

Bucket_sort

The screenshot shows an Online Python 3 IDE with a dark theme. The main editor displays a Python script for the Bucket Sort algorithm. The script includes comments in Russian explaining the steps: finding min/max values, creating buckets, distributing elements, sorting each bucket, and merging them. The input array is [0.42, 0.32, 0.33, 0.52, 0.37, 0.47, 0.51]. The output shows the sorted array: [0.32, 0.33, 0.37, 0.42, 0.47, 0.51, 0.52]. The right sidebar contains a 'JDoodle.ai' logo, language version settings (3.11.5), and interactive mode options.

```
1 def bucket_sort(arr):
2     """
3     Блочная сортировка - распределяет элементы по "корзинам",
4     затем сортирует каждую корзину отдельно
5     """
6     if len(arr) == 0:
7         return arr
8
9     # Шаг 1: Находим минимальное и максимальное значение
10    min_val = min(arr)
11    max_val = max(arr)
12
13    # Шаг 2: Создаем пустые корзины
14    bucket_count = len(arr) # Количество корзин = количеству элементов
15    buckets = [[] for _ in range(bucket_count)]
16
17    # Шаг 3: Распределяем элементы по корзинам
18    for num in arr:
19        # Вычисляем индекс корзины для текущего числа
20        index = int((num - min_val) / (max_val - min_val) * (bucket_count - 1))
21        buckets[index].append(num)
22
23    # Шаг 4: Сортируем каждую корзину (используем встроенную сортировку)
24    for bucket in buckets:
25        bucket.sort()
26
27    # Шаг 5: Объединяем отсортированные корзины
28    sorted_arr = []
29    for bucket in buckets:
30        sorted_arr.extend(bucket)
31
32    return sorted_arr
33
34    # Пример использования
35    arr = [0.42, 0.32, 0.33, 0.52, 0.37, 0.47, 0.51]
36    print("Исходный массив:", arr)
37    print("Отсортированный массив:", bucket_sort(arr))
```

Bead_sort

The screenshot shows an Online Python 3 IDE with a dark theme. The main editor displays a Python script for the Bead Sort algorithm. The script includes comments in Russian explaining the steps: finding the maximum value, creating a grid of beads, counting beads in each column, and then sorting by height. The input array is [3, 1, 4, 1, 5, 0, 2, 6]. The output shows the sorted array: [1, 1, 2, 3, 4, 5, 6, 9]. The right sidebar contains a 'JDoodle.ai' logo, language version settings (3.11.5), and interactive mode options.

```
1 def bead_sort(arr):
2     """
3     Сортировка бусинами - визуализирует сортировку как падающие бусины
4     Работает только с неотрицательными целыми числами
5     """
6     if len(arr) == 0:
7         return arr
8
9     # Шаг 1: Находим максимальное значение
10    max_val = max(arr)
11
12    # Шаг 2: Создаем "стержни" с бусинами
13    # в каждой строке представляем один элемент массива
14    beads = [[0] * num + [1] * (max_val - num) for num in arr]
15
16    # Шаг 3: "Заставляем бусины падать" - транспонируем матрицу
17    for i in range(max_val):
18        for j in range(len(arr)):
19            # Подсчитываем количество бусин в каждой строке
20            beads[j][i] = 1 if i < arr[j] else 0
21
22    # Шаг 4: Подсчитываем бусины в каждом стержне после "падения"
23    result = []
24    for i in range(max_val):
25        count = 0
26        for j in range(len(arr)):
27            count += beads[j][i]
28        result.append(count)
29
30    # Шаг 5: Преобразуем обратно в отсортированный массив
31    # более эффективная реализация "падения бусин"
32    for i in range(len(arr)):
33        for j in range(max_val - 1, -1, -1):
34            if beads[i][j] == 1:
35                arr[i] = j + 1
36                break
37
38    return sorted(arr) # Возвращаем отсортированную копию
39
40    # Альтернативная, более эффективная реализация
41    def bead_sort_optimized(arr):
42        """Оптимизированная версия сортировки бусинами"""
43        if not arr:
44            return []
45
46        max_val = max(arr)
47
48        # Создаем матрицу бусин
49        beads = [[0] * max_val for _ in range(len(arr))]
50
51        # Расставляем бусины
52        for i, num in enumerate(arr):
```

