Sorting Algorithms and Time Complexities

Algorithm	Best Case	Average Case	Worst Case
Bubble Sort	O(n)	O(n²)	O(n²)
Insertion Sort	O(n)	O(n²)	O(n²)
Selection Sort	O(n²)	O(n²)	O(n²)
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Quick Sort	O(n log n)	O(n log n)	O(n²)

Python Code for Sorting Algorithms

```python
# Bubble Sort
def bubble_sort(arr):
    """Sorts an array using Bubble Sort algorithm."""
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr  # Time Complexity: O(n^2) worst, O(n) best (already sorted)

# Insertion Sort
def insertion_sort(arr):
    """Sorts an array using Insertion Sort algorithm."""
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr  # Time Complexity: O(n^2) worst, O(n) best (already sorted)

# Selection Sort
def selection_sort(arr):
    """Sorts an array using Selection Sort algorithm."""
    n = len(arr)
    for i in range(n - 1):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr  # Time Complexity: O(n^2) in all cases

# Merge Sort
def merge_sort(arr):
    """Sorts an array using Merge Sort algorithm."""
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = merge_sort(arr[:mid])
        right_half = merge_sort(arr[mid:])

        return merge(left_half, right_half)
    return arr  # Time Complexity: O(n log n) in all cases

def merge(left, right):
    """Merges two sorted halves."""
    sorted_arr = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_arr.append(left[i])
            i += 1
        else:
            sorted_arr.append(right[j])
            j += 1
    sorted_arr.extend(left[i:])
    sorted_arr.extend(right[j:])
    return sorted_arr

# Quick Sort
def quick_sort(arr):
    """Sorts an array using Quick Sort algorithm."""
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    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)
    # Time Complexity: O(n log n) average, O(n^2) worst (bad pivot selection)
```