**Алгоритмы сортировки на Python**

1. **Блочная (корзинная) сортировка**

def bucket\_sort(arr):  
 # Оператор 1: Создание пустых корзин  
 n = len(arr)  
 buckets = [[] for \_ in range(n)]  
  
 # Оператор 2: Распределение элементов по корзинам  
 for num in arr:  
 index = int(num \* n) # Для чисел в диапазоне [0,1)  
 buckets[index].append(num)  
  
 # Оператор 3: Сортировка каждой корзины  
 for bucket in buckets:  
 bucket.sort() # Используем встроенную сортировку  
  
 # Оператор 4: Объединение корзин  
 result = []  
 for bucket in buckets:  
 result.extend(bucket)  
  
 return result  
  
  
# Пример использования  
arr = [0.42, 0.32, 0.33, 0.52, 0.37, 0.47, 0.51]  
sorted\_arr = bucket\_sort(arr)  
print(f"Блочная сортировка: {sorted\_arr}")

**Блочная сортировка: [0.32, 0.33, 0.37, 0.42, 0.47, 0.51, 0.52]**

1. **Блинная Сортировка**

def flip(arr, i):  
 # Оператор 1: Переворот массива до позиции i  
 start = 0  
 while start < i:  
 arr[start], arr[i] = arr[i], arr[start]  
 start += 1  
 i -= 1  
  
  
def pancake\_sort(arr):  
 n = len(arr)  
  
 # Оператор 2: Поиск максимального элемента  
 for curr\_size in range(n, 1, -1):  
 # Находим индекс максимального элемента  
 max\_idx = arr.index(max(arr[:curr\_size]))  
  
 # Оператор 3: Переворот до максимального элемента  
 if max\_idx != curr\_size - 1:  
 flip(arr, max\_idx)  
 flip(arr, curr\_size - 1)  
  
 return arr  
  
  
# Пример использования  
arr = [23, 10, 20, 11, 12, 6, 7]  
sorted\_arr = pancake\_sort(arr.copy())  
print(f"Блинная сортировка: {sorted\_arr}")

**Блинная сортировка: [6, 7, 10, 11, 12, 20, 23]**

1. **Сортировка Бусинами**

def bead\_sort(arr):  
 if not arr:  
 return []  
  
 # Оператор 1: Находим максимальный элемент  
 max\_val = max(arr)  
  
 # Оператор 2: Создаем "абак" с бусинами  
 beads = [[0] \* max\_val for \_ in range(len(arr))]  
  
 # Оператор 3: Размещаем бусины  
 for i in range(len(arr)):  
 for j in range(arr[i]):  
 beads[i][j] = 1  
  
 # Оператор 4: Симуляция падения бусин  
 for j in range(max\_val):  
 sum\_beads = 0  
 for i in range(len(arr)):  
 sum\_beads += beads[i][j]  
 beads[i][j] = 0  
  
 # Оператор 5: Бусины падают вниз  
 for i in range(len(arr) - sum\_beads, len(arr)):  
 beads[i][j] = 1  
  
 # Оператор 6: Преобразуем обратно в числа  
 result = [sum(row) for row in beads]  
 return result  
  
  
# Пример использования  
arr = [3, 1, 4, 1, 5]  
sorted\_arr = bead\_sort(arr)  
print(f"Сортировка бусинами: {sorted\_arr}")

**Сортировка бусинами: [1, 1, 3, 4, 5]**

1. **Поиск Скачками**

import math  
  
  
def jump\_search(arr, x):  
 n = len(arr)  
  
 # Оператор 1: Определение размера прыжка  
 step = int(math.sqrt(n))  
 prev = 0  
  
 # Оператор 2: Поиск блока, содержащего элемент  
 while arr[min(step, n) - 1] < x:  
 prev = step  
 step += int(math.sqrt(n))  
 if prev >= n:  
 return -1  
  
 # Оператор 3: Линейный поиск в найденном блоке  
 while arr[prev] < x:  
 prev += 1  
 if prev == min(step, n):  
 return -1  
  
 # Оператор 4: Проверка найденного элемента  
 if arr[prev] == x:  
 return prev  
  
 return -1  
  
  
# Пример использования  
arr = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]  
target = 55  
result = jump\_search(arr, target)  
print(f"Поиск скачками: элемент {target} найден на позиции {result}")

**Поиск скачками: элемент 55 найден на позиции 10**

1. **Экспоненциальный Поиск**

def binary\_search(arr, left, right, x):  
 # Вспомогательная функция бинарного поиска  
 while left <= right:  
 mid = left + (right - left) // 2  
 if arr[mid] == x:  
 return mid  
 elif arr[mid] < x:  
 left = mid + 1  
 else:  
 right = mid - 1  
 return -1  
  
  
def exponential\_search(arr, x):  
 n = len(arr)  
  
 # Оператор 1: Поиск диапазона  
 if arr[0] == x:  
 return 0  
  
 i = 1  
 while i < n and arr[i] <= x:  
 i \*= 2 # Экспоненциальное увеличение границы  
  
 # Оператор 2: Бинарный поиск в найденном диапазоне  
 return binary\_search(arr, i // 2, min(i, n - 1), x)  
  
  
# Пример использования  
arr = [2, 3, 4, 10, 40, 45, 50, 60, 70, 80, 90, 100]  
target = 45  
result = exponential\_search(arr, target)  
print(f"Экспоненциальный поиск: элемент {target} найден на позиции {result}")

**Экспоненциальный поиск: элемент 45 найден на позиции 5**

1. **Тернарный Поиск**

def ternary\_search(arr, left, right, x):  
 if right >= left:  
 # Оператор 1: Вычисление двух точек деления  
 mid1 = left + (right - left) // 3  
 mid2 = right - (right - left) // 3  
  
 # Оператор 2: Проверка точек деления  
 if arr[mid1] == x:  
 return mid1  
 if arr[mid2] == x:  
 return mid2  
  
 # Оператор 3: Рекурсивный поиск в одной из трех частей  
 if x < arr[mid1]:  
 return ternary\_search(arr, left, mid1 - 1, x)  
 elif x > arr[mid2]:  
 return ternary\_search(arr, mid2 + 1, right, x)  
 else:  
 return ternary\_search(arr, mid1 + 1, mid2 - 1, x)  
  
 return -1  
  
  
# Пример использования  
arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
target = 6  
result = ternary\_search(arr, 0, len(arr) - 1, target)  
print(f"Тернарный поиск: элемент {target} найден на позиции {result}")

**Тернарный поиск: элемент 6 найден на позиции 5**