Jump_search

```
main.py
1 import math
2
3 def jump_search(arr, target):
4     """
5     Поиск скачками - прыгает с фиксированным шагом, затем делает линейный поиск
6     Требуется отсортированный массив
7     """
8     n = len(arr)
9
10    # Шаг 1: Определяем размер прыжка (квадратный корень из длины)
11    step = int(math.sqrt(n))
12
13    # Шаг 2: Находим блок, где может находиться элемент
14    prev = 0
15    while arr[min(step, n) - 1] < target:
16        prev = step
17        step += int(math.sqrt(n))
18        if prev >= n:
19            return -1 # Элемент не найден
20
21    # Шаг 3: Линейный поиск в найденном блоке
22    while arr[prev] < target:
23        prev += 1
24        if prev == min(step, n):
25            return -1 # Элемент не найден
26
27    # Шаг 4: Проверка, найден ли элемент
28    if arr[prev] == target:
29        return prev
30    else:
31        return -1
32
33    # Пример использования
34    arr = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
35    target = 55
36    print(f"Массив: {arr}")
37    print(f"Поиск элемента {target}")
38    result = jump_search(arr, target)
39    if result != -1:
40        print(f"Элемент найден на позиции {result}")
41    else:
42        print("Элемент не найден")
```

Build beautiful web apps with JDoodle.ai

Language version: 3.11.5 Interactive Mode

Input arguments

Output Generated files

Массив: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
Поиск элемента 55
Элемент найден на позиции 10

Ternary_search

```
main.py
1 def ternary_search(arr, target):
2     """
3     Тернарный поиск - делит массив на три части
4     Рекурсивная реализация
5     """
6     def ternary_recursive(left, right):
7         if left > right:
8             return -1
9
10        # Шаг 1: Делим массив на три части
11        mid1 = left + (right - left) // 3
12        mid2 = right - (right - left) // 3
13
14        # Шаг 2: Проверка точки деления
15        if arr[mid1] == target:
16            return mid1
17        if arr[mid2] == target:
18            return mid2
19
20        # Шаг 3: Определяем, в какой части продолжать поиск
21        if target < arr[mid1]:
22            return ternary_recursive(left, mid1 - 1)
23        elif target > arr[mid2]:
24            return ternary_recursive(mid2 + 1, right)
25        else:
26            return ternary_recursive(mid1 + 1, mid2 - 1)
27
28    return ternary_recursive(0, len(arr) - 1)
29
30 def ternary_search_iterative(arr, target):
31     """
32     Тернарный поиск - итеративная реализация
33     """
34    left, right = 0, len(arr) - 1
35
36    while left <= right:
37        # Шаг 1: Делим массив на три части
38        mid1 = left + (right - left) // 3
39        mid2 = right - (right - left) // 3
40
41        # Шаг 2: Проверка точки деления
42        if arr[mid1] == target:
43            return mid1
44        if arr[mid2] == target:
45            return mid2
46
47        # Шаг 3: Определяем, в какой части продолжать поиск
48        if target < arr[mid1]:
49            right = mid1 - 1
50        elif target > arr[mid2]:
51            left = mid2 + 1
52        else:
```

Build beautiful web apps with JDoodle.ai

Language version: 3.11.5 Interactive Mode

Input arguments

Output Generated files

Массив: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Поиск элемента 7
Рекурсивный поиск: элемент на позиции 6
Итеративный поиск: элемент на позиции 6

