

**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**(МИИТ)**

|  |
| --- |
| **ИНСТИТУТ ТРАНСПОРТНОЙ ТЕХНИКИ И СИСТЕМ УПРАВЛЕНИЯ**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Кафедра «Управление и защита информации»

**Лабораторная работа №4**

**«Реализация алгоритма сортировки слиянием»**

**по дисциплине**

**«Методы программирования»**

**Выполнил:** студент группы ТКИ-311

Куминов В. П.

**Проверил:** к.т.н., доц. Логинова Л. Н.,

к.т.н., доц. Сафронов А. И.

**Москва – 2022 г.**

**1. Цель работы**

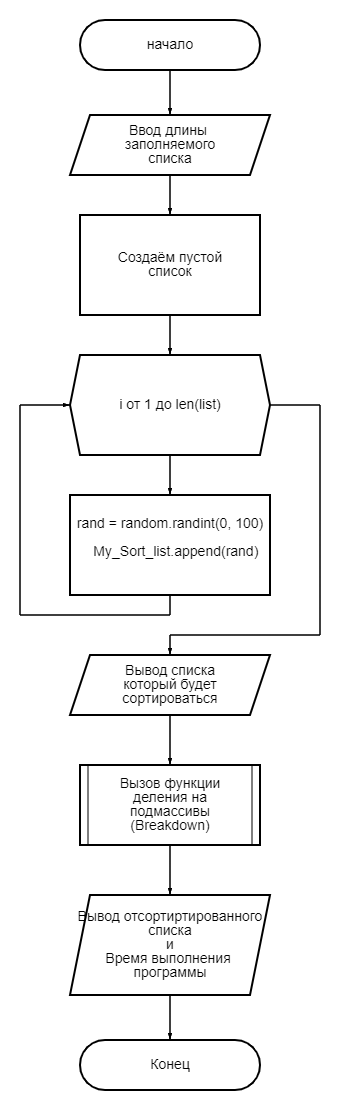
Научиться реализовывать алгоритм сортировки слиянием, по заданному псевдокоду.

**2. Формулировка задачи**

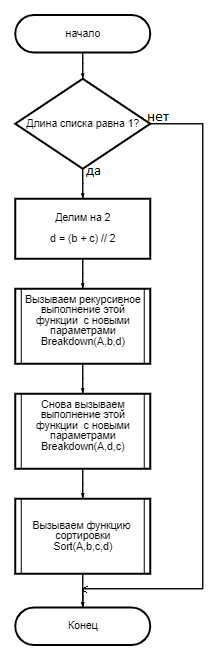
1. Реализуйте алгоритм сортировки слиянием по заданному псевдокоду.

**3. Блок схема**

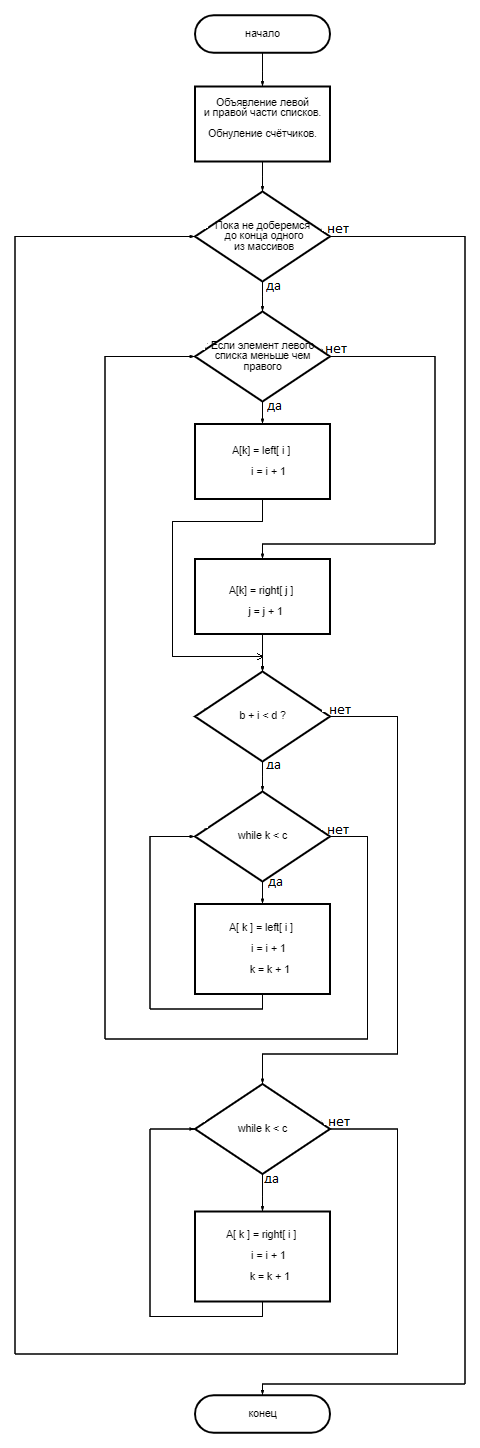
Основная программа



Метод разбиения списка, на списки длинной 1



Функция сортировки



**4. Подбор тестовых примеров**

4.1 Вводим длину нашего списка (6)

