МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
 РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный университет геодезии и картографии»  
(МИИГАиК)  
Факультет геоинформатики и информационной безопасности  
Кафедра геоинформационных систем и технологий

**Лабораторная работа №3**  
**«Алгоритмы сортировки»**

Проверил: Выполнил:

Лебедев Е.Д. Студент группы: 2024-ФГиИБ-ПИ-1б

Николаев М.А.

Москва 2025

**Цель работы**

Выполнить предоставленные задачи на языке **Python**.

**Задание (вариант №19)**

Произвести сортировки используя:

* Поразрядная сортировка
* Сортировка Шелла
* Быстрая сортировка

**Анализ задания**

*Входные данные:*

На вход программа получает по умолчанию размерность массива в 1000000, которую при желании можно изменить

*Промежуточные данные:*

-

*Выходные данные:*

На выходе программа выдаёт первые 20 элементов оригинального и отсортированного массивов и время, потребовавшееся на сортировку.

**Листинг программы**Ссылка на **GitHub**: <https://github.com/Ep1cMax/Second_Semester/tree/main/Works>

***HM3\_Radix.py***

import random

import time

from typing import List

def combine\_buckets(buckets: List[List[int]]) -> List[int]:

combined\_array = []

for bucket in buckets:

combined\_array.extend(bucket)

return combined\_array

def sort\_by\_digit(arr: List[int], digit\_place: int) -> None:

buckets = [[] for \_ in range(10)]

for num in arr:

bucket\_index = (num // digit\_place) % 10

buckets[bucket\_index].append(num)

arr[:] = combine\_buckets(buckets)

def radix\_sort(arr: List[int]) -> None:

max\_val = max(arr)

digit\_place = 1

while max\_val // digit\_place > 0:

sort\_by\_digit(arr, digit\_place)

digit\_place \*= 10

def print\_first\_elements(arr: List[int], count: int = 20) -> None:

print(" ".join(map(str, arr[:count])))

random.seed(time.time())

N = 1000000 # Больше достаточно сильно замедляет программу

RANGE = N

arr = [random.randint(1, RANGE) for \_ in range(N)]

print("First 20 elements of the original array:")

print\_first\_elements(arr)

start\_time = time.time()

radix\_sort(arr)

end\_time = time.time()

duration = (end\_time - start\_time) \* 1000

print("\nFirst 20 elements of the sorted array:")

print\_first\_elements(arr)

print(f"\nTime taken by the Radix sort: {duration:.2f} ms")

***HM3\_Quick.py***

import random

import time

from typing import List

def quick\_sort(arr: List[int], low: int, high: int) -> None:

if low >= high:

return

random\_index = random.randint(low, high)

randArr = arr[random\_index]

arr[random\_index], arr[low] = arr[low], arr[random\_index]

lowerThanRandArr = low

greaterThanRandArr = high

i = low + 1

while i <= greaterThanRandArr:

if arr[i] < randArr:

arr[i], arr[lowerThanRandArr] = arr[lowerThanRandArr], arr[i]

i += 1

lowerThanRandArr += 1

elif arr[i] > randArr:

arr[i], arr[greaterThanRandArr] = arr[greaterThanRandArr], arr[i]

greaterThanRandArr -= 1

else:

i += 1

quick\_sort(arr, low, lowerThanRandArr - 1)

quick\_sort(arr, greaterThanRandArr + 1, high)

def print\_first\_elements(arr: List[int], count: int = 20) -> None:

print(" ".join(map(str, arr[:count])))

random.seed(time.time())

N = 1000000 # Больше достаточно сильно замедляет программу

RANGE = N

arr = [random.randint(1, RANGE) for \_ in range(N)]

print("First 20 elements of the original array:")

print\_first\_elements(arr)

start\_time = time.time()

quick\_sort(arr, 0, len(arr) - 1)

end\_time = time.time()

duration = (end\_time - start\_time) \* 1000

print("\nFirst 20 elements of the sorted array:")

print\_first\_elements(arr)

print(f"\nTime taken by Quick sort: {duration:.2f} ms")

***HM3\_Shell.py***

import random

import time

from typing import List

def shell\_sort(arr: List[int]) -> None:

n = len(arr)

step = n // 2

while step > 0:

for i in range(step, n):

temp = arr[i]

j = i

while j >= step and arr[j - step] > temp:

arr[j] = arr[j - step]

j -= step

arr[j] = temp

step = step // 2

def print\_first\_elements(arr: List[int], count: int = 20) -> None:

print(" ".join(map(str, arr[:count])))

random.seed(time.time())

N = 100000 # Больше достаточно сильно замедляет программу

RANGE = N

arr = [random.randint(1, RANGE) for \_ in range(N)]

print("Первые 20 элементов исходного массива:")

print\_first\_elements(arr)

start\_time = time.time()

shell\_sort(arr)

end\_time = time.time()

duration = (end\_time - start\_time) \* 1000

print("\nПервые 20 элементов отсортированного массива:")

print\_first\_elements(arr)

print(f"\nВремя выполнения сортировки Шелла: {duration:.2f} мс")

