Sorting Algorithms

Bitonic Sort :-

→ Parallel sorting algorithms that performs $O(n \log^2 n)$ comparisions
→ No. of comparision is more than that in other algorithms.
→ It performs better for the parallel implementation because elements are compared in a pre-defined sequence.
→ The predefined sequence is called bitonic sequence.

In Bitonic sequence, elements are first arranged in increasing order, and then after particular index its decreasing.

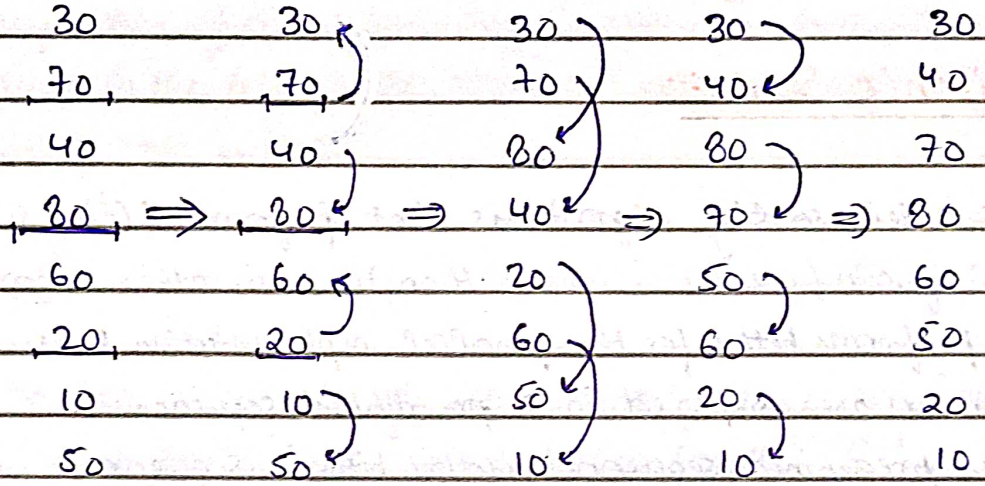
Conversion of Random Sequence into bitonic sequence :-

1. Let assume a elements of array i.e 8 elements in an array.
   i.e. {30, 70, 40, 80, 60, 20, 10, 50}
2. Divide the array into half i.e $n/2$.

30
70
40
80 — (1)
60
20
10
50

[Divide into n/2]

3. Now again divide elements of (1) into two half and again half and sort above two elements in ascending and below two elements into descending.



Bitonic Sequence

→ using array example :-

array[] = 30, 70, 40, 80, 60, 20, 10, 50

1→ To create bitonic sequence first create two subsequences one in ascending, another in descending order.

2→ create pair of elements.

array[] = {(30,70), (40,80), (60,20), (10,50)}
0 1 2 3

sort the pairs of 0,1,2,3 index.

array[] = {(30,70), (80,40), (20,60), (50,10)}

3→ Then create pair of these pairs in 4 elements in bitonic sequence and compare these elements which are at distance 2 i.e. i and i+2

array[] = {(30, 70, 80, 40), (20, 60, 50, 10)}
0 1 2 3 4 5 6 7

array[] = {(30, 40, 80, 70), (50, 60, 20, 10)}
1 2 3 4 4 5 6 7

4→ again follow step 2

array[] = {(30,40), (80,70), (50,60), (20,10)}

array[] = {(30,40), (70,80), (60,50), (20,10)} create sorting ~~sorting~~

→ bitonic sequence created.