Exponential_search

```
Online Python 3 IDE
main.py
108-         else:
109-             right = mid - 1
110-             return -1
111-
112-     n = len(arr)
113-
114-     # Шаг 1: Проверка пустого массива и первого элемента
115-     if n == 0:
116-         return -1
117-
118-     if arr[0] == target:
119-         return 0
120-
121-     # Шаг 2: Экспоненциальное увеличение границы
122-     i = 1
123-     while i < n and arr[i] <= target:
124-         i *= 2 # Увеличиваем в 2 раза на каждом шаге
125-
126-     # Шаг 3: Бинарный поиск в найденном диапазоне
127-     left = i // 2 # Начало диапазона
128-     right = min(i, n - 1) # Конец диапазона (не превышает длину массива)
129-
130-     return binary_search(arr, left, right, target)
131-
132-
133- def compare_search_algorithms():
134-     """Сравнение различных алгоритмов поиска"""
135-     import time
136-
137-     # Создаем тестовый массив
138-     arr = list(range(1, 1000001)) # 1 до 1,000,000
139-     target = 54321
140-
141-     algorithms = {
142-         "Jump Search": jump_search,
143-         "Exponential Search": exponential_search,
144-         "Ternary Search": ternary_search_iterative
145-     }
146-
147-     print("Сравнение алгоритмов поиска:")
148-     print(f"Размер массива: {len(arr)}")
149-     print(f"Целевой элемент: {target}")
150-     print()
151-
152-     for name, algorithm in algorithms.items():
153-         start_time = time.time()
154-         result = algorithm(arr, target)
155-         end_time = time.time()
156-
157-         print(f"{name}:")
158-         print(f"  Позиция: {result}")
159-         print(f"  Время: {(end_time - start_time)*1000:.4f} мс")
160-         print()
161-
162-     # Запуск сравнения
163-     compare_search_algorithms()
```

Build beautiful web apps with JDoodle.ai

Language version: 3.11.5 Interactive Mode

Input arguments

Output Generated files

Сравнение алгоритмов поиска:
Размер массива: 1,000,000
Целевой элемент: 54321

Jump Search:
Позиция: 54320
Время: 0.2499 мс

Exponential Search:
Позиция: 54320
Время: 0.0246 мс

Ternary Search:
Позиция: 54320
Время: 0.0191 мс

Сравнение

```
main.py
1 def exponential_search(arr, target):
2     """
3     Экспоненциальный поиск - состоит из двух этапов:
4     1. Нахождение диапазона экспоненциальным увеличением границ
5     2. Бинарный поиск в найденном диапазоне
6     Особенно эффективен для неограниченных массивов
7     """
8
9     def binary_search(arr, left, right, target):
10         """Вспомогательная функция бинарного поиска"""
11         while left <= right:
12             mid = left + (right - left) // 2
13
14             if arr[mid] == target:
15                 return mid
16             elif arr[mid] < target:
17                 left = mid + 1
18             else:
19                 right = mid - 1
20         return -1
21
22     n = len(arr)
23
24     # Шаг 1: Проверка пустого массива и первого элемента
25     if n == 0:
26         return -1
27
28     if arr[0] == target:
29         return 0
30
31     # Шаг 2: Экспоненциальное увеличение границы
32     i = 1
33     while i < n and arr[i] <= target:
34         i *= 2 # Увеличиваем в 2 раза на каждом шаге
35
36     # Шаг 3: Бинарный поиск в найденном диапазоне
37     left = i // 2 # Начало диапазона
38     right = min(i, n - 1) # Конец диапазона (не превышает длину массива)
39
40     return binary_search(arr, left, right, target)
41
42 # Примеры использования
43
44 print("\n" + "="*50)
45 print("=== Пример 1: Эффективность на большом массиве ===")
46 import time
47
48 # Создаем большой отсортированный массив
49 large_arr = list(range(1, 1000001)) # 1 до 1,000,000
50 target3 = 543210
51
52 start_time = time.time()
53 result3 = exponential_search(large_arr, target3)
54 end_time = time.time()
55
56 """
57 Пример 1: Эффективность на большом массиве
58 Поиск элемента 543210 в массиве из 1,000,000 элементов
59 Результат: 543209
60 Время выполнения: 0.0610 мс
61 """
```

Build beautiful web apps with JDoodle.ai

Language version: 3.11.5 Interactive Mode

Input arguments

Output Generated files

=====

=== Пример 1: Эффективность на большом массиве ===

Поиск элемента 543210 в массиве из 1,000,000 элементов

Результат: 543209

Время выполнения: 0.0610 мс