geeksforgeeks.org

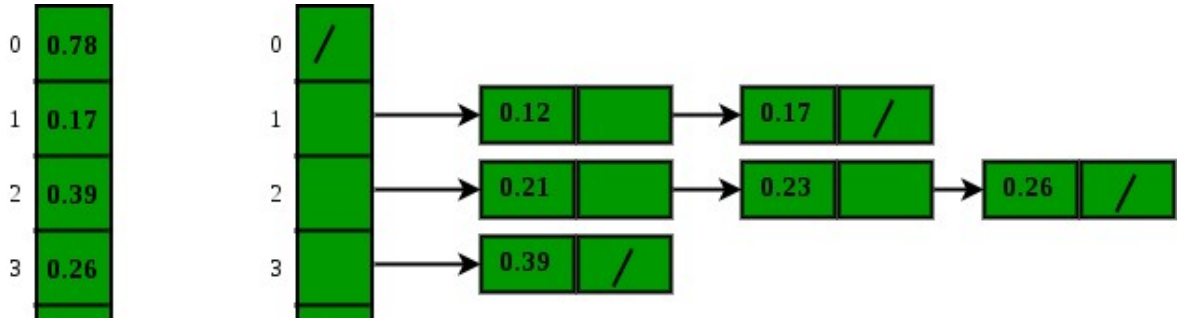
Bucket Sort - GeeksforGeeks

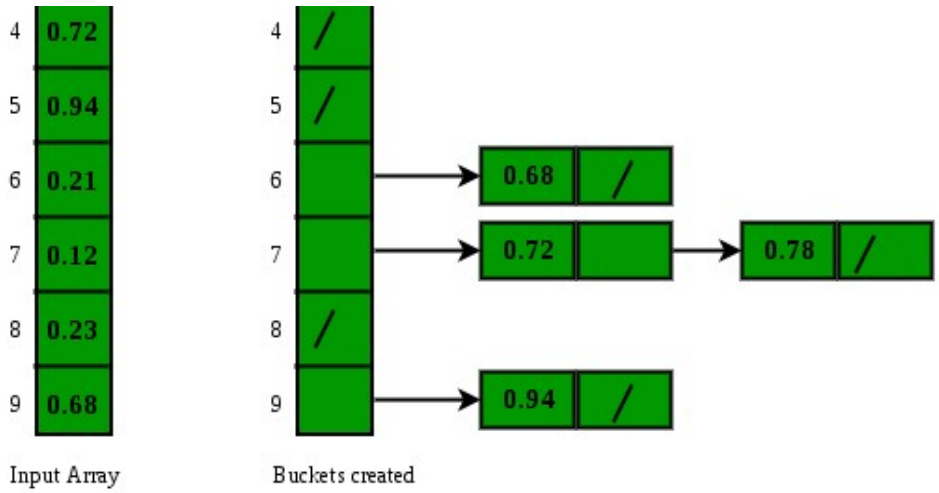
3-4 minutes

Bucket sort is mainly useful when input is uniformly distributed over a range. For example, consider the following problem.
Sort a large set of floating point numbers which are in range from 0.0 to 1.0 and are uniformly distributed across the range. How do we sort the numbers efficiently?

A simple way is to apply a comparison based sorting algorithm. The lower bound for Comparison based sorting algorithm (Merge Sort, Heap Sort, Quick-Sort .. etc) is Ω(n Log n), i.e., they cannot do better than nLogn.
Can we sort the array in linear time? Counting sort can not be applied here as we use keys as index in counting sort. Here keys are floating point numbers.
The idea is to use bucket sort. Following is bucket algorithm.

```
bucketSort(arr[], n)
1) Create n empty buckets (Or lists).
2) Do following for every array element arr[i].
.......a) Insert arr[i] into bucket[n*array[i]]
3) Sort individual buckets using insertion sort.
4) Concatenate all sorted buckets.