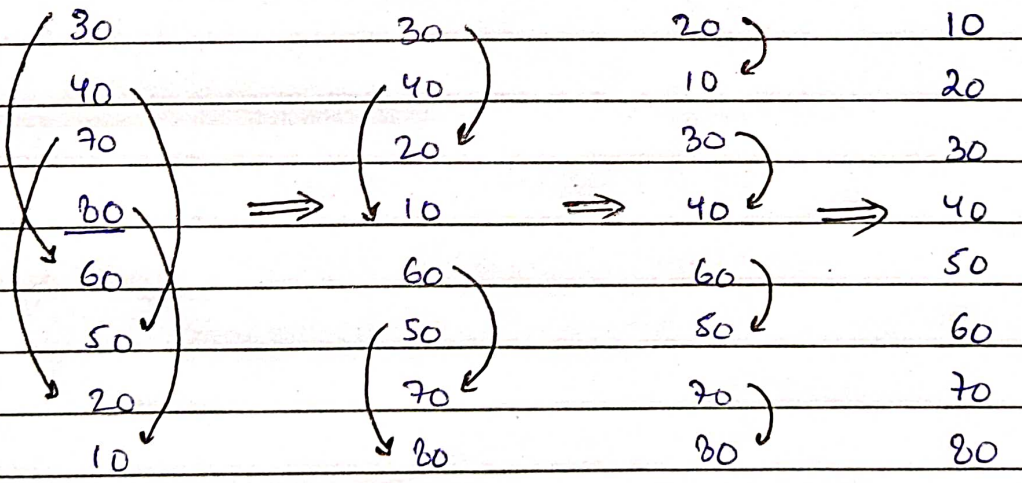
Bitonic sorting algorithm :-

1- Create bitonic sequence from the given random sequence, first half in ascending and second half in descending.

2- Compare first ~~step~~ element of first half with the first element of second half,
then the second element of second half and so on.
Swap elements if any element in second half is found to be smaller.

3- After step-2, all elements in the first half is smaller than all second half and so on.
Then compare swap results into the two sequences of n/2 length each ⟹

Repeat the process performed in 2 step recursively, until we get a sorted common length.



Bitonic Time Complexity :-

$O(\log^2 n)$

Space Complexity :-

$O(n \log^2 n)$

Not Stable :-

Radix Sort :-

Linear sorting algorithm used for integers
Digit by digit sorting is performed from the least significant digit (LSB) to most significant digit (MSB).

Algorithm :-

max → a b c
(a: most significant, c: least significant)

```
radixSort (arr)
max = largest element in the given array
d = no. of digits in the largest element (or, max)
Now, create d buckets of size 0-9.
for i = 0 to d
    sort the array elements using counting sort (or any stable sort)
    according the the digits at the ith place.
```

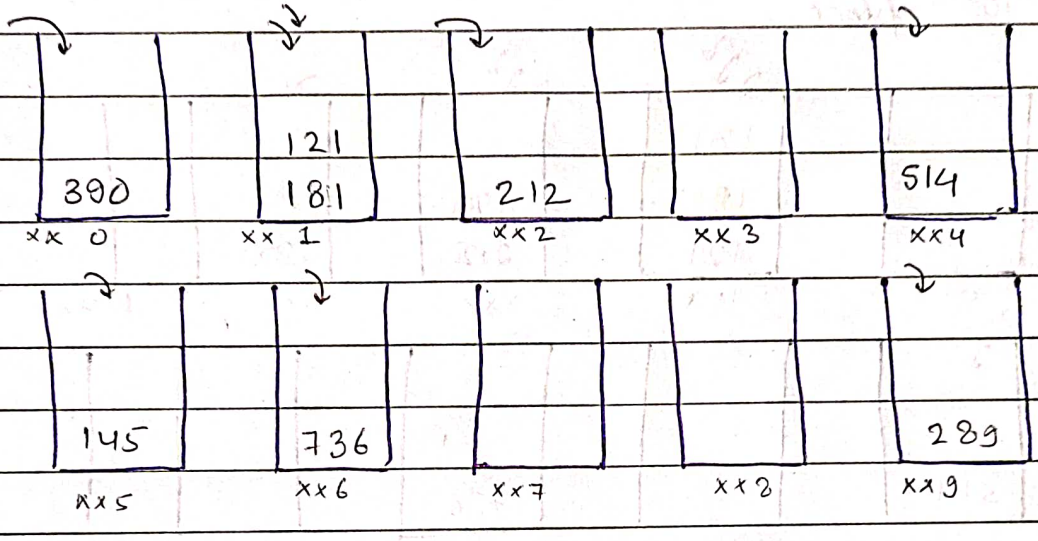
Working :-

Find the largest element (max) in array. x is the number of digits in max.

unsorted-array = [181, 289, 390, 121, 145, 736, 514, 212]
largest element = 736
    x or d = 3 (no. of digits)
∴ means loop will run three times, means we require three passes to sort the array.

