Bucket Sort

Bucket sort is mainly useful when input is uniformly distributed over a range. For example, consider the following problem.

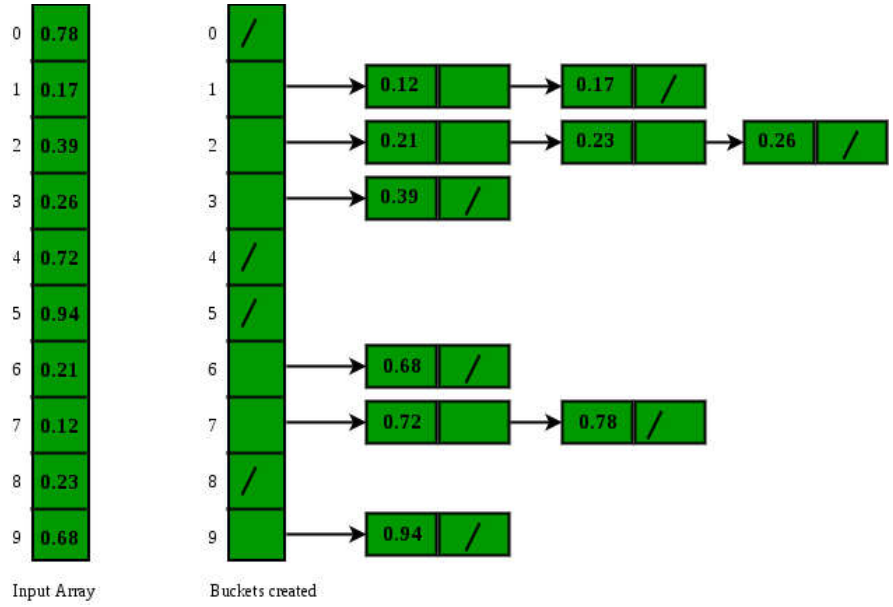
Sort a large set of floating point numbers which are in range from 0.0 to 1.0 and are uniformly distributed across the range. How do we sort the numbers efficiently?

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 the implementation of the above algorithm.

C++

```
// C++ program to sort an
// array using bucket sort
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

// Function to sort arr[] of
// size n using bucket sort
void bucketSort(float arr[], int n)
{

    // 1) Create n empty buckets
    vector<float> b[n];
```

```
        int bi = n * arr[i]; // Index in bucket
        b[bi].push_back(arr[i]);
    }

    // 3) Sort individual buckets
    for (int i = 0; i < n; i++)
        sort(b[i].begin(), b[i].end());

    // 4) Concatenate all buckets into arr[]
    int index = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < b[i].size(); j++)
            arr[index++] = b[i][j];
}

/* Driver program to test above function */
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;
}
```

Java

```
// Java program to sort an array
// using bucket sort
import java.util.*;
import java.util.Collections;

class GFG {

    // Function to sort arr[] of size n
    // using bucket sort
    static void bucketSort(float arr[], int n)
    {
        if (n <= 0)
            return;

        // 1) Create n empty buckets
        @SuppressWarnings("unchecked")
```

```
        }

        // 2) Put array elements in different buckets
        for (int i = 0; i < n; i++) {
            float idx = arr[i] * n;
            buckets[(int)idx].add(arr[i]);
        }

        // 3) Sort individual buckets
        for (int i = 0; i < n; i++) {
            Collections.sort(buckets[i]);
        }

        // 4) Concatenate all buckets into arr[]
        int index = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < buckets[i].size(); j++) {
                arr[index++] = buckets[i].get(j);
            }
        }
    }

    // Driver code
    public static void main(String args[])
    {
        float arr[] = { (float)0.897, (float)0.565,
                        (float)0.656, (float)0.1234,
                        (float)0.665, (float)0.3434 };

        int n = arr.length;
        bucketSort(arr, n);

        System.out.println("Sorted array is ");
        for (float el : arr) {
            System.out.print(el + " ");
        }
    }
}

// This code is contributed by Himangshu Shekhar Jha
```

