1. **Java Implementation of Radix Sort Algorithm with Big-Oh Analysis**

**Radix Sort Class**

public class RadixSort {

/\*\*

\* Sorts an array of integers using the Radix Sort algorithm

\* Radix Sort processes each digit of the numbers, starting from the least significant digit (LSD)

\* to the most significant digit (MSD). It uses Counting Sort as a subroutine to sort the numbers

\* based on each digit.

\*

\* **@param** arr The array of integers to be sorted.

\*/

public static void radixSort(int[] arr) {

// Find the maximum number to know the number of digits

int max = getMax(arr); // Get the maximum value in the array

// Do counting sort for every digit.

// exp is 1, 10, 100, representing the digit's place value

**for** (int exp = 1; max / exp > 0; exp \*= 10) { // Loop through each digit place

countingSort(arr, exp); // Perform counting sort based on the current digit

}

}

/\*\*

\* Finds and returns the maximum value in an array.

\* This is necessary to determine the number of digits in the largest number.

\*

\* **@param** arr The array of integers.

\* **@return** the maximum integer in the array

\*/

private static int getMax(int[] arr) {

int max = arr[0]; // Initialize max with the first element

**for** (int i=1; i<arr.length; i++) {// Iterate over the array starting from the second element

**if** (arr[i] > max) {// If the current element is greater than max

max = arr[i]; // Update max with the current element

}

}

**return** max; // Return the maximum value found

}

/\*\*

\* Performs Counting Sort on the array based on the digit represented by exp.

\* Counting Sort is a stable sort that sorts elements based on a single digit.

\*

\* **@param** arr The array to be sorted.

\* **@param** exp the exponent representing the digit position (1 for units, 10 for tens, 100 for hundredths)

\*/

private static void countingSort(int[] arr, int exp) {

int n = arr.length; // Get the number of elements in the array

int output[] = **new** int[n]; // Output array to store the sorted elements

int count[] = **new** int[10]; // Count array to store count of occurences for digits 0-9

// Store count of occurrences of each digiti in count[]

**for** (int i = 0; i < n; i++) {

int digit = (arr[i] / exp) % 10; // Extract the digit at the current exponenet (digit position)

count[digit]++; // Increment the count for this digit

}

// Change count[i] so that it contains the actual position of this digit in output[]

**for** (int i = 1; i < 10; i++) {

count[i] += count[i - 1]; // Update count[i] by adding the count of the previous digit

}

// Build the output array by placing elements in their correct positions

**for** (int i = n - 1; i >= 0; i--) { // Iterate from the end to maintain stability

int digit = (arr[i] / exp) % 10; // Extract the digit at the current exponent

output[count[digit] - 1] = arr[i]; // Place the element at the correct position in output[]

count[digit]--; // Decrement the count for this digit

}

// Copy the output array to arr[]

System.arraycopy(output, 0, arr, 0, n);

}

}

**RadixSortTest Class**

public class RadixSortTest {

/\*\*

\* The main method to test the Radix Sort method

\* It initializes an array, prints it, sorts it, and then prints the sorted array.

\*

\* **@param** args Command-line arguments.

\*/

public static void main(String[] args) {

// Initialize an array of integers to be sorted

int[] arr = {783, 99, 472, 182, 264, 543, 356, 295, 692, 491, 94};

System.out.println("Original array:");

printArray(arr); // Print the original unsorted array

RadixSort.radixSort(arr); // Sort the array using Radix Sort

System.out.println("Sorted array:");

printArray(arr); // Print the sorted array

}

// Utility method to print an array

public static void printArray(int[] arr) {

**for** (int num **:** arr) { // Loop through each element in the array

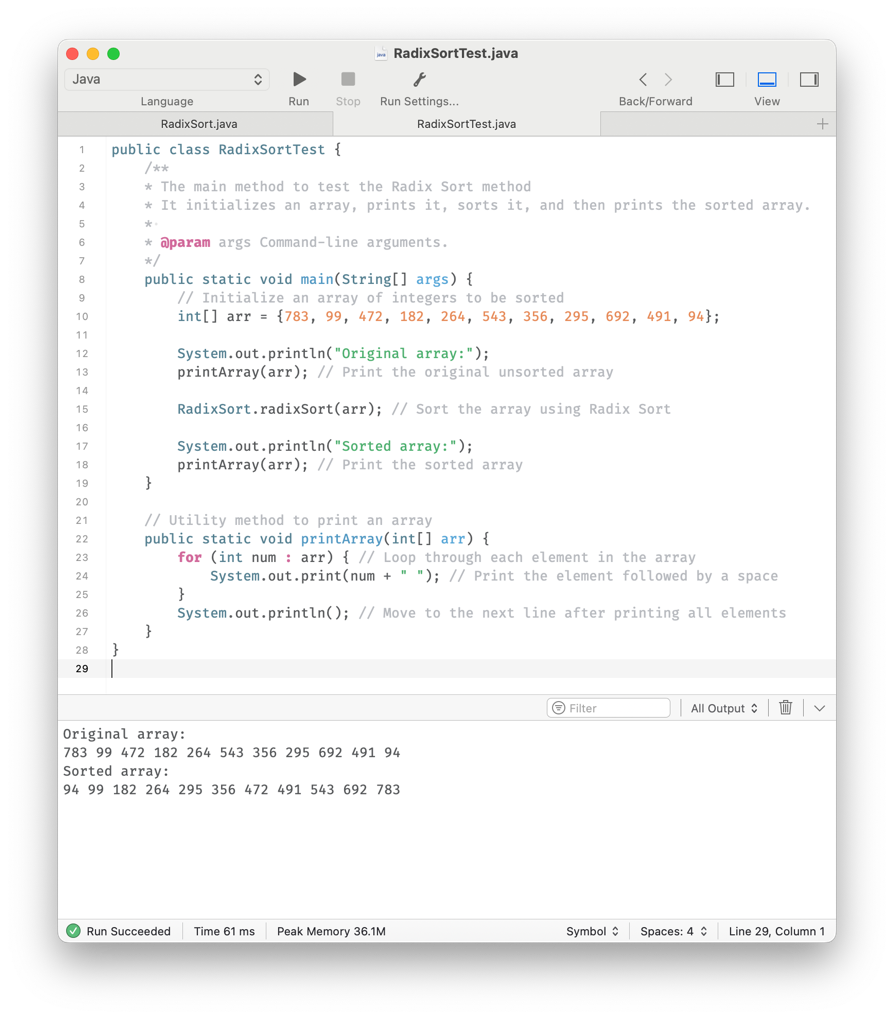
System.out.print(num + " "); // Print the element followed by a space

