**Counting Sort**

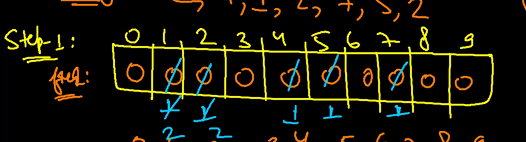
Array;

1,4,1,2,7,5,2

Step-1:

Count the frequency of the numbers. Range should be known.

It is encouraged only in unit digits. Here Range is 0 to 9.

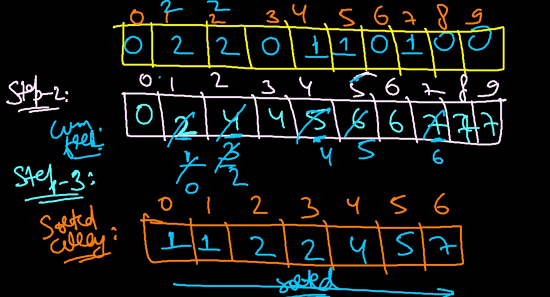


Step-2 and Step 3:

Cumulative frequency. (Keep adding the value to the previous value)

Index is the number and the value is the position.

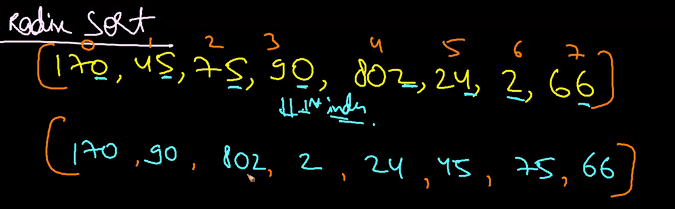
Traversal only in possible through right to left.



Time Complexity: O(Range+Arraysize)

It is beneficial is the Range=ArraySize -> O(n). It will be used in linear.

Radix Sort:

Digit Based sorting

After sorting it, we need to sort the number by next digit.

Then we get the array as,

**802,02,24,45,66,170,75,90**

After this based on the 3rd digit we need to sort

**002,024,045,066,075,090,170,804**

Hence we obtain the sorted array(Ascending order).

Counting Sort is used in Radix sort to fetch the digits and order them in ASC.

Enumeration:

170,45,75,90,802,24,2,66

Here range is 0 to 9. Because the digit will be of 0 to 9.

After taking the unit digit frequency we get,

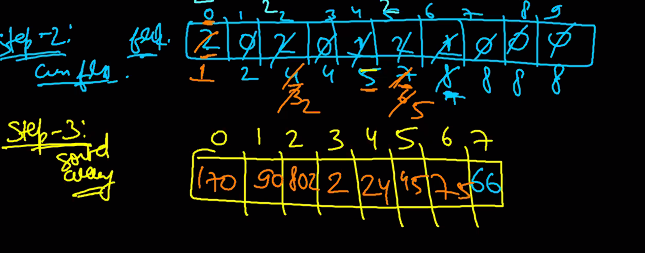
Step 1: 2,0,2,0,1,2,1,0,0,0

Step 2: Cumulative frequency

2,2,4,4,5,7,8,8,8,8

Step 3: We need to do from right to left.

After placing the number in the particular index decrement the frequency value by 1.



After completing the counting sort for the unit digit. Next we need to move to next digit. For that recommended condition is N%100.

**Time Complexity:**

b = base (10); Range(0 to 9)

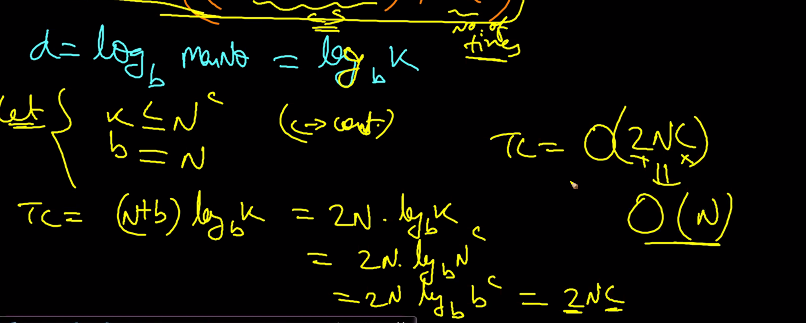
d = digit in maximum number. (number of steps that counting sort will work)

n = number of element

**T.Complexity of Radix sort: O((b+n)\*d)**

if the max number is k;

d=log(base b) K

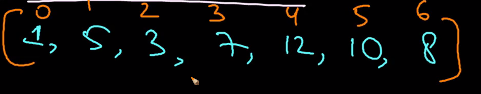


This will not have followed by all the programs. It works only if (b=n).

