1)Блочная (корзинная) сортировка

from math import sqrt

from typing import List, Iterable, TypeVar

T = TypeVar("T", int, float)

def insertion\_sort(a: List[T]) -> None:

"""

Стабильная сортировка вставками (in-place) для содержимого одной корзины.

В среднем O(k^2) на корзину, где k — размер корзины.

"""

for i in range(1, len(a)):

key = a[i]

j = i - 1

# Сдвигаем элементы вправо, пока не найдём позицию для key

while j >= 0 and a[j] > key:

a[j + 1] = a[j]

j -= 1

a[j + 1] = key

def bucket\_sort(arr: Iterable[T], bucket\_count: int | None = None) -> List[T]:

"""

Блочная (корзинная) сортировка.

Идея: распределяем элементы по корзинам по диапазону значений -> сортируем каждую корзину ->

склеиваем корзины слева направо.

:param arr: итерируемая последовательность чисел (int/float), допускаются отрицательные значения

:param bucket\_count: количество корзин; по умолчанию ~ sqrt(n) — сбалансированная эвристика

:return: новый отсортированный список

"""

a = list(arr) # не мутируем вход

n = len(a)

if n <= 1:

return a.copy()

# Эвристика: число корзин ~ sqrt(n) (можно переопределить параметром)

if bucket\_count is None:

bucket\_count = max(1, int(sqrt(n)))

# Находим диапазон значений

mn = min(a)

mx = max(a)

if mn == mx:

# Все элементы равны — уже отсортировано

return a.copy()

rng = mx - mn

# Создаём пустые корзины

buckets: List[List[T]] = [[] for \_ in range(bucket\_count)]

# Функция распределения: нормализуем x в [0, 1] и масштабируем в индексы [0 .. bucket\_count-1]

# Важно: крайние значения (особенно mx) должны попадать в последнюю корзину.

for x in a:

# При rng > 0 значение ((x - mn) / rng) в диапазоне [0, 1].

# Умножаем на (bucket\_count - 1), чтобы верхняя граница ушла в последнюю корзину.

idx = int((x - mn) / rng \* (bucket\_count - 1))

buckets[idx].append(x)

# Сортируем каждую корзину алгоритмом вставок (можно заменить на другой при желании)

for b in buckets:

insertion\_sort(b)

# Склеиваем корзины слева направо

result: List[T] = []

for b in buckets:

result.extend(b)

return result

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

# Демонстрация: разные наборы данных

tests = [

[0.42, 0.32, 0.23, 0.52, 0.25, 0.47, 0.51],

[3, 1, 4, 1, 5, 9, 2, 6],

[-3.5, 2.2, 0.0, -1.1, 4.8, 2.2, -3.6],

[10, 10, 10, 10],

]

for i, arr in enumerate(tests, 1):

print(f"Тест {i}: исходные данные = {arr}")

sorted\_arr = bucket\_sort(arr) # можно передать bucket\_count=... при необходимости

print(f"Тест {i}: отсортировано = {sorted\_arr}\n")

РЕЗУЛЬТАТ РАБОТЫ КОДА:

Тест 1: исходные данные = [0.42, 0.32, 0.23, 0.52, 0.25, 0.47, 0.51]

Тест 1: отсортировано = [0.23, 0.25, 0.32, 0.42, 0.47, 0.51, 0.52]

Тест 2: исходные данные = [3, 1, 4, 1, 5, 9, 2, 6]

Тест 2: отсортировано = [1, 1, 2, 3, 4, 5, 6, 9]

Тест 3: исходные данные = [-3.5, 2.2, 0.0, -1.1, 4.8, 2.2, -3.6]

Тест 3: отсортировано = [-3.6, -3.5, -1.1, 0.0, 2.2, 2.2, 4.8]

Тест 4: исходные данные = [10, 10, 10, 10]

Тест 4: отсортировано = [10, 10, 10, 10]