```





Time Complexity: If we assume that insertion in a bucket takes O(1) time then steps 1 and 2 of the above algorithm clearly take O(n) time. The O(1) is easily possible if we use a linked list to represent a bucket (In the following code, C++ vector is used for simplicity). Step 4 also takes O(n) time as there will be n items in all buckets.
The main step to analyze is step 3. This step also takes O(n) time on average if all numbers are uniformly distributed (please refer CLRS book for more details)

Following is C++ implementation of the above algorithm.

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
void bucketSort(float arr[], int n)
{
    vector<float> b[n];
    for (int i=0; i<n; i++)
    {
        int bi = n*arr[i];
```

```
        b[bi].push_back(arr[i]);
    }
    for (int i=0; i<n; i++)
        sort(b[i].begin(), b[i].end());
    int index = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < b[i].size(); j++)
          arr[index++] = b[i][j];
}
int main()
{
    float arr[] = {0.897, 0.565, 0.656, 0.1234,
0.665, 0.3434};
    int n = sizeof(arr)/sizeof(arr[0]);
    bucketSort(arr, n);
    cout << "Sorted array is \n";
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
    return 0;
}
```

Output:

```
Sorted array is
0.1234 0.3434 0.565 0.656 0.665 0.897
```

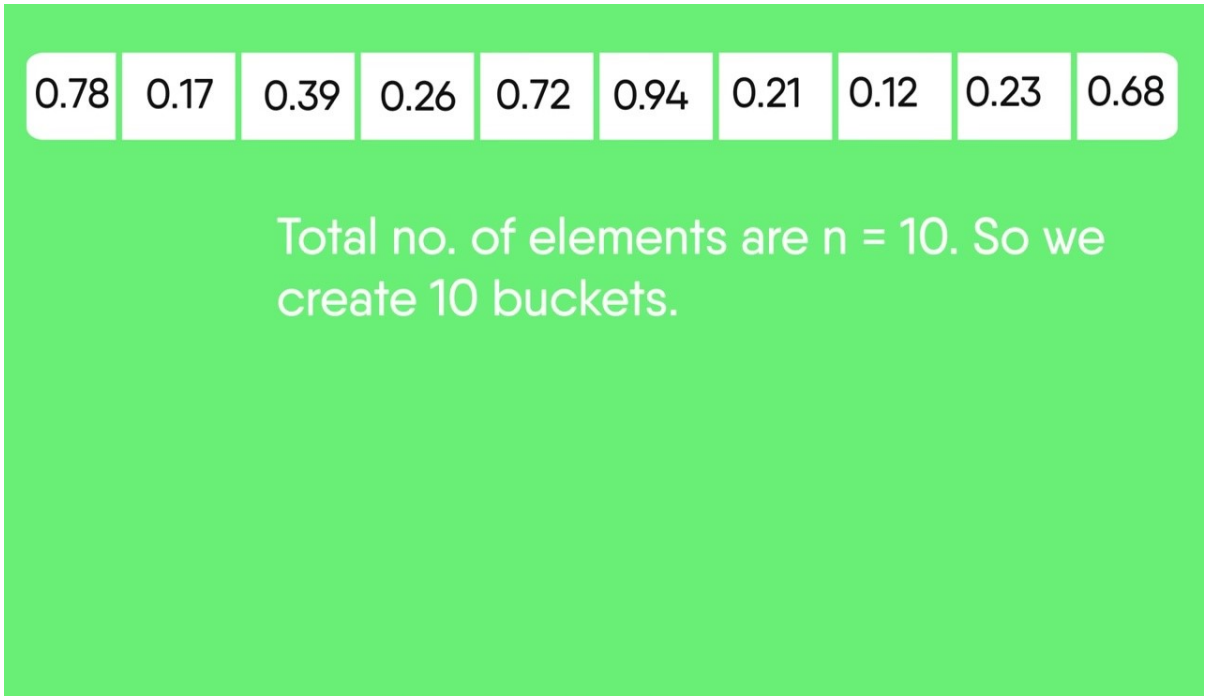
Bucket Sort To Sort an Array with Negative Numbers

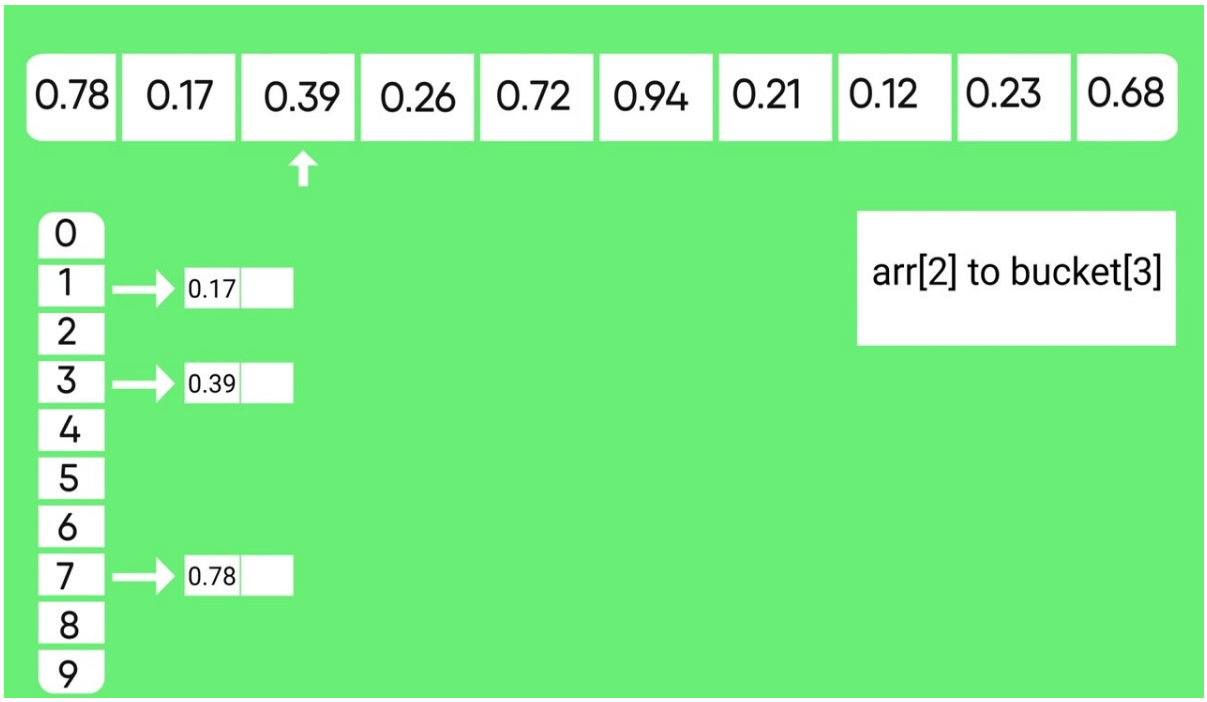
References:

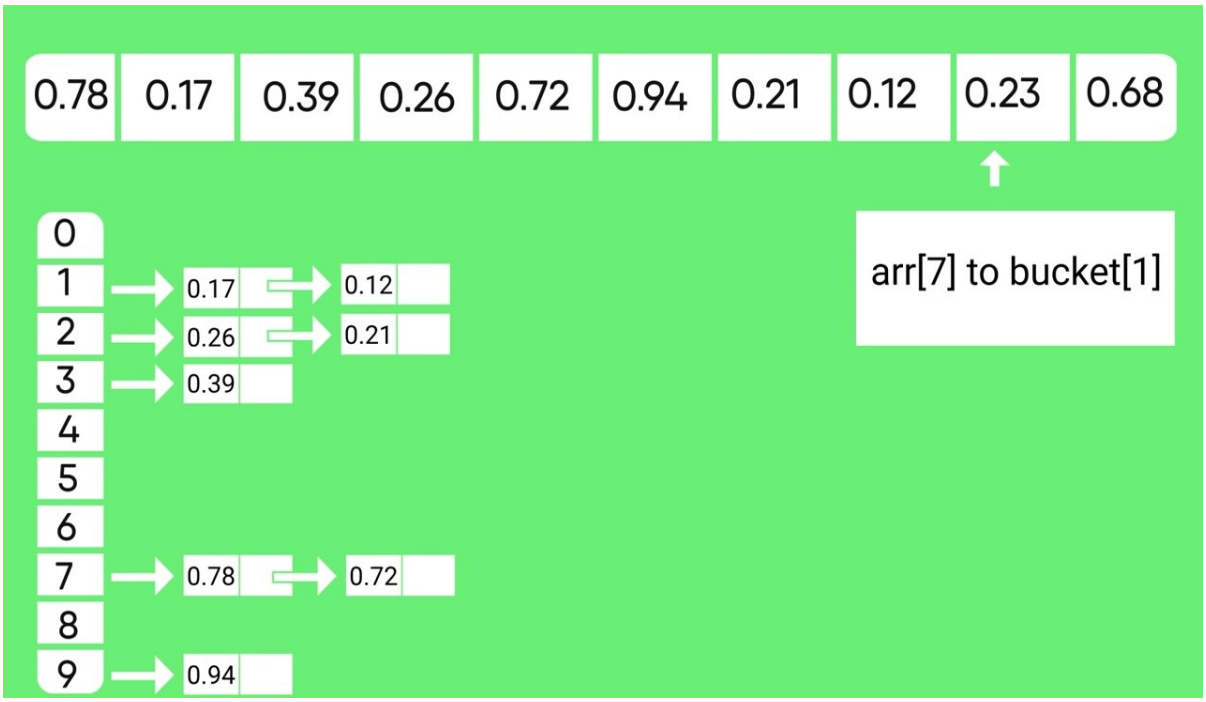
Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

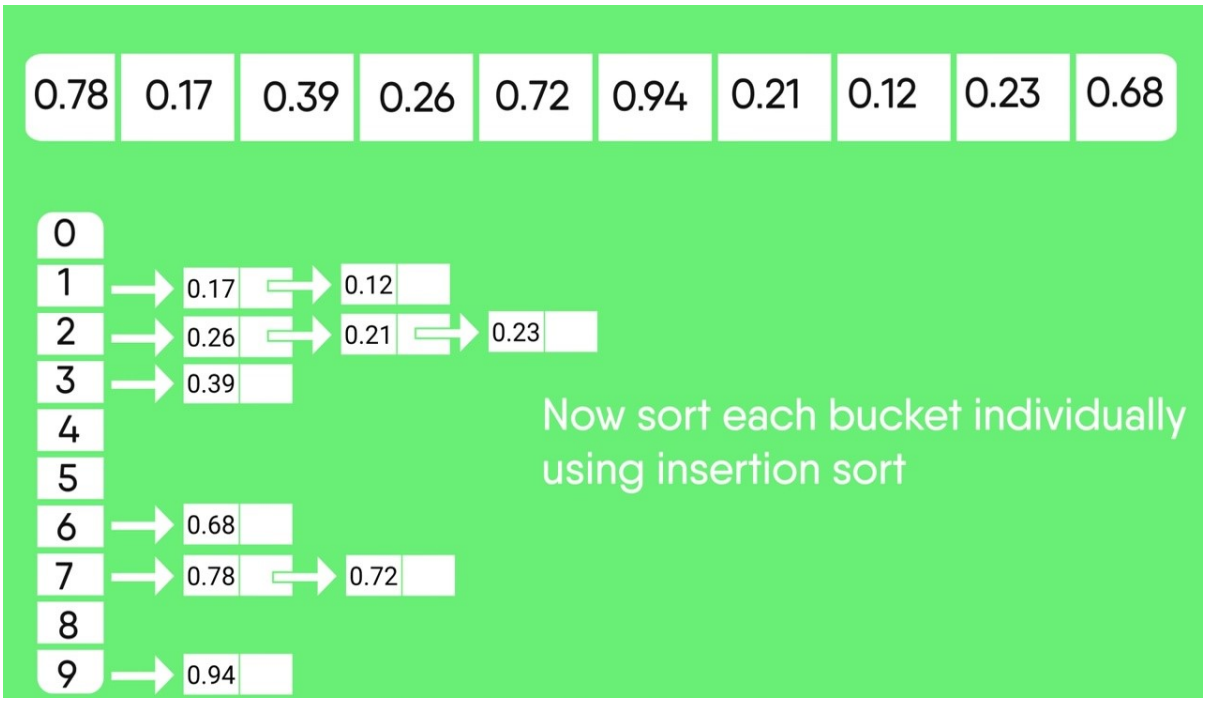
http://en.wikipedia.org/wiki/Bucket_sort

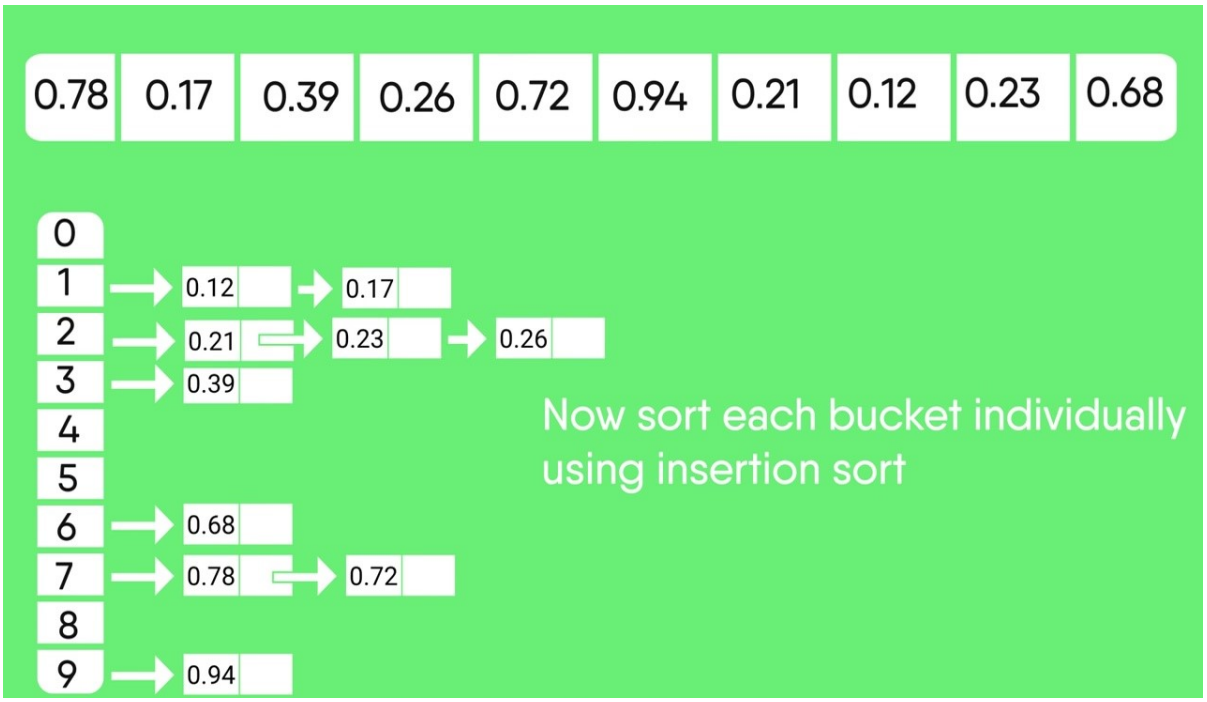
Snapshots:

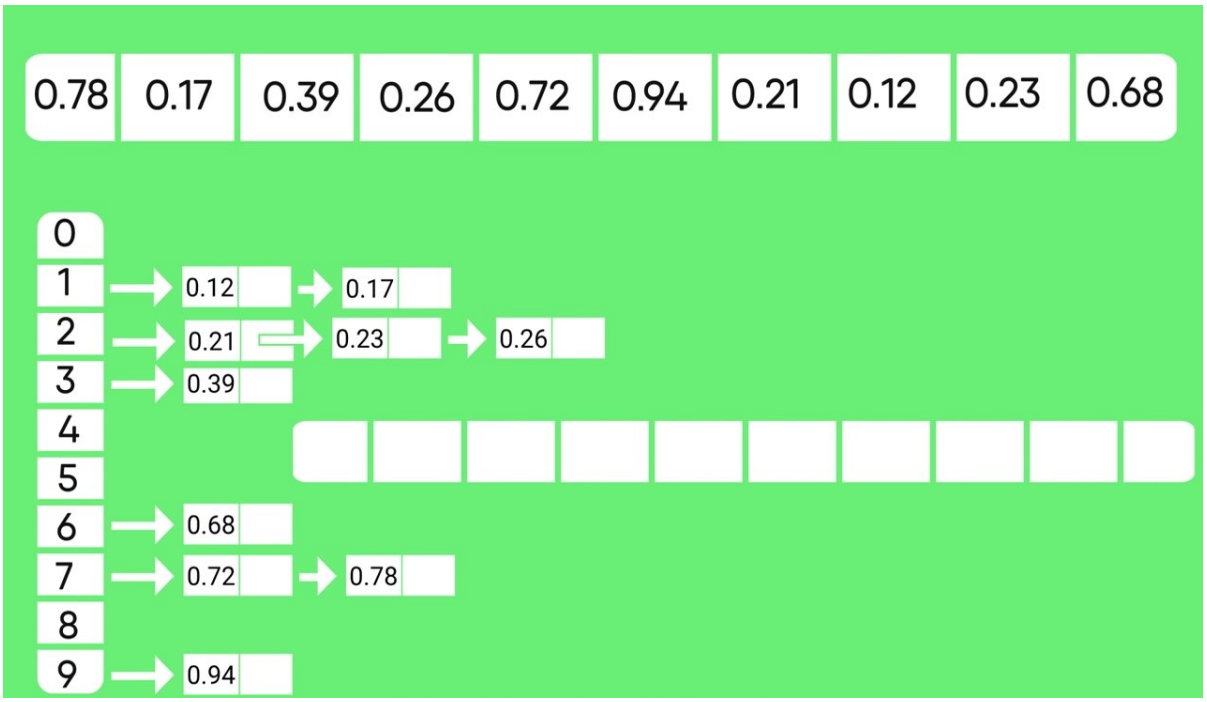


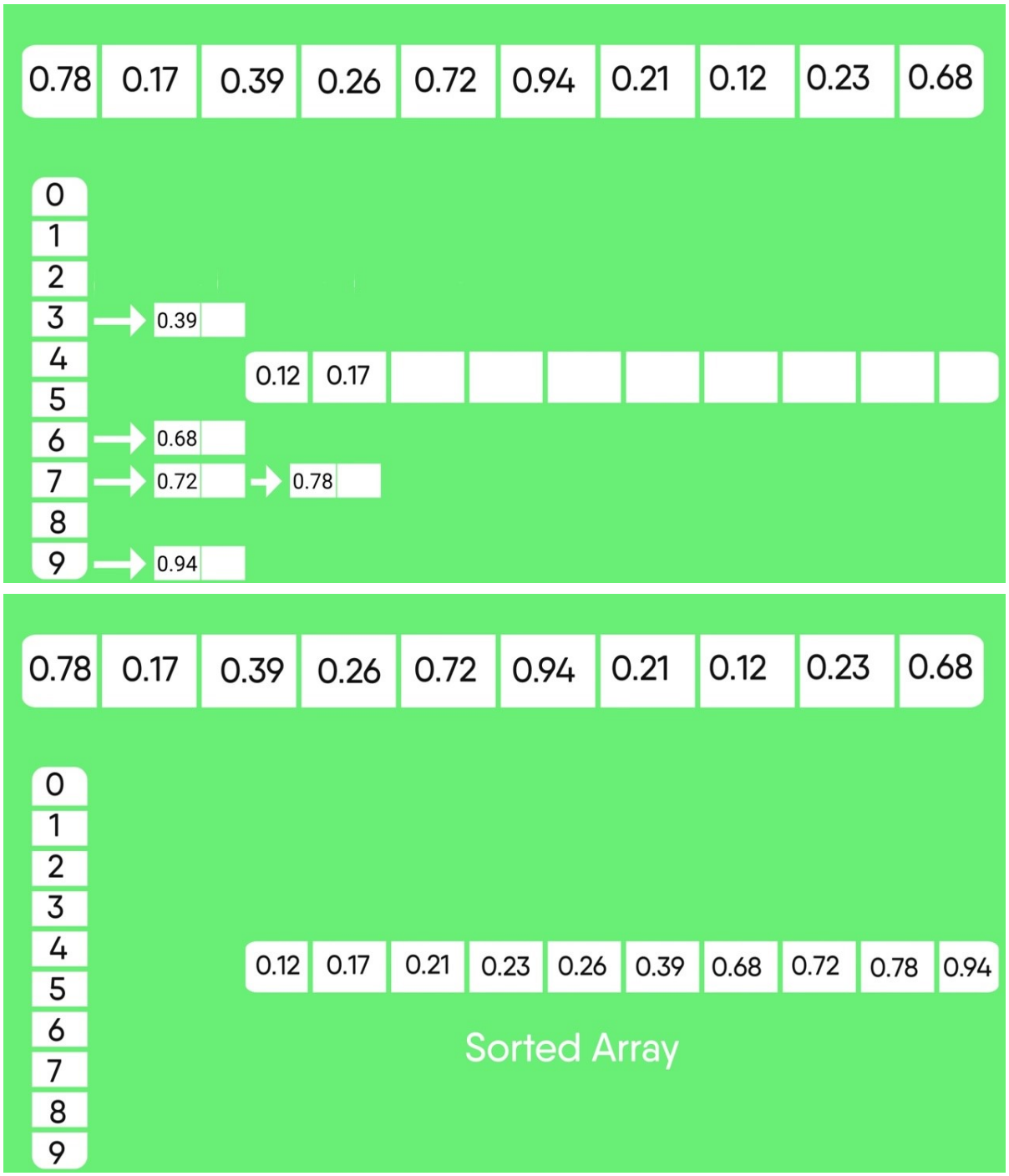