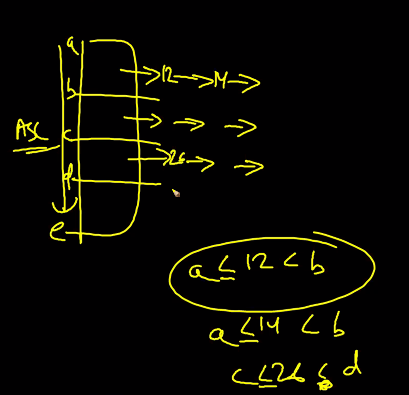
**8. Bucket Sort:**

Bucket sort is basically like throwing the elements in a bucket.

For example;



Create the bucket and throw them in the particular bucket according the values that we choose in it.

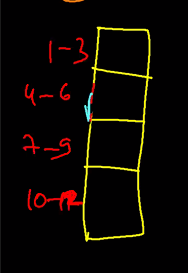


What factor the bucket need to be created?!

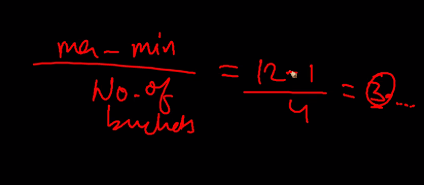
Min value < = elements <= Max value.

How many bucket we need?

Let us assume we are creating 4 buckets.



How we create 4 buckets in here?

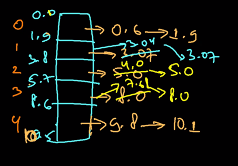


Whenever we get an element arr[i] and place them in a correct bucket?

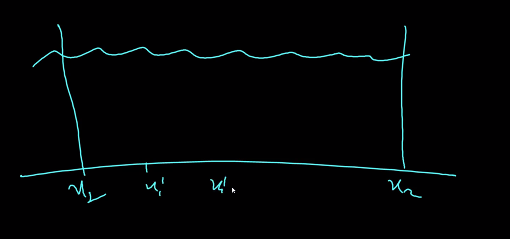
(arr[i]-minvalue/range);

Bucket starting index is 0;

Need to append the value in the bucket according to the range. We need to use the insertion sort to maintain them in ASC.



Even distribution;



Frequency is almost equal it is called the Even distribution.

This algorithm is applied **only when guaranteed with the Even Distribution**

**Time Complexity: O(1+k) or O(K)**

K defines the maximum list size. **More buckets, less list size. Less bucket, more list size.**

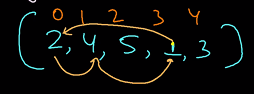
If the range is more the bucket will be more and less list of elements in the bucket.

**Cycle sort:**

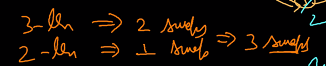
Let us take,

2,4,5,1,3

* Find the minimum number of swap to make the array sorted.



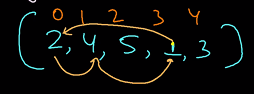
Find the swap cycle and swap the elements in the array.

There are 2 cycles in here -> 

If the write cost is high we go for this sorting.

Time Complexity : O(n^2) but best for write algorithm

Example;



Writes=0;

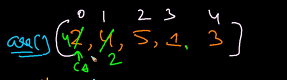
For(each cycle start\_pos(0 to 3)) ie 0 to (n-2)

Repeat

1. Item = all(cycle\_start)
2. Count the number which are less than item in the right side.
3. Increment the position in those many times.
4. Swap the item in the position.

Cycle start will be 2 in the array

* Get the count - Now need to check how many elements are less than 2 in the right side.
* The count will be 1. If the count is 0 stays in the same position. But the position is not correct as count is 1. Now swap the element to next index as it is 1.



* Now arr[0] has 4 and again get the count of the number lesser in the right side.

The count will be 3.

So it will move to the 3 position from the current position.

* In case the count is 0. the cycle start will be incremented by 1.
* We will keep on repeating all the elements in all the cycle.

Finally, we get the sorted array.

**Time Complexity of the algorithm is O(n^2).**