Lesson:


# Advanced Sorting Algorithms


PW SKILLS

# Pre-Requisites

- Java Loops
- Java Arrays

# List of Concepts Involved

- Count Sort
- Radix Sort
- Bucket Sort

# Topic 1: Count sort

This sorting technique is used when the range of input is limited.This sort is a sorting technique which is based on the range of input values. For example, if we use the array arr = {1, 2, 1, 3, 4, 5}. All the values of the array lie between 1 and 5. So we can use this method.

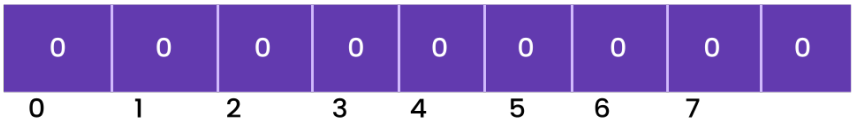
**Steps**

1. Find out the maximum element (let it be max) from the given array.
2. Initialize an array of length at least max+1 with all elements 0. This array is used for storing the frequency of the elements in the array.
3. Store the frequency of each element at their respective index in freq array
4. We will be using our original array only to store the sorted values.
5. Start iterating from 0 till max element and if an index has value greater than zero then store that particular index in the original array as our sorted value and reduce the count of that index by 1. Repeat this process till that element value decreases to 0. Once done you can move to the next index.
6. We can also start from the index with the minimum value instead of zero to save some iterations.
7. Since indices in the array are marked in sorted order from 0 to last index this will always ensure the array obtained will be in sorted order.

**Illustration:**
Consider the data in the range of 0 to 7.
**Input data:** {1, 4, 1, 2, 6, 5, 2}

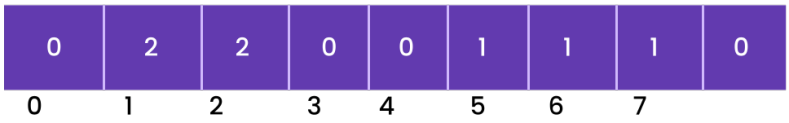
- Find out the maximum element (let it be max) from the given array.
  max = 6

- Take a count array of length max+1 with all elements 0 to store the count of each unique object.

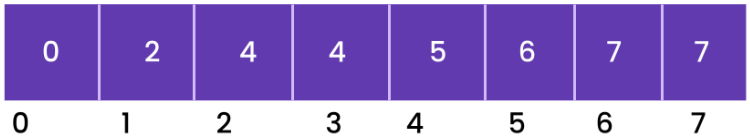


- Store the count of each unique element in the count array, if any element repeats itself, simply increase its count.

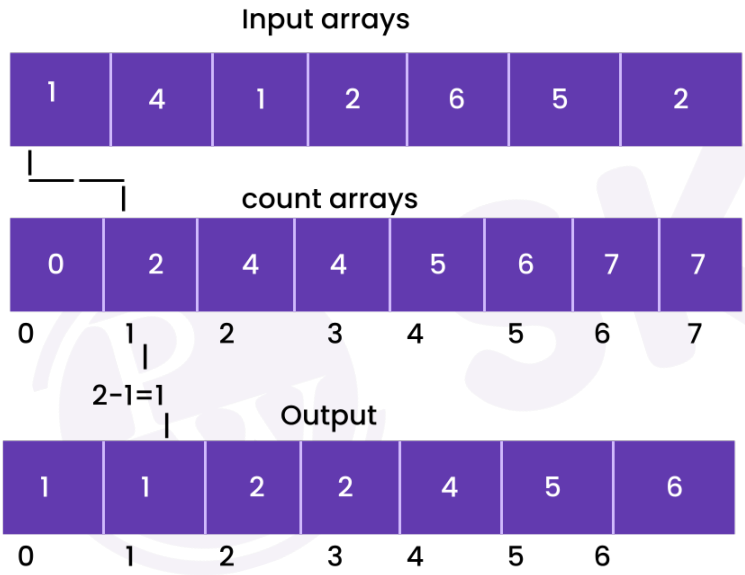
**Count array:**




- Modify the count array such that each element at each index stores the sum of previous counts.

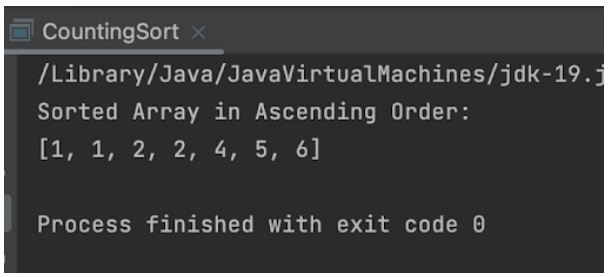



- Find the index of each element of the original array in the count array. This gives the cumulative count. Place the element at the index calculated as shown in figure below.




- After placing each element at its correct position, decrease its count by one.

**CODE :**
https://pastebin.com/P2uEcKp0



## TIME COMPLEXITY:

Best case: When all items are in the same range, or when k is equal to 1, k is the time required to find the maximum number.
O(n)

Average case: k computes to (k+1/2), and the time complexity is N+(K+1)/2. = O(n+k)

Worst case: The largest element is much larger than the other elements.
O(n+k)

## SPACE COMPLEXITY:

O(k)

# Topic 2: Radix sort

In Radix sort, digit by digit sorting is performed that is started from the least significant digit to the most significant digit.
Suppose, we have an array of 5 elements. First, we will sort elements based on the value of the unit place. Then, we will sort elements based on the value of the tenth place. This process goes on until the last significant place

**Steps:**

1. Find the maximum element in the array, i.e. max to calculate the no of digits because we have to go through all the significant places of all elements.
2. Now, go through each significant place one by one.
3. Any stable sorting technique can be used to sort the digits at each significant place.Count sort has been used here.
4. Now, sort the elements based on digits at the unit's place.
5. Now, sort the elements based on digits at tens place.
6. Finally, sort the elements based on the digits at hundreds place.