4.2. Получаем список (3, 4, 8, 2, 1, 5)

4.3. Разбиваем наш список на список длиной 1

4.4. Сравниваем 2 соседних списка (3) и (4)

4.5. 3 меньше 4? (Да)

4.6. Вносим эти два списка в новый список без изменений (3, 4)

4.7. Сравниваем следующие два соседних списка (8) и (2)

4.8. 2 Меньше 8? (Нет)

4.9. Вносим эти два списка в новый список, сначала 2, потом 8 (2, 8)

4.10. Сравниваем следующие два соседних списка (1) и (5)

4.11. 1 Меньше 5? (Да)

4.6. Вносим эти два списка в новый список, сначала 2, потом 8 (2, 8)

4.10. Получаем следующие списки (3, 4), (2, 8), (1, 5)

4.11. Сравниваем следующие два соседних списка (3, 4) и (2, 8)

4.12. Сравниваем нулевой элемент первого списка (3, 4) с нулевым элементом соседнего списка (2, 8)

4.13. 3 Меньше 2? (Нет)

4.14. Вносим число 2 в новый список.

4.15. Сравниваем нулевой элемент первого списка (3, 4) с нулевым элементом соседнего списка (8)

4.16. 3 Меньше 8? (Да)

4.17. Вносим число 3 в список.

4.18. Сравниваем нулевой элемент первого списка (4) с нулевым элементом соседнего списка (8)

4.19. 4 Меньше 8? (Да)

4.20. Вносим эти два списка в новый список, сначала 4, потом 8

4.21. Получаем список (2, 3, 4, 8)

4.22. Сравниваем нулевой элемент первого списка (2, 3, 4, 8) с нулевым элементом соседнего списка (1, 5)

4.23. 2 Меньше 1? (Нет)

4.24. Вносим число 1 в результирующий список

4.25. Сравниваем нулевой элемент первого списка (2, 3, 4, 8) с нулевым элементом соседнего списка (5)

4.26. 2 Меньше 5? (Да)

4.27. Вносим число 2 в результирующий список

4.28. Сравниваем нулевой элемент первого списка (3, 4, 8) с нулевым элементом соседнего списка (5)

4.29. 3 Меньше 5? (Да)

4.30. Вносим число 3 в результирующий список

4.31. Сравниваем нулевой элемент первого списка (4, 8) с нулевым элементом соседнего списка (5)

4.32. 4 Меньше 5? (Да)

4.33. Вносим число 4 в результирующий список

4.34. Сравниваем нулевой элемент первого списка (8) с нулевым элементом соседнего списка (5)

4.35. 8 Меньше 5? (Нет)

4.36. Вносим число 5 в результирующий список и за иссяканием одного из списков, сравниваемый с оставшимися элементами вносится в результирующий список целиком.

4.37. Получаем отсортированный массив (1, 2, 3, 4, 5, 8)

**5. Код программы**

import time

import random

# Функция деления списка на списки длиной 1

def Breakdown(A, b, c):

if c - b > 1:

d = (b + c)//2

Breakdown(A, b, d)

Breakdown(A, d, c)

Sort(A, b, c, d)

# Функция сортировки

def Sort(A, b, c, d):

left = A[b:d]

right = A[d:c]

k = b

i = 0

j = 0

while (b + i < d and d + j < c):

# Если элемент из левого списка меньше чем из правого, тогда:

if (left[i] <= right[j]):

A[k] = left[i]

i = i + 1

# Иначе помещаем элемент из правого списка в отсортированный

else:

A[k] = right[j]

j = j + 1

k = k + 1

# Когда добрались до конца левого массива

if b + i < d:

while k < c:

A[k] = left[i]

i = i + 1

k = k + 1

# Когда добрались до конца правого массива

else:

while k < c:

A[k] = right[j]

j = j + 1

k = k + 1

# Создание списка случайных элементов

l = int(input("Введите длину сортируемого списка: "))

My\_Sort\_list = []

for i in range(0, l):

rand = random.randint(0, 100)

My\_Sort\_list.append(rand)

print("Будет отсортирован следующий список: ", My\_Sort\_list)

