A. Which, if any, of the following algorithms, bubble-sort, heap-sort, insertion sort, merge-sort, and quick-sort, are stable? Briefly justify your answer.

A sorting algorithm is stable if it does not change the order of the element with the same values

Example :bubble sort ,insertion sort ,merge sort are stable

Selection sort is unstable and heap sort also unstable

Sample for selection sort;

If arr=[4,4,2,1]

Using selection sort

4,4,2,1. Find the mini

4,4,2,1 swap

1,4,2,4 find mini

1,4,2,4 swap

1,2,4,4. So that her the order of 4 is changes the first 4 comes before the seconded 4 this is unstable;

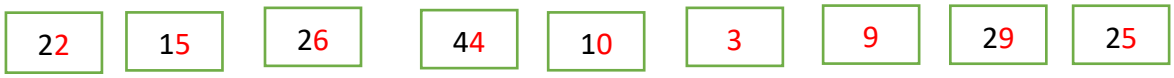
Out of the not inplace sort algorthim radix sort ,bucket sort are stable and PQ sort is unstable because of the buble down and buble up it may interchanges their order of the same values;

B. Is the bucket-sort algorithm in-place? Why or why not?

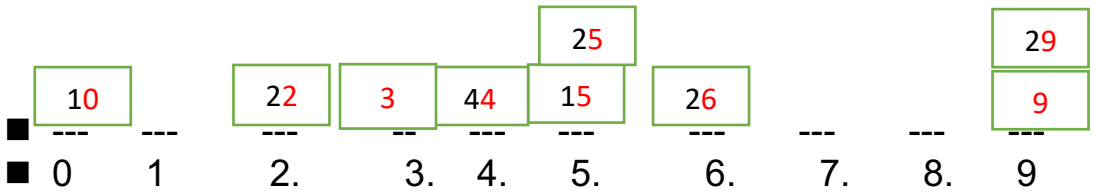
Not in place because this sorting techniques uses extra memory to sort the elements so that this is not in place sorting algorthim

C. Illustrate the performance of the radix-sort algorithm on the following input sequence

(22, 15, 26, 44, 10, 3, 9, 13, 29, 25).

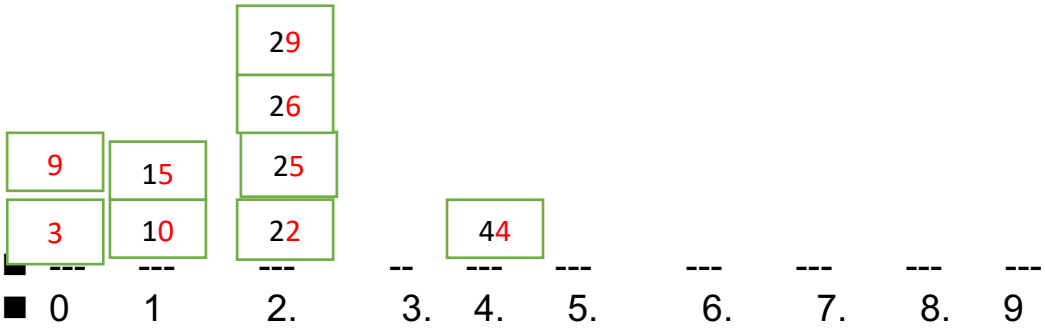


Step 1:



Step 2

Then compared the second digit



Finally it will be sorted

Result will be **[3,9,10,15,22,25,26,29,44]**

E. What can you conclude about the different sort algorithms?

Generally buble sort, shell sort, selction sort, heap sort, insertion sort are inplace sorting algorthim techiniqs that means the do not use extra memory to do sorting and PQ, radix, bucket sort are generally not inplace they use extra memory's to do sorting

For almost sorted elements and few elements insertion sort is better than the other techniques

But overall heap sort has better efficiency than the other inplace sorting algorthim

Generally, to compare all the sorting algorthim based on their time complexity and space complexity is

Radix sort are more efficient than the other not in place sorting algorthim

In place sorting algorthim					
	Time complexity			Space complexty	
Sorting Algorthim	best case	average case	wores case	worse case	stable
Bubble sort	O(N^2)	O(N^2)	O(N^2)	O(1)	yes
Selection sort	O(N^2)	O(N^2)	O(N^2)	O(1)	no
Insertion sort	O(N)	O(N^2)	O(N^2)	O(1)	yes
Heap sort	O(NlogN)	O(NlogN)	O(NlogN)	O(1)	no
Quick sort	?	?	?	?	?
not In place sorting algorthim					
	Time complexity			Space complexty	
Sorting Algorthim	best case	average case	wores case	worse case	stable
Bucket sort	O(N+k)	O(N+k)	O(N^2)	O(N)	yes
Radix Sort	O(N*k)	O(N*k)	O(N*k)	O(N)	yes
PQ sort	O(NlogN)	O(NlogN)	O(NlogN)	O(N)	no
Merge sort	?	?	?	?	?

```
Algorthim theSameSet(A,B)
    If A.length!=B.length              1
        return false;                  1
    PQSort(A)                          nlogn
   PqSort(B)                           nlogn
for i:=0 to A.length do n
      if A[i]!=B[i]                    n
            return false               1
  return true;                         1
```

Big O of this is O(nlogn+n)

```
function theSameSet(A,B){
    if(A.length!==B.length) return false;
    let sort=new Sorts.ArraySorter();
    sort.PQSort(A)
    sort.PQSort(B)
  for(let i=0;i<A.length;i++){
      if(A[i]!=B[i])
      return false
  }
 return true
}
let A= [1,4,3]
let B= [4,1,3]
console.log("expected true: "+ theSameSet(A,B))
let C=[9,1,3]
let D=[9,1,2]
console.log("expected false: "+ theSameSet(C,D))
```