**Illustration:**

- Consider this input array:

| 170 | 45 | 75 | 90 | 802 | 2 |
|---|---|---|---|---|---|

- Sort on ones place.

| 170 | 90 | 802 | 2 | 45 | 75 |
|---|---|---|---|---|---|

Note, here 170 comes before 90 and 802 comes before 2 because it has appeared in this way in the original array.

- Sort on tens place.

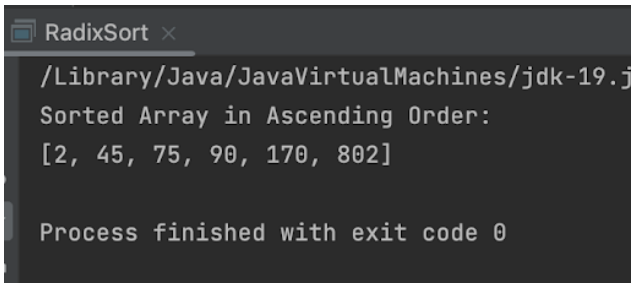
| | | | | | |
|---|---|---|---|---|---|
| 02 | 802 | 45 | 170 | 75 | 90 |

- Sort on hundreds place.

| | | | | | |
|---|---|---|---|---|---|
| 02 | 045 | 075 | 090 | 170 | 802 |

- The array is now sorted.

**Code:**
https://pastebin.com/5QNxq4dq




## TIME COMPLEXITY:

n = number of digits in the largest element

**Best case:** When all elements have the same number of digits.
O(a(n+b))

**Average case:** There are 'p' passes, and each digit can have up to 'd' different values. O(p*(n+d))

**Worst case:** When all elements have the same number of digits except one, which has a significantly larger number of digits.
$O(n^2)$

## SPACE COMPLEXITY:

O(n+k)

# Topic 3 : Bucket Sort

This is a sorting technique that separates an unsorted array of data into buckets. The buckets are then sorted using any of the appropriate sorting algorithms or recursively using the same bucket algorithm.
The sorted buckets are then joined to make a final sorted array.

**Steps:**

- **Step 1:** To know the number of passes required, first find the largest element of the array and count the number of digits.
- **Step 2:** Initialize buckets for all the digits starting from 0 to 9
- **Step 3:** Perform sorting by initially considering the least significant bit (LSB) i.e., one's digit, and insert the digit value in the respective bucket. For example, if LSB is 5 then the key is inserted in the 5th bucket, if it is 2 then the digit value is inserted in the 2nd bucket.
- **Step 4:** Now perform sorting by considering the second last digit and inserting the values in the respective bucket.
- **Step 5:** Repeat steps (4) and (5) for all the digits the digits are examined.

**Illustration:**

- Consider this input array:

| 0.42 | 0.32 | 0.23 | 0.52 | 0.25 | 0.47 | 0.51 |
|---|---|---|---|---|---|---|

- Create an array of size 10. Each slot of this array is used as a bucket for storing elements.

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|

- Insert elements into the buckets from the array. The elements are inserted according to the range of the bucket.
  In our example code, we have buckets each of ranges from 0 to 1, 1 to 2, 2 to 3,..... (n-1) to n.
  Suppose, an input element is .23 is taken. It is multiplied by size = 10 (ie. .23*10=2.3). Then, it is converted into an integer (ie. 2.3≈2). Finally, .23 is inserted into bucket-2.

| 0 | 0 | 0.23<br>0.25 | 0.32 | 0.42<br>0.47 | 0.52<br>0.51 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

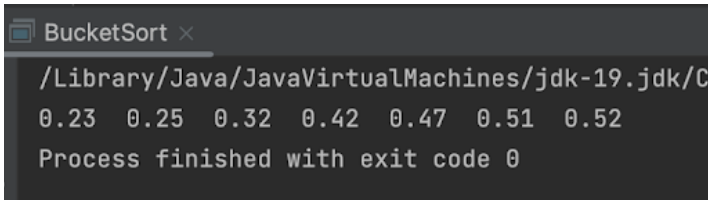
- The elements of each bucket are sorted using any of the stable sorting algorithms.

| 0 | 0 | 0.23<br>0.25 | 0.32 | 0.42<br>0.47 | 0.52<br>0.51 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

- The elements from each bucket are gathered. It is done by iterating through the bucket and inserting an individual element into the original array in each cycle. The element from the bucket is erased once it is copied into the original array.

| 0.23 | 0.25 | 0.32 | 0.42 | 0.47 | 0.51 | 0.52 |
|---|---|---|---|---|---|---|

**Code:**
https://pastebin.com/6Y9679MX




## TIME COMPLEXITY:

n = number of elements, k = number of buckets

**Best case:** When the elements are distributed in a balanced manner across the buckets.
O(n+k)

**Average case:** When the elements are distributed randomly in the list.
O(n)

**Worst case:** When every element is stored in the same bucket
$O(n^2)$

## SPACE COMPLEXITY:

O(n+k)

**Note :** It is not an in-place sorting algorithm,as the inputs are sorted by placing them into several buckets, so the sorting is not happening in-place here.

# Upcoming Class Teasers:

- Problems based on sorting algorithms.