**Результаты работы программы:**

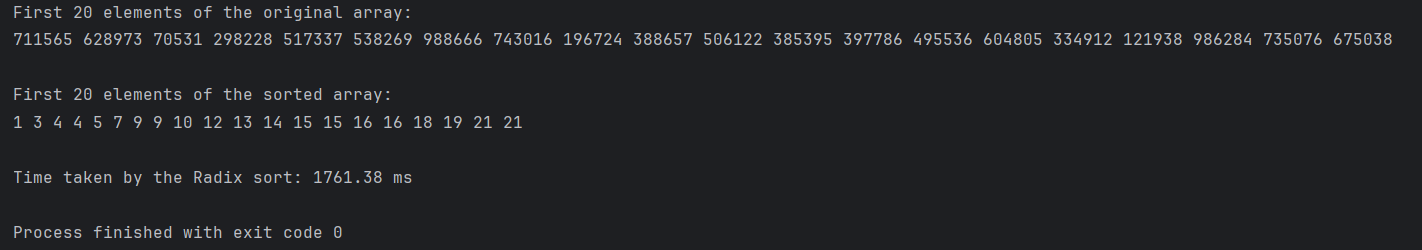


Рис. 3. – Работа Radix\_Sort

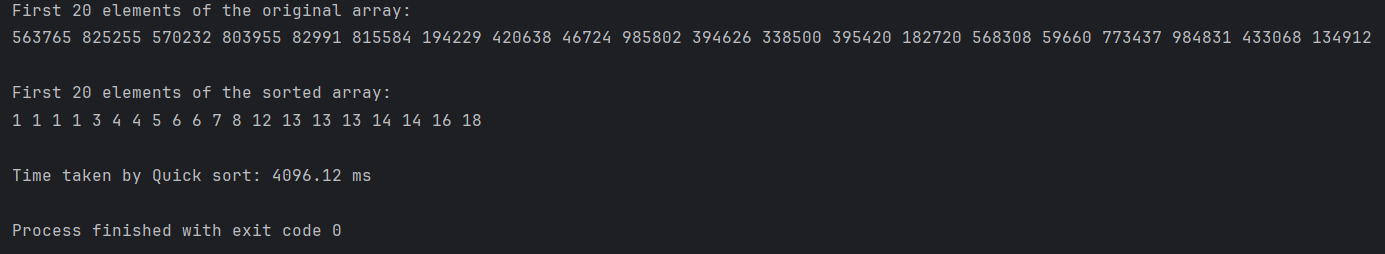


Рис. 2 – Работа Quick\_Sort

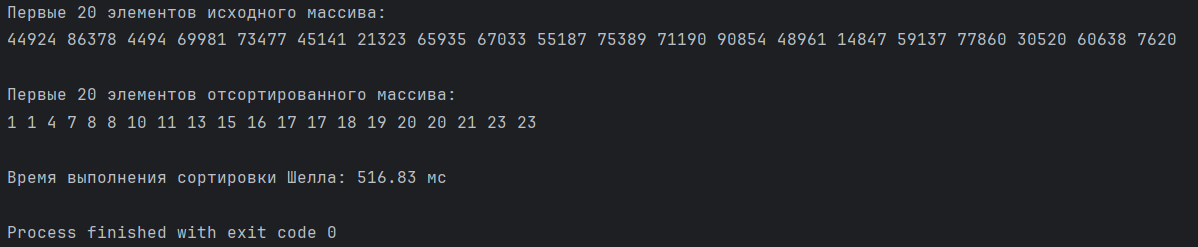


Рис. 3. – Работа Shell\_Sort

**Блок-схема**

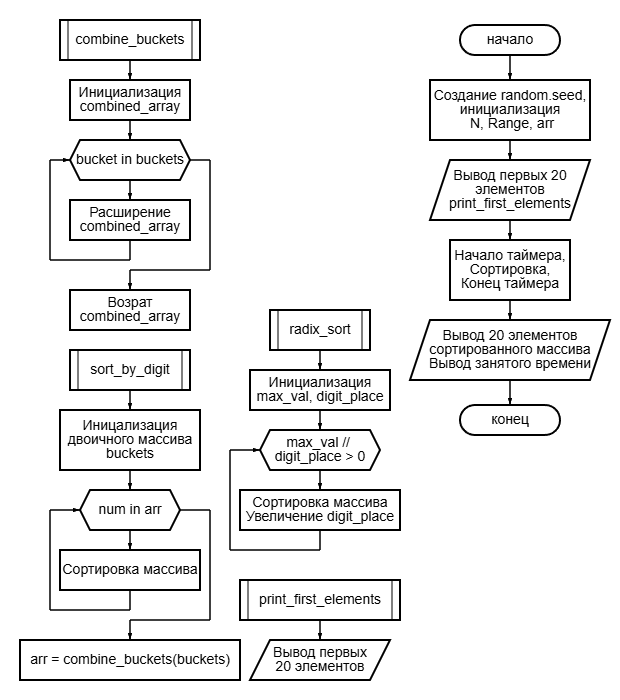


Рис. 4. – Блок-схема Radix\_Sort

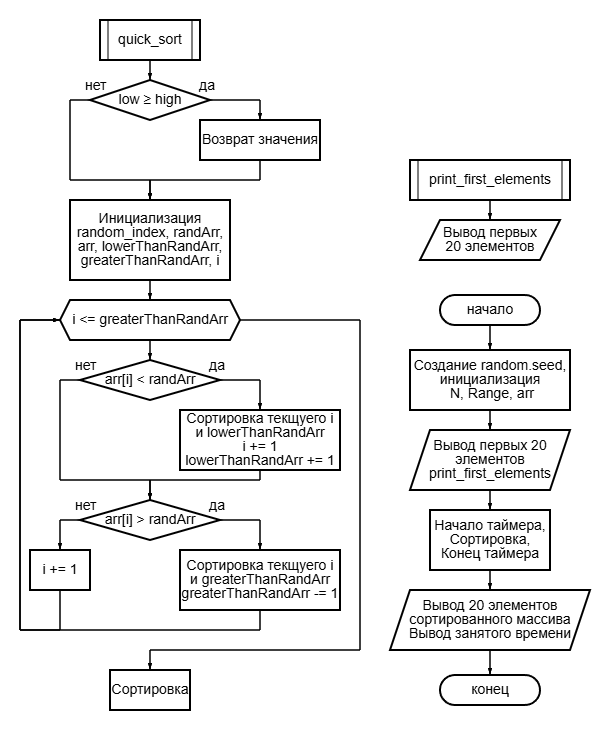


Рис. 5. – Блок-схема Quick\_Sort

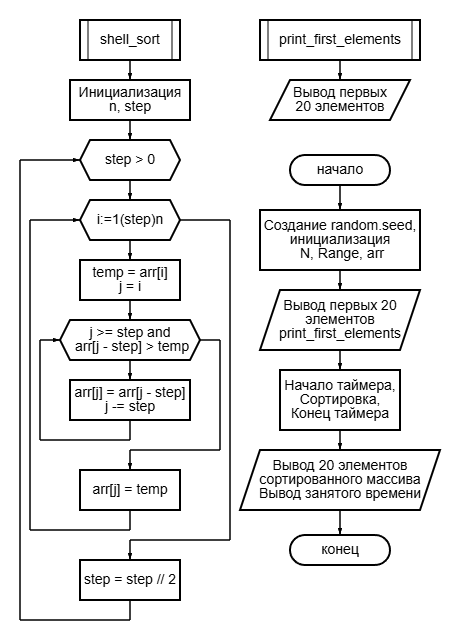


Рис. 6. – Блок-схема Shell\_sort

**Асимптотическая оценка**

Поразрядная сортировка:

* Лучший случай – O(n)
* Средний случай – O(n)
* Худший случай – O(n)

Сортировка Шелла

* Лучший случай – O(n \* log(n))
* Средний случай – O(n4/3)
* Худший случай – O(n2)

Быстрая сортировка

* Лучший случай – O(n \* log(n))
* Средний случай – O(n \* log(n))
* Худший случай – O(n2)

В лучшем/среднем случае кол-во подзадач:

(а): 2; во сколько раз уменьшается количество данных при вызове

(b): 2; время работы не рекурсивного случая

(c): 1

