_name):  
 self.results[sort\_name] = {}  
 self.results[sort\_name]['raw\_arrays'] = self.arrays\_generator()  
 self.results[sort\_name]['result\_arrays'] = []  
 self.results[sort\_name]['iterations\_count'] = []  
 self.results[sort\_name]['comparison\_count'] = []  
 self.results[sort\_name]['arr\_len'] = []  
 self.results[sort\_name]['name'] = sort\_name  
  
 def bubble\_sort(self):  
 print('Bubble sort')  
 sort\_name = 'bubble\_sort'  
 self.sort\_init(sort\_name)  
  
 for sort\_cycle in range(len(self.results[sort\_name]['raw\_arrays'])):  
 print('Bubble sort started!', f'Array - {sort\_cycle + 1}', sep='\n')  
 bubble\_list = self.results[sort\_name]['raw\_arrays'][sort\_cycle]  
 iterations\_count = 0  
 comparison\_count = 0  
 for i in range(len(bubble\_list) - 1):  
 for j in range(0, len(bubble\_list) - i - 1):  
 comparison\_count += 1  
 if bubble\_list[j] > bubble\_list[j + 1]:  
 bubble\_list[j], bubble\_list[j + 1] = bubble\_list[j + 1], bubble\_list[j]  
 iterations\_count += 1  
 print('Bubble sort finished!', f'Result - {bubble\_list}', f'Number of iterations - {iterations\_count}',  
 f'Number of comparison - {comparison\_count}', sep='\n', end='\n\n')  
 self.results[sort\_name]['result\_arrays'].append(bubble\_list)  
 self.results[sort\_name]['iterations\_count'].append(iterations\_count)  
 self.results[sort\_name]['comparison\_count'].append(comparison\_count)  
  
 def linear\_sort(self):  
 print('Linear Sort')  
 sort\_name = 'linear\_sort'  
 self.sort\_init(sort\_name)  
  
 for sort\_cycle in range(len(self.results[sort\_name]['raw\_arrays'])):  
 print('Linear sort started!', f'Array - {sort\_cycle + 1}', sep='\n')  
 linear\_list = self.results[sort\_name]['raw\_arrays'][sort\_cycle]  
 iterations\_count = 0  
 comparison\_count = 0  
 for i in range(len(linear\_list) - 1):  
 for j in range(i + 1, len(linear\_list)):  
 comparison\_count += 1  
 if linear\_list[j] < linear\_list[i]:  
 linear\_list[i], linear\_list[j] = linear\_list[j], linear\_list[i]  
 iterations\_count += 1  
 print('Linear Sort finished!', f'Result - {linear\_list}', f'Number of iterations - {iterations\_count}',  
 f'Number of comparison - {comparison\_count}', sep='\n', end='\n\n')  
 self.results[sort\_name]['result\_arrays'].append(linear\_list)  
 self.results[sort\_name]['iterations\_count'].append(iterations\_count)  
 self.results[sort\_name]['comparison\_count'].append(comparison\_count)  
  
 def get\_result\_list(self, sort\_name):  
 try:  
 return self.results[sort\_name]  
 except KeyError:  
 print(f"{sort\_name} doesn't exist!")

Рисунок 2 – Исходный код программы – Класс сортировки

**Результаты сортировки**

Таблица 1 – Число сравнений и перестановок для различных методов сортировк

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод сортировки | Размер массива | 25 | 79 | 160 | 245 | 362 | 725 | 909 | 2457 |
| Пузырьковая | Сравнения | 300 | 3081 | 12720 | 29890 | 65341 | 262450 | 412686 | 3017196 |
| Перестановки | 161 | 1435 | 6873 | 14690 | 31702 | 128479 | 204879 | 1506144 |
| Отбор (линейная) | Сравнения | 300 | 3081 | 12720 | 29890 | 65341 | 262450 | 412686 | 3017196 |
| Перестановки | 150 | 1443 | 6353 | 13515 | 31962 | 132283 | 208145 | 1484933 |

График 1 - График изменения числа сравнений в зависимости от размера сортируемых данных для различных методов сортировки

График 2 - Графики изменения числа перестановок в зависимости от размера сортируемых данных для различных методов сортировки

**Блок-схема алгоритма**

На рисунке 3 представлена блок-схема алгоритма сортировки пузырьком.

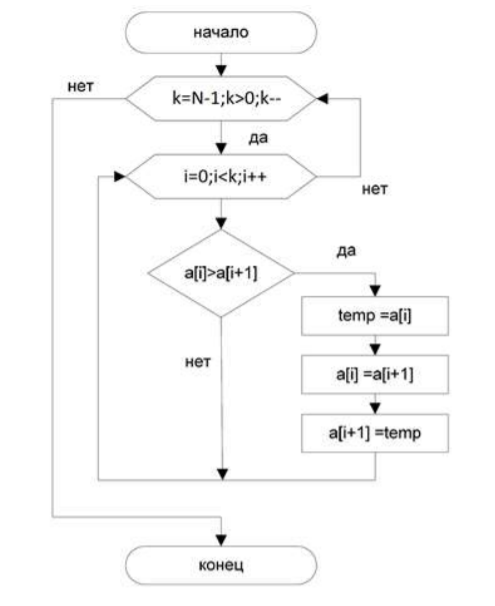


Рисунок 3 – Блок-схема алгоритма сортировки пузырьком

**Вывод**

В ходе выполнения расчетно-графической работы был проведен анализ изменения зависимости числа сравнений и перестановок элементов при сортировках от размера сортируемого массива. В качестве объектов исследования выступали алгоритмы сортировки «пузырьком» и сортировка «отбором (линейная)»..

Как видно на графиках, кол-во сравнений у этих графиков – одинаковое.  
А кол-во перестановок у сортировки отборного типа незначительно ниже, чем у сортировки пузырьком.

Из этого можно сделать вывод, что при сортировке, оба представленных типа, нагружают систему одинаково