2)Блинная сортировка  
**Код**  
  
def flip(arr, i):  
    start = 0  
    while start < i:  
        arr[start], arr[i] = arr[i], arr[start]  
        start += 1  
        i -= 1  
  
def find\_max(arr, n):  
    max\_i = 0  
    for i in range(1, n):  
        if arr[i] > arr[max\_i]:  
            max\_i = i  
    return max\_i  
  
def pancake\_sort(arr):  
    n = len(arr)  
    for size in range(n, 1, -1):  
        max\_i = find\_max(arr, size)  
        if max\_i != size - 1:  
            flip(arr, max\_i)  
            flip(arr, size - 1)  
    return arr  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    a = [3, 6, 1, 10, 2, 7]  
    print("Исходный:", a)  
    print("Отсортированный:", pancake\_sort(a))

РЕЗУЛЬТАТ РАБОТЫ КОДА:

Исходный: [3, 6, 1, 10, 2, 7]

Отсортированный: [1, 2, 3, 6, 7, 10]

3)Сортировка бусинами

def bead\_sort(arr):  
    if any(x < 0 for x in arr):  
        raise ValueError("Bead sort works only with non-negative integers")  
    max\_val = max(arr)  
    beads = [[1]\*x + [0]\*(max\_val-x) for x in arr]  
    for j in range(max\_val):  
        col\_sum = sum(row[j] for row in beads)  
        for i in range(len(arr)):  
            beads[i][j] = 1 if i >= len(arr) - col\_sum else 0  
    result = [sum(row) for row in beads]  
    return result  
  
if name == "\_\_main\_\_":  
    a = [5, 3, 1, 7, 4]  
    print("Исходный:", a)  
    print("Отсортированный:", bead\_sort(a))

4)поиск скачками  
import math  
  
def jump\_search(arr, target):  
    n = len(arr)  
    step = int(math.sqrt(n))  
    prev = 0  
  
    while prev < n and arr[min(step, n)-1] < target:  
        prev = step  
        step += int(math.sqrt(n))  
        if prev >= n:  
            return -1  
  
    for i in range(prev, min(step, n)):  
        if arr[i] == target:  
            return i  
    return -1  
  
ifke\_sort(a)== "\_\_main\_\_":  
    a = [1, 3, 5, 7, 9, 11, 13, 15]  
    x = 9  
    print("Индекс элемента:", jump\_search(a, x))

5)экспонециальный поиск  
def binary\_search(arr, left, right, target):  
    while left <= right:  
        mid = (left + right) // 2  
        if arr[mid] == target:  
            return mid  
        elif arr[mid] < target:  
            left = mid + 1  
        else:  
            right = mid - 1  
    return -1  
  
def exponential\_search(arr, target):  
    if arr[0] == target:  
        return 0  
    i = 1  
    while i < len(arr) and arr[i] <= target:  
        i \*= 2  
    return binary\_search(arr, i//2, min(i, len(arr)-1), target)  
  
ifсходный:",== "\_\_main\_\_":  
    a = [2, 4, 6, 8, 10, 12, 14, 16]  
    x = 14  
    print("Индекс элемента:", exponential\_search(a, x))

6) тернарный поиск

def ternary\_search(arr, left, right, target):  
    while left <= right:  
        third = (right - left) // 3  
        mid1 = left + third  
        mid2 = right - third  
        if arr[mid1] == target:  
            return mid1  
        if arr[mid2] == target:  
            return mid2  
        if target < arr[mid1]:  
            right = mid1 - 1  
        elif target > arr[mid2]:  
            left = mid2 + 1  
        else:  
            left = mid1 + 1  
            right = mid2 - 1  
    return -1  
  
if 2, 7]  
   == "\_\_main\_\_":  
    a = [1, 3, 5, 7, 9, 11, 13]  
    x = 11  
    print("Индекс элемента:", ternary\_search(a, 0, len(a)-1, x))