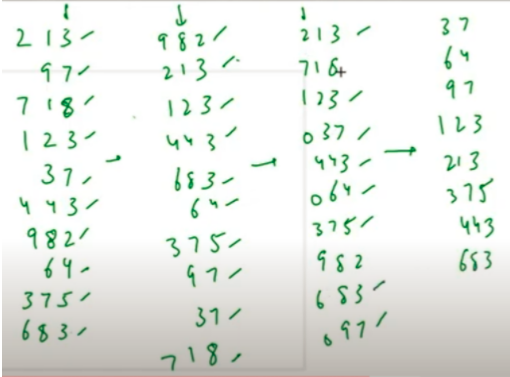
**1. Problem Discussion:**

You will be given an array of integers. You have to sort the given array in increasing order using the radix sort algorithm. It is recommended that you go and watch the video of count sort before heading over to this question, and understand it well enough, because the concept of count sort has been used here.

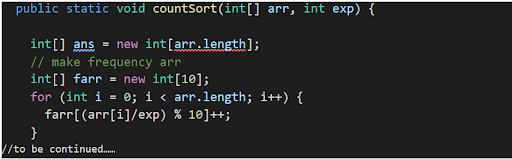
[**Count Sort**](https://www.youtube.com/embed/p-OyKUgIrx4)

**2. Approach:**

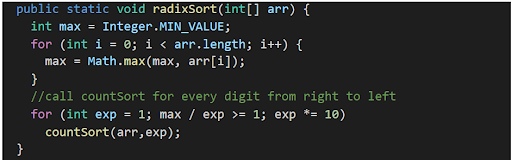
Let's start with an understanding of WHAT has to be done. Consider the input array depicted in the image's first column.

****

Step 1: Using stable count sort, scan the given array (arr[]) and sort it by unit place. The array should now resemble column 2 of the image. Step 2: Next, we use stable count sort to sort it by tens place. The array should now resemble column 3 of the image. Step 3: Finally, using stable count sort, sort on the hundredth place. The array should now resemble column 4 of the image. Why "finally"? Because in the preceding example, our maximum value is just three digits, i.e. only up to the hundredth place. In the IDE of the website, you’ll find that the snippet of the code which is provided contains the signature of count sort along with radix sort. Why? For radix sort, we will be using count sort also, However, we must change the Count Sort code since the code we saw in the last article applied to whole numbers rather than any single digit of that number, but when we rearrange these digits, complete numbers are considered. Furthermore, because the digits will only range from 0 to 9, the range will be 10.

****

Now we'll figure out how to handle individual digits in a count sort while still considering the complete number. Radix sort will be used to solve this problem.

****

To do so, we must first determine the array's maximum element. Then we call Count Sort and execute a for loop for exp (exponential), which determines how many times our loop will run. It will make calls equal to the number of digits in the maximum value, in simple terms. But how do you do it? The for loop's condition and updating sections assist us in accomplishing this. We double our exp by 10 after each iteration, and the process continues until the maximum value on division with exp is larger than 1. And, instead of farr[arr[i]-min], we'll use farr[arr[i] / exp percent 10] because our digits only range from 0 to 9 and we want to sort our array by digit at a specific location. Here’s the code for better and clear understanding of the approach discussed above:

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void radixSort(int[] arr) {

int max = Integer.MIN\_VALUE;

for (int i = 0; i < arr.length; i++) {

max = Math.max(max, arr[i]);

}

//call countSort for every digit from right to left

for (int exp = 1; max / exp >= 1; exp \*= 10)

countSort(arr, exp);

}

public static void countSort(int[] arr, int exp) {

int[] ans = new int[arr.length];

// make frequency arr

int[] farr = new int[10];

for (int i = 0; i < arr.length; i++) {

farr[(arr[i] / exp) % 10]++;

}

// convert it into prefix sum array

for (int i = 1; i < farr.length; i++) {

farr[i] += farr[i - 1];

}

// stable sorting(filling ans array)

for (int i = arr.length - 1; i >= 0; i--) {

int pos = farr[(arr[i] / exp) % 10] - 1;

ans[pos] = arr[i];

farr[(arr[i] / exp) % 10]--;

}

// filling original array with the help of ans array

for (int i = 0; i < arr.length; i++) {

arr[i] = ans[i];

}

System.out.print("After sorting on " + exp + " place -> ");

print(arr);

}

public static void print(int[] arr) {

for (int i = 0; i < arr.length; i++) {

System.out.print(arr[i] + " ");

}

System.out.println();

}

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[] arr = new int[n];

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

arr[i] = scn.nextInt();

}

radixSort(arr);

print(arr);

}

}

**3. Analysis:**

Time Complexity: O (bn)

We call count sort b times (b being the number of digits in maximum value). Count sort has an O(n+k) time complexity, however k is constant(10) in this case. As a result, it becomes bO(n) or O. (bn).

Space Complexity: O(n+b)