1. Quick Sort

```
function quickSort(arr, l, r)
    if l < r
        pivotIndex = partition(arr, l, r)
        quickSort(arr, l, pivotIndex - 1)
        quickSort(arr, pivotIndex + 1, r)

function partition(arr, l, r)
    pivot = arr[r]
    i = l - 1

    for j = l to r - 1
        if arr[j] < pivot
            i = i + 1
            swap arr[i] and arr[j]

    swap arr[i + 1] and arr[r]
    return i + 1
```

2. Radix Sort

```
procedure RadixSort(arr)
    // Find the maximum number in the array to determine the number of digits
    max = findMax(arr)
    // Perform counting sort for every digit, from the least significant to the most significant
    for exp = 1 to max
        CountSort(arr, exp)
procedure CountSort(arr, exp)
    n = length(arr)
    output = new Array[n]
    count = new Array[10]  // 0 to 9, as we're working with base 10
    // Initialize count array with zeros
    for i = 0 to 9
        count[i] = 0
    // Count the occurrences of each digit at the current place value
    for i = 0 to n - 1
        index = (arr[i] / exp) % 10
        count[index] = count[index] + 1
    // Update count array to store the positions of digits in the output
    for i = 1 to 9
        count[i] = count[i] + count[i - 1]
    // Build the output array by placing elements in their correct positions
    for i = n - 1 to 0
        index = (arr[i] / exp) % 10
        output[count[index] - 1] = arr[i]
        count[index] = count[index] - 1
```

```
 // Copy the sorted elements from the output array to the original array
    for i = 0 to n - 1
        arr[i] = output[i]

function findMax(arr)
    max = arr[0]
    for i = 1 to length(arr) - 1
        if arr[i] > max
            max = arr[i]
    return max
```

3. Merge Sort

```
procedure MergeSort(arr)
    if length(arr) <= 1
        return arr  // Already sorted
    // Split the array into two halves
    mid = length(arr) / 2
    left = arr[0 to mid - 1]
    right = arr[mid to end]
    // Recursively sort both halves
    left = MergeSort(left)
    right = MergeSort(right)
    // Merge the sorted halves
    result = Merge(left, right)
    return result
procedure Merge(left, right)
    result = new Array
    i = 0  // Index for left array
    j = 0  // Index for right array
    // Compare elements from both arrays and merge them in sorted order
    while i < length(left) and j < length(right)
        if left[i] <= right[j]
            result.append(left[i])
            i = i + 1
        else
            result.append(right[j])
            j = j + 1
```

```
// Append any remaining elements from both arrays (if any)
while i < length(left)
    result.append(left[i])
    i = i + 1
while j < length(right)
    result.append(right[j])
    j = j + 1
return result
```

4. Binary search