**Пример на Python**

1. **сортировка выбором**

def selection\_sort(arr):  
 n = len(arr)  
  
 for i in range(n):  
 # Предположим, что первый элемент несортированной части является минимальным  
 min\_idx = i  
  
 # Поиск минимального элемента среди оставшихся элементов  
 for j in range(i + 1, n):  
 if arr[j] < arr[min\_idx]:  
 min\_idx = j  
  
 # Меняем местами минимальный элемент с началом несортированного участка  
 arr[i], arr[min\_idx] = arr[min\_idx], arr[i]  
  
  
# Пример использования  
arr = [64, 25, 12, 22, 11]  
selection\_sort(arr)

**Отсортированный массив: [11, 12, 22, 25, 64]**

1. **сортировка обменом (пузырек)**
2. def bubble\_sort(arr):  
    n = len(arr)  
     
    # Внешний цикл проходит по каждому элементу  
    for i in range(n):  
    swapped = False  
     
    # Внутренний цикл сравнивает соседние элементы  
    for j in range(0, n - i - 1):  
    if arr[j] > arr[j + 1]:  
    # Обмен местоположением  
    arr[j], arr[j + 1] = arr[j + 1], arr[j]  
    swapped = True  
     
    # Если ни разу не произошло перестановок, значит массив уже отсортирован  
    if not swapped:  
    break  
     
     
   # Пример использования  
   arr = [64, 34, 25, 12, 22, 11, 90]  
   bubble\_sort(arr)  
   print("Отсортированный массив:", arr)

**Отсортированный массив: [11, 12, 22, 25, 34, 64, 90]**

1. **Алгоритм сортировки вставками**
2. def insertion\_sort(arr):  
    n = len(arr)  
     
    # Начинаем с второго элемента  
    for i in range(1, n):  
    key = arr[i]  
    j = i - 1  
     
    # Перемещаем большие элементы вправо, освобождая место для нового ключа  
    while j >= 0 and key < arr[j]:  
    arr[j + 1] = arr[j]  
    j -= 1  
     
    # Вставляем ключ в нужное место  
    arr[j + 1] = key  
     
     
   # Пример использования  
   arr = [12, 11, 13, 5, 6]  
   insertion\_sort(arr)  
   print("Отсортированный массив:", arr)

**Отсортированный массив: [5, 6, 11, 12, 13]**

**4.** **сортировка слиянием**

def merge(left, right):  
 result = []  
 i = j = 0  
  
 # Пока оба списка содержат элементы  
 while i < len(left) and j < len(right):  
 if left[i] <= right[j]:  
 result.append(left[i])  
 i += 1  
 else:  
 result.append(right[j])  
 j += 1  
  
 # Добавляем оставшиеся элементы  
 result.extend(left[i:])  
 result.extend(right[j:])  
 return result  
  
  
def merge\_sort(arr):  
 if len(arr) <= 1:  
 return arr  
  
 mid = len(arr) // 2  
 left\_half = merge\_sort(arr[:mid])  
 right\_half = merge\_sort(arr[mid:])  
  
 return merge(left\_half, right\_half)  
  
  
# Пример использования  
arr = [38, 27, 43, 3, 9, 82, 10]  
sorted\_arr = merge\_sort(arr)  
print("Отсортированный массив:", sorted\_arr)

**Отсортированный массив: [3, 9, 10, 27, 38, 43, 82]**

1. **Алгоритм сортировки Шелла**

def shell\_sort(arr):  
 gap = len(arr) // 2 # Начальный интервал (шаг)  
  
 while gap > 0:  
 for i in range(gap, len(arr)):  
 temp = arr[i]  
 j = i  
  
 # Инсерционная сортировка с заданным шагом  
 while j >= gap and arr[j - gap] > temp:  
 arr[j] = arr[j - gap]  
 j -= gap  
  
 arr[j] = temp  
  
 gap //= 2 # Уменьшение шага вдвое  
  
  
# Пример использования  
arr = [9, 8, 3, 7, 5, 6, 4, 1]  
shell\_sort(arr)  
print("Отсортированный массив:", arr)

**Отсортированный массив: [1, 3, 4, 5, 6, 7, 8, 9]**

1. **Быстрая сортировка (Quick Sort)**
2. def quick\_sort(arr):  
    if len(arr) <= 1:  
    return arr  
     
    pivot = arr[len(arr) // 2] # Опорный элемент выбирается посередине  
    left = [x for x in arr if x < pivot]  
    middle = [x for x in arr if x == pivot]  
    right = [x for x in arr if x > pivot]  
     
    return quick\_sort(left) + middle + quick\_sort(right)  
     
     
   # Пример использования  
   arr = [10, 7, 8, 9, 1, 5]  
   sorted\_arr = quick\_sort(arr)  
   print("Отсортированный массив:", sorted\_arr)