0.78 0.17 0.39 0.26 0.72 0.94 0.21 0.12 0.23 0.68
arr[7] to bucket[1]
0
1 0.17 0.12
2 0.26 0.21
3 0.39
4
5
6
7 0.78 0.72
8
9 0.94


0.78 0.17 0.39 0.26 0.72 0.94 0.21 0.12 0.23 0.68
0
1 0.17 0.12
2 0.26 0.21 0.23
3 0.39
4
5
6 0.68
7 0.78 0.72
8
9 0.94
Now sort each bucket individually
using insertion sort


0.78 0.17 0.39 0.26 0.72 0.94 0.21 0.12 0.23 0.68
0
1
0.12
0.17
2
0.21
0.23
0.26
3
0.39
4
5
6
0.68
7
0.78
0.72
8
9
0.94
Now sort each bucket individually using insertion sort


0.78 0.17 0.39 0.26 0.72 0.94 0.21 0.12 0.23 0.68
0
1
0.12
0.17
2
0.21
0.23
0.26
3
0.39
4
5
6
0.68
7
0.72
0.78
8
9
0.94



Quiz on Bucket Sort

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

- Selection Sort
- Bubble Sort
- Insertion Sort

- Merge Sort
- Heap Sort
- QuickSort
- Radix Sort
- Counting Sort
- ShellSort

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above