# Вызываем функцию деления списка на списки длиной 1

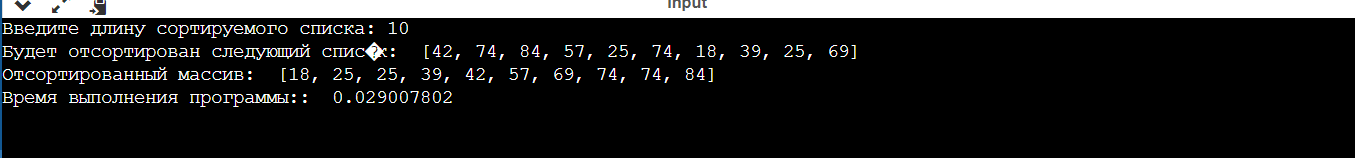
Breakdown(My\_Sort\_list, 0, len(My\_Sort\_list))

# Выводим отсортированный список

print("Отсортированный массив: ", My\_Sort\_list)

print("Время выполнения программы:: ", time.process\_time())

**6. Расчёт тестовых примеров на ПК**

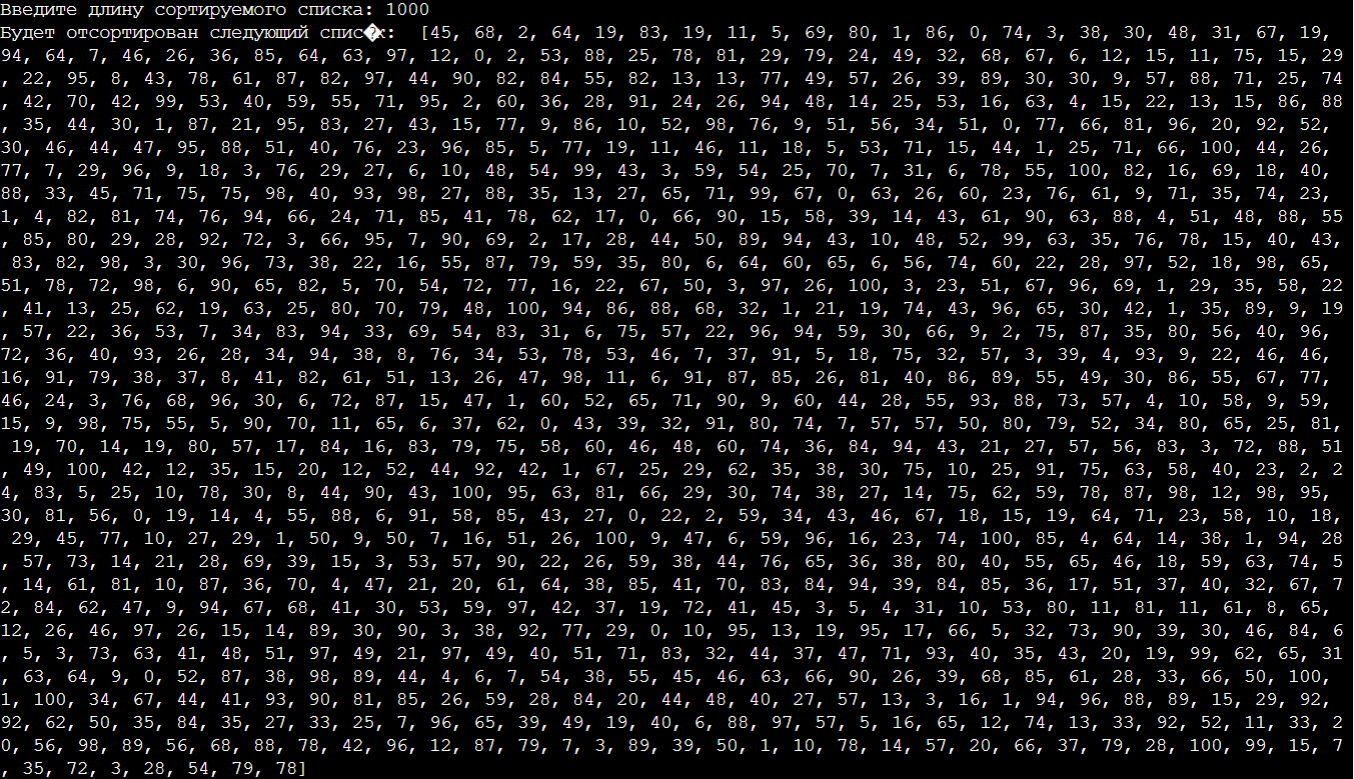
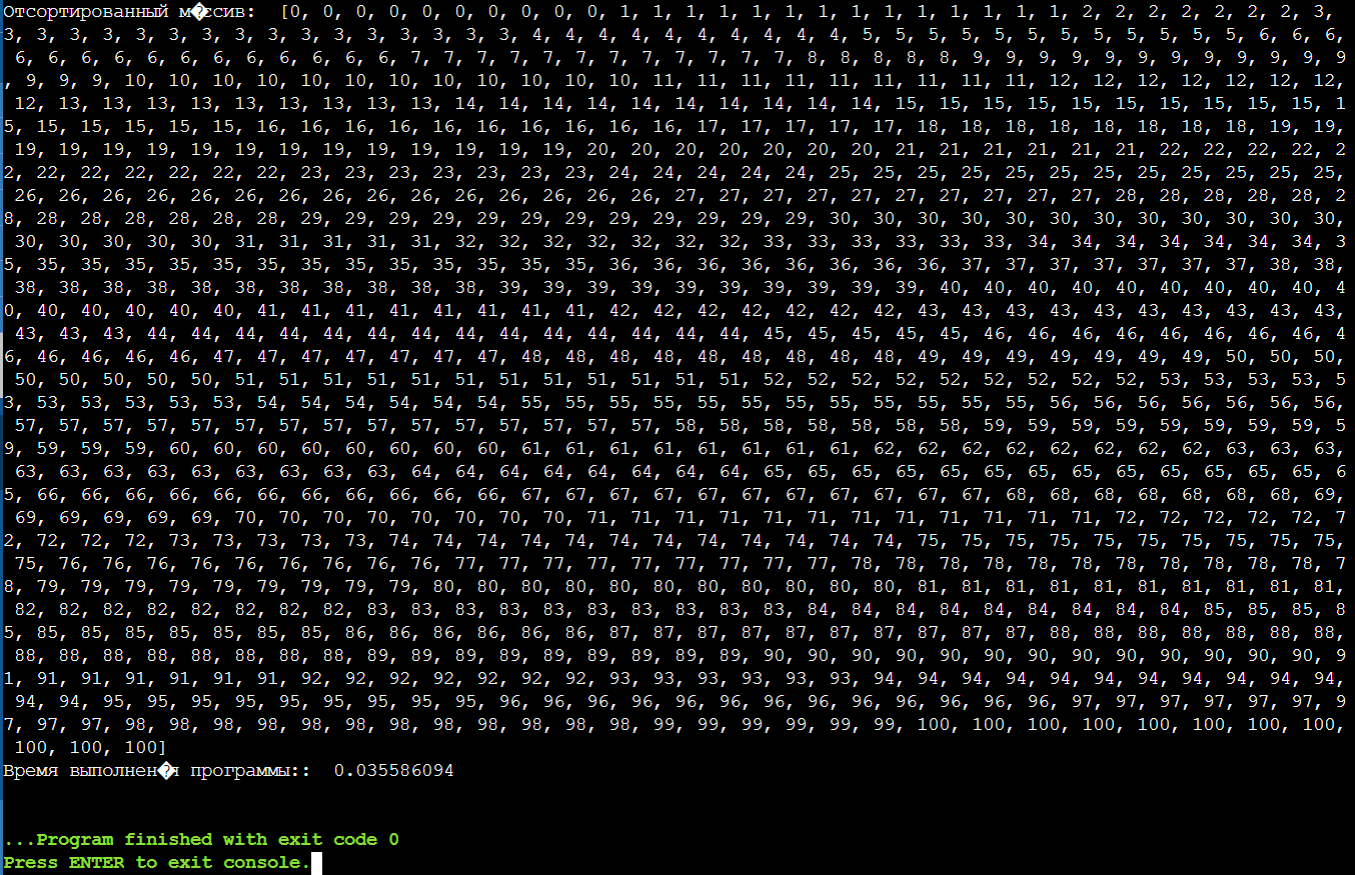


10 элементов

Изображение выглядит как текст

Автоматически созданное описание

100 элементов

1000 элементов

**7. Вывод**

В данной практической работе я научился реализовывать псевдокод алгоритма сортировки слиянием на выбранном мной языке программирования – *Python*. При реализации кода, проблем не возникло. Однако, пришлось вспомнить как работать с функциями на языке *Python*, поскольку алгоритм сортировки слиянием основан на рекурсиях. По поводу времени выполнения программы: Я наблюдаю, что 10 элементов сортируются быстрее, чем 100, а 100 быстрее 1000 элементов. Ответ оказался ожидаем. Реализация метода сортировки слиянием оказалась несколько сложнее, чем реализация метода сортировки вставкой. Мне кажется, это связанно с особенностями данного алгоритма сортировки, поскольку он требует выполнения большего количества итераций (это можно увидеть в тестовом примере). Нужно разбивать списки на списки меньшей длины, а потом собирать эти маленькие списки заново, как мозаику, но уже в сортируемом порядке, в то время как при сортировке вставкой работа идёт непосредственно в одном списке.