**Отсортированный массив: [1, 5, 7, 8, 9, 10]**

## **7 . Пирамидальная сортировка (Heapsort)**

def heapify(arr, n, i):  
 largest = i # Текущий корень  
 left = 2 \* i + 1 # левый потомок  
 right = 2 \* i + 2 # правый потомок  
  
 # Проверка наличия большего левого сына  
 if left < n and arr[left] > arr[largest]:  
 largest = left  
  
 # Проверка наличия большего правого сына  
 if right < n and arr[right] > arr[largest]:  
 largest = right  
  
 # Замена корня и дальнейшие операции  
 if largest != i:  
 arr[i], arr[largest] = arr[largest], arr[i]  
 heapify(arr, n, largest)  
  
  
def heapsort(arr):  
 n = len(arr)  
  
 # Создание max-кучи  
 for i in range(n // 2 - 1, -1, -1):  
 heapify(arr, n, i)  
  
 # Извлечение элементов из кучи  
 for i in range(n - 1, 0, -1):  
 arr[i], arr[0] = arr[0], arr[i] # Перемещение наибольшего элемента в конец  
 heapify(arr, i, 0) # Восстанавливаем кучу без последнего элемента  
  
  
# Пример использования  
arr = [12, 11, 13, 5, 6, 7]  
heapsort(arr)  
print("Отсортированный массив:", arr)

**Отсортированный массив: [5, 6, 7, 11, 12, 13]**

1. **последовательный поиск**
2. def sequential\_search(arr, target):  
    for index, value in enumerate(arr):  
    if value == target:  
    return index  
    return -1 # Элемент не найден  
     
   # Пример использования  
   arr = [10, 20, 30, 40, 50]  
   target = 30  
   result = sequential\_search(arr, target)  
   if result != -1:  
    print(f"Элемент {target} найден на позиции {result}.")  
   else:  
    print(f"Элемент {target} не найден.")

**Элемент 30 найден на позиции 2.**

**9. бинарный поиск**

**9** def binary\_search(arr, target):  
 low, high = 0, len(arr) - 1  
  
 while low <= high:  
 mid = (low + high) // 2  
  
 if arr[mid] == target:  
 return mid  
 elif arr[mid] < target:  
 low = mid + 1  
 else:  
 high = mid - 1  
  
 return -1 # Элемент не найден  
  
  
# Пример использования  
arr = [10, 20, 30, 40, 50]  
target = 30  
result = binary\_search(arr, target)  
if result != -1:  
 print(f"Элемент {target} найден на позиции {result}.")  
else:  
 print(f"Элемент {target} не найден.")

**Элемент 30 найден на позиции 2.**

## **Интерполирующий поиск**

1. def interpolation\_search(arr, target):  
    low, high = 0, len(arr) - 1  
     
    while low <= high and target >= arr[low] and target <= arr[high]:  
    pos = low + ((target - arr[low]) \* (high - low)) // (arr[high] - arr[low])  
     
    if arr[pos] == target:  
    return pos  
    elif arr[pos] < target:  
    low = pos + 1  
    else:  
    high = pos - 1  
     
    return -1 # Элемент не найден  
     
     
   # Пример использования  
   arr = [10, 20, 30, 40, 50, 60, 70]  
   target = 30  
   result = interpolation\_search(arr, target)  
   if result != -1:  
    print(f"Элемент {target} найден на позиции {result}.")  
   else:  
    print(f"Элемент {target} не найден.")

**Элемент 30 найден на позиции 2.**

## **Фибоначчи поиск**

import math  
  
  
# Генерация ряда Фибоначчи до указанного предела  
def fib\_gen(limit):  
 fib = [0, 1]  
 while fib[-1] < limit:  
 fib.append(fib[-1] + fib[-2])  
 return fib  
  
  
def fibonacci\_search(arr, target):  
 n = len(arr)  
 fib\_nums = fib\_gen(n)  
 fib\_m = len(fib\_nums) - 1 # Наибольшее подходящее число Фибоначчи  
  
 offset = -1  
 while fib\_nums[fib\_m] > 1:  
 i = min(offset + fib\_nums[fib\_m - 2], n - 1)  
  
 if arr[i] < target:  
 fib\_m -= 1  
 offset = i  
 elif arr[i] > target:  
 fib\_m -= 2  
 else:  
 return i  
  
 if fib\_nums[fib\_m - 1] and arr[offset + 1] == target:  
 return offset + 1  
  
 return -1 # Элемент не найден  
  
  
# Пример использования  
arr = [10, 20, 30, 40, 50, 60, 70]  
target = 30  
result = fibonacci\_search(arr, target)  
if result != -1:  
 print(f"Элемент {target} найден на позиции {result}.")  
else:  
 print(f"Элемент {target} не найден.")

**Элемент 30 найден на позиции 2.**