Python3

```
# Python3 program to sort an array
# using bucket sort
def insertionSort(b):
    for i in range(1, len(b)):
```

```
            j -= 1
        b[j + 1] = up
    return b

def bucketSort(x):
    arr = []
    slot_num = 10 # 10 means 10 slots, each
                  # slot's size is 0.1
    for i in range(slot_num):
        arr.append([])

    # Put array elements in different buckets
    for j in x:
        index_b = int(slot_num * j)
        arr[index_b].append(j)

    # Sort individual buckets
    for i in range(slot_num):
        arr[i] = insertionSort(arr[i])

    # concatenate the result
    k = 0
    for i in range(slot_num):
        for j in range(len(arr[i])):
            x[k] = arr[i][j]
            k += 1
    return x

# Driver Code
x = [0.897, 0.565, 0.656,
     0.1234, 0.665, 0.3434]
print("Sorted Array is")
print(bucketSort(x))

# This code is contributed by
# Oneil Hsiao
```

C#

```
// C# program to sort an array
// using bucket sort
using System;
using System.Collections;
using System.Collections.Generic;
class GFG {

  // Function to sort arr[] of size n
```

```
        return;

      // 1) Create n empty buckets
      List<float>[] buckets = new List<float>[n];

      for (int i = 0; i < n; i++) {
        buckets[i] = new List<float>();
      }

      // 2) Put array elements in different buckets
      for (int i = 0; i < n; i++) {
        float idx = arr[i] * n;
        buckets[(int)idx].Add(arr[i]);
      }

      // 3) Sort individual buckets
      for (int i = 0; i < n; i++) {
        buckets[i].Sort();
      }

      // 4) Concatenate all buckets into arr[]
      int index = 0;
      for (int i = 0; i < n; i++) {
        for (int j = 0; j < buckets[i].Count; j++) {
          arr[index++] = buckets[i][j];
        }
      }
    }

    // Driver code
    public static void Main()
    {
      float []arr = { (float)0.897, (float)0.565,
                      (float)0.656, (float)0.1234,
                      (float)0.665, (float)0.3434 };

      int n = arr.Length;
      bucketSort(arr, n);

      Console.WriteLine("Sorted array is ");
      foreach(float el in arr) {
        Console.Write(el + " ");
      }
    }
}

// This code is contributed by rutvik_56
```

```
// using bucket sort

// Function to sort arr[] of size n
// using bucket sort
function bucketSort(arr,n)
{
    if (n <= 0)
            return;

        // 1) Create n empty buckets
        let buckets = new Array(n);

        for (let i = 0; i < n; i++)
        {
            buckets[i] = [];
        }

        // 2) Put array elements in different buckets
        for (let i = 0; i < n; i++) {
            let idx = arr[i] * n;
            let flr = Math.floor(idx);
            buckets[flr].push(arr[i]);
        }

        // 3) Sort individual buckets
        for (let i = 0; i < n; i++) {
            buckets[i].sort(function(a,b){return a-b;});
        }

        // 4) Concatenate all buckets into arr[]
        let index = 0;
        for (let i = 0; i < n; i++) {
            for (let j = 0; j < buckets[i].length; j++) {
                arr[index++] = buckets[i][j];
            }
        }
}

// Driver code
let arr = [0.897, 0.565,
         0.656, 0.1234,
         0.665, 0.3434];
let n = arr.length;
bucketSort(arr, n);

document.write("Sorted array is <br>");
for (let el of arr.values()) {
    document.write(el + " ");
```

Output

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

Bucket Sort for numbers having integer part:

Algorithm :

1. Find maximum element and minimum of the array
2. Calculate the range of each bucket

```
range = (max - min) / n
n is the number of buckets
```