Iteration - 1

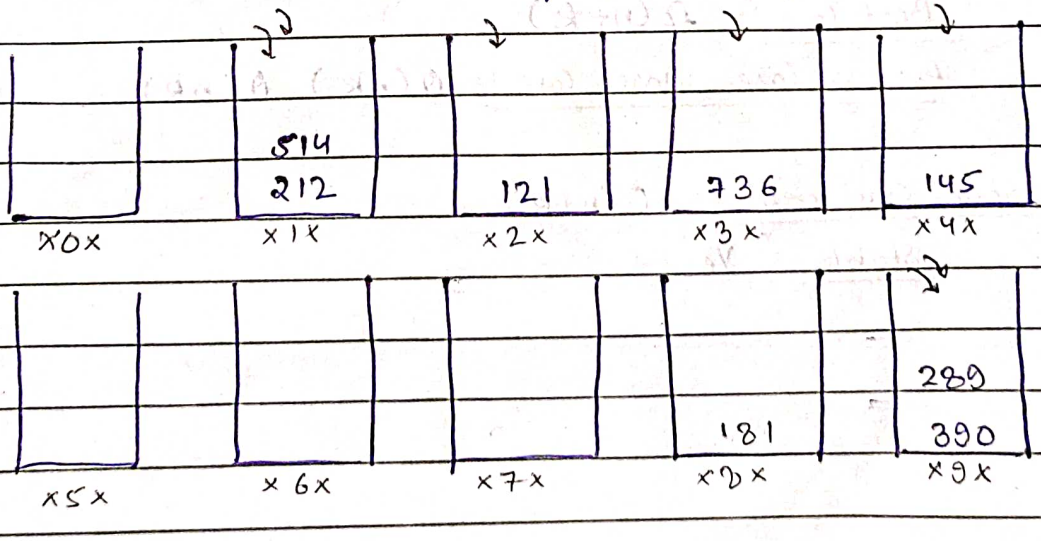
The list is sorted on the basis of the digit at 0's place means least significant digit.



The array elements are → 390, 181, 121, 212, 514, 145, 736, 289

Iteration - 2

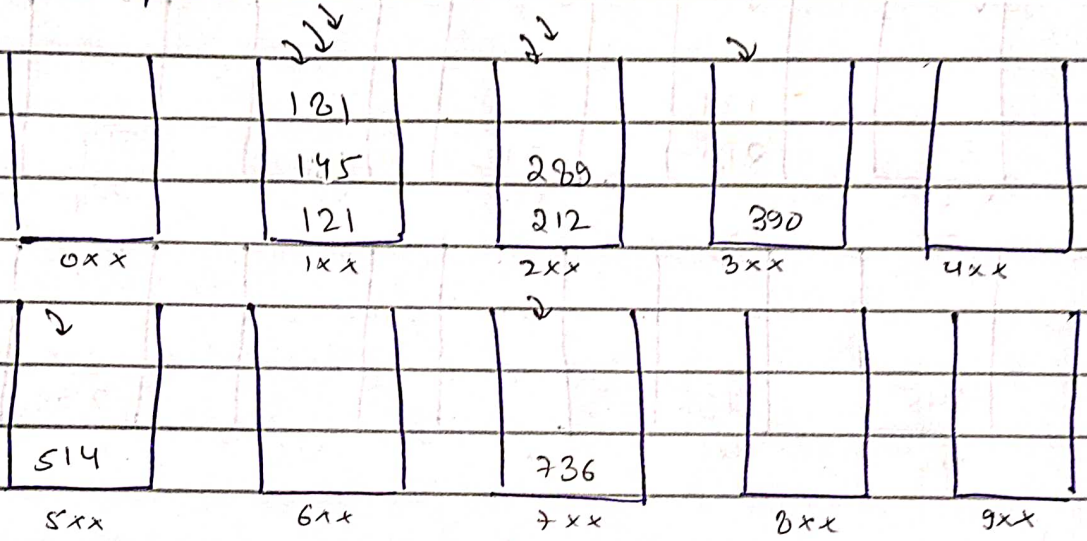
Sort the list for the next significant digit at 10th place.



The array elements are → 212, 514, 121, 736, 145, 181, 390, 289

Iteration - 3

Now solve and sort for the last i.e max significant digit. i.e. at 100th place.



The final sorted array :- 121, 145, 121, 212, 289, 396, 514, 736

Time Complexity :-

Best Case :- $\Omega(n+k)$

Average Case, Worst Case :- $O(nk)$, $O(nk)$

Space Complexity :- $O(n+k)$

Stable :- Yes.