◇ 1. Insertion Sort

Algorithm Steps:

1. Start from index 1 (second element) of the array.
2. Compare the current element with the elements before it.
3. Shift all elements that are greater than the current element to the right.
4. Insert the current element into the correct position in the sorted subarray.
5. Repeat steps 2–4 for all elements from index 1 to n−1.

Pseudocode (Python Style):

```
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
```

◇ 2. Selection Sort

Algorithm Steps:

1. Start from the first index of the array.
2. Find the minimum element in the unsorted part of the array.
3. Swap the minimum element with the first element of the unsorted part.
4. Move the boundary of the sorted part one element forward.
5. Repeat steps 2–4 until the array is fully sorted.

Pseudocode:

```
for i in range(len(arr)):
    min_idx = i
    for j in range(i + 1, len(arr)):
        if arr[j] < arr[min_idx]:
            min_idx = j
    arr[i], arr[min_idx] = arr[min_idx], arr[i]
```

◇ 3. Bubble Sort

Algorithm Steps:

1. Start from the beginning of the array.
2. Compare each pair of adjacent elements.
3. Swap the elements if they are in the wrong order.
4. Continue this process for the entire array (one pass).
5. After each pass, the largest unsorted element will have “bubbled” to its correct position.
6. Repeat steps 1–5 until no swaps are needed.

Pseudocode:

```
n = len(arr)
for i in range(n):
    for j in range(0, n - i - 1):
        if arr[j] > arr[j + 1]:
            arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

◇ 4. Merge Sort

Algorithm Steps:

1. If the array has more than one element:

   a. Divide the array into two halves.

   b. Recursively apply merge sort on each half.

   c. Merge the two sorted halves into one sorted array.

2. Use a helper function to merge the two halves:

   a. Compare the smallest elements of each half.

   b. Insert the smaller one into the output array.

   c. Repeat until all elements are merged.

Pseudocode:

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
```

◇ 5. Quick Sort

Algorithm Steps:

1. Choose a pivot element.
2. Partition the array into two parts:

   a. Elements less than the pivot go to the left.

   b. Elements greater than the pivot go to the right.
3. Recursively apply quick sort on the left and right subarrays.
4. Combine the sorted subarrays with the pivot in between.

Pseudocode:

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[-1]
    left = [x for x in arr[:-1] if x <= pivot]
    right = [x for x in arr[:-1] if x > pivot]
    return quick_sort(left) + [pivot] + quick_sort(right)
```

◇ 6. Counting Sort

Algorithm Steps:

1. Find the maximum and minimum elements in the array.
2. Create a count array of size (max – min + 1), initialized to 0.
3. Count the frequency of each element and store it in the count array.
4. Modify the count array by adding previous counts to get cumulative counts.
5. Use the count array to place elements in the correct position in the output array.
6. Copy the sorted output array back into the original array.

Pseudocode:

```
def counting_sort(arr):
    max_val = max(arr)
    min_val = min(arr)
    range_of_elements = max_val - min_val + 1
    count = [0] * range_of_elements
    output = [0] * len(arr)

    for num in arr:
        count[num - min_val] += 1

    for i in range(1, len(count)):
        count[i] += count[i - 1]
    for num in reversed(arr):
        output[count[num - min_val] - 1] = num
        count[num - min_val] -= 1

    for i in range(len(arr)):
        arr[i] = output[i]
```

◇ 7. Radix Sort

Algorithm Steps:

1. Determine the maximum number of digits in the input.
2. Starting from the least significant digit:
   a. Use a stable sort (e.g., counting sort) to sort based on that digit.
3. Repeat step 2 for each digit.
4. After processing all digits, the array is sorted.

Pseudocode:

```
def counting_sort_digit(arr, exp):
    n = len(arr)
    output = [0] * n
    count = [0] * 10

    for i in range(n):
        index = (arr[i] // exp) % 10
        count[index] += 1

    for i in range(1, 10):
        count[i] += count[i - 1]

    for i in reversed(range(n)):
        index = (arr[i] // exp) % 10
        output[count[index] - 1] = arr[i]
        count[index] -= 1

    for i in range(n):
        arr[i] = output[i]

def radix_sort(arr):
    max_num = max(arr)
    exp = 1
    while max_num // exp > 0:
        counting_sort_digit(arr, exp)
        exp *= 10
```

◇ 8. Heap Sort

Algorithm Steps:

1. Build a max heap from the input array.
2. The largest element (root) is swapped with the last element.
3. Reduce the heap size by one and heapify the root.
4. Repeat steps 2–3 until the heap size is 1.

Pseudocode:

```
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[l] > arr[largest]:
        largest = l
    if r < n and arr[r] > arr[largest]:
        largest = r
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
```

◇ 9. Bucket Sort

Algorithm Steps:

1. Create several empty buckets.
2. Distribute the input elements into buckets.
3. Sort each bucket individually.
4. Concatenate all buckets in order.

Pseudocode:

```
def bucket_sort(arr):
    n = len(arr)
    buckets = [[] for _ in range(n)]

    for num in arr:
        index = int(n * num)
        buckets[index].append(num)

    for bucket in buckets:
        bucket.sort()

    k = 0
    for bucket in buckets:
        for num in bucket:
            arr[k] = num
            k += 1
```