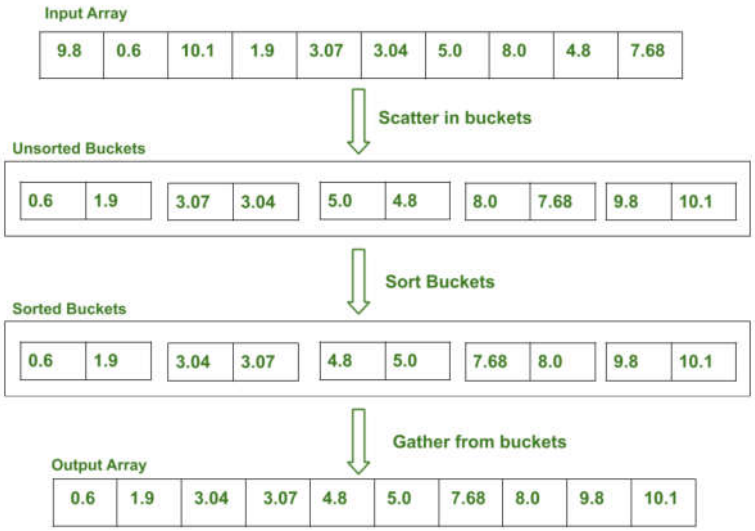
3. Create n buckets of calculated range
4. Scatter the array elements to these buckets

```
BucketIndex = ( arr[i] - min ) / range
```

5. Now sort each bucket individually
6. Gather the sorted elements from buckets to original array

Input :
Unsorted array: [9.8 , 0.6 , 10.1 , 1.9 , 3.07 , 3.04 , 5.0 , 8.0 , 4.8 , 7.68]
No of buckets : 5

Output :
Sorted array: [0.6 , 1.9 , 3.04 , 3.07 , 4.8 , 5.0 , 7.68 , 8.0 , 9.8 , 10.1]




```
Input :
Unsorted array:  [0.49 , 5.9 , 3.4 , 1.11 , 4.5 , 6.6 , 2.0]
No of buckets: 3

Output :
Sorted array:   [0.49 , 1.11 , 2.0 , 3.4 , 4.5 , 5.9 , 6.6]
```

Code :

C++

```
#include <algorithm>
#include <iostream>
#include <vector>

using namespace std;

// Bucket sort for numbers having integer part
void bucketSort(vector<double>& arr, int noOfBuckets)
{
  double max_ele = *max_element(arr.begin(), arr.end());
  double min_ele = *min_element(arr.begin(), arr.end());

  // range (for buckets)
  double rnge = (max_ele - min_ele) / noOfBuckets;

  vector<vector<double> > temp;
```

```
    }

    // scatter the array elements into the correct bucket
    for (int i = 0; i < arr.size(); i++) {
      double diff = (arr[i] - min_ele) / rnge
        - int((arr[i] - min_ele) / rnge);

      // append the boundary elements to the lower array
      if (diff == 0 && arr[i] != min_ele) {
        temp[int((arr[i] - min_ele) / rnge) - 1]
          .push_back(arr[i]);
      }
      else {
        temp[int((arr[i] - min_ele) / rnge)].push_back(
          arr[i]);
      }
    }

    // Sort each bucket individually
    for (int i = 0; i < temp.size(); i++) {
      if (!temp[i].empty()) {
        sort(temp[i].begin(), temp[i].end());
      }
    }

    // Gather sorted elements to the original array
    int k = 0;
    for (vector<double>& lst : temp) {
      if (!lst.empty()) {
        for (double i : lst) {
          arr[k] = i;
          k++;
        }
      }
    }
}

int main()
{
    vector<double> arr = { 9.8,  0.6, 10.1, 1.9, 3.07,
                           3.04, 5.0, 8.0,  4.8, 7.68 };
    int noOfBuckets = 5;
    bucketSort(arr, noOfBuckets);
    cout << "Sorted array: ";
    for (double i : arr) {
      cout << i << " ";
    }
    cout << endl;
```

Java

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

// Main class
public class GFG
{

  // bucketSort method that sorts an array of double values using bucket sort
  public static void bucketSort(double[] arr, int noOfBuckets)
  {

    // find the max and min elements in the array
    double maxEle = Arrays.stream(arr).max().getAsDouble();
    double minEle = Arrays.stream(arr).min