По мастер-теореме:

c = logba (1 = log22 = 1), тогда T(n) = O(nс\*log(n)) = O(n\*log(n))

В худшем случае массив делится неравномерно, из-за чего увеличивается время работы не рекурсивного случая (c): 2. Тогда:

c > logba (2 > log22 = 1), следовательно T(n) = O(nc) = O(n2).

|  |  |  |  |
| --- | --- | --- | --- |
| длина массива | сорт. Шелла, мс | поразрядная сорт., мс | быстрая сорт., мс |
| 10 | 0,00 | 0,00 | 0,00 |
| 100 | 0,00 | 0,00 | 0,00 |
| 1000 | 0,00 | 1,02 | 1,02 |
| 10000 | 31,08 | 3,03 | 21,70 |
| 100000 | 555,52 | 97,22 | 295,55 |
| 200000 | 1341,42 | 268,84 | 687,17 |
| 300000 | 2081,83 | 397,88 | 1129,97 |
| 400000 | 3387,79 | 540,44 | 1383,34 |
| 500000 | 4214,37 | 683,40 | 1899,22 |

Табл. 1. – Таблица времени сортировки при разной размерности

График 1. – Зависимость скорости сортировки от размерности массива

**Вывод**

Массивы успешно были отсортированы, используя разные методы, которые были асимптотически оценены и графически показаны с явной зависимостью времени от размерности.