}

System.out.println(); // Move to the next line after printing all elements

}

}

**Output Screenshot  
**

**Big-Oh Analysis of the Algorithm**

**Time Complexity:**

* Let *n* be the number of elements in the array.
* Let *k* be the number of digits in the maximum number
* The getMax method runs in *O(n)* time.
* The outer loop in radixSort runs *k* times (once of each digit)
* The countingSort method runs in *O(n + b)* time for each digit, where *b* is the base (10 in this case)
* Since *b* is constant (10), *O(n + b)* simplifies to *O(n)*
* Therefore, the total time complexity is *O(k x n).*

**Space Complexity:**

* The algorithm uses additional arrays of size *n* and *b.*
* Since *b* is constant, the space complexity is *O(n).*

**Conclusion:**

* **Overall Time Complexity:** *O(n x k)*
* **Overall Space Complexity:** *O(n)*

1. **Step-by-Step Analysis of Radix Sort on the Given Array**

Given array:

[783, 99, 472, 182, 264, 543, 356, 295, 692, 491, 94]

**Maximum Number of Digits:**

The maximum number in the array is 783, which has 3 digits.

So, we need to perform 3 passes (for ones, tens, and hundreds digits).

**First Pass (Sorting by ones Digit, exp = 1):**

1. **Extract Units Digit and Count Frequencies:**

Extract ones place digits:

783 -> 3

99 -> 9

472 -> 2

182 -> 2

264 -> 4

543 -> 3

356 -> 6

295 -> 5

692 -> 2

491 -> 1

94 -> 4

Ones digits: [3, 9, 2, 2, 4, 3, 6, 5, 2, 1, 4]

Count after counting occurrences: [0, 1, 3, 2, 2, 1, 1, 0, 0, 1]

Cumulative counts: [0, 1, 4, 6, 8, 9, 10, 10, 10, 11]

**Output after first pass:**

[491, 472, 182, 692, 783, 543, 264, 94, 295, 356, 99]

**Second Pass (Sorting by Tens Digits):**

Extract tens digit and count frequency:

491 -> 9

472 -> 7

182 -> 8

692 -> 9

783 -> 8

543 -> 4

264 -> 6

94 -> 9

295 -> 9

356 -> 5

99 -> 9

Tens digits: [9,7,8,9,8,4,6,9,5,9]

Counting occurrences and cumulative counts:

Count after counting occurrences: [0,0,0,0,1,1,1,1,2,5]

Cumulative counts: [0,0,0,0,1,2,3,4,6,11]

**Output after second pass:**

[543, 356, 264, 472, 182, 783, 491, 692, 94, 295, 99]

**Third Pass (Sorting by Hundreds Digit):**

Extract hundreds digit and count frequency:

543 -> 5

356 -> 3

264 -> 2

472 -> 4

182 -> 1

783 -> 7

491 -> 4

692 -> 6

94 -> 0

295 -> 2

99 -> 0

Hundreds digits: [5,3,2,4,1,7,4,6,0,2,0]

Counting occurrences and cumulative counts:

Count after counting occurrences: [2,1,2,1,2,1,1,1,0,0]

Cumulative counts: [2,3,5,6,8,9,10,11,11,11]

**Output after third pass:**

[94, 99, 182, 264, 295, 356, 472, 491, 543, 692, 783]

1. **Big-Oh Analysis of Radix Sort Algorithm**

* **Time Complexity:** The Radix Sort algorithm processes each digit of the numbers. For n numbers and *d* digits (maximum number of digits in the largest number), the time complexity is *O(n x d).*
* **Space Complexity:** Additional space is used for the output array and the count array, both proportional to *n* and a constant size (10 for decimal digits), respectively. So, the space complexity is *O(n + k)*, where *k* is the base of the numbering system (10 in this case).

**Conclusion:** The Radix Sort algorithm operates in linear time relative to the number of elements when the number of digits d is constant, resulting in a time complexity of O(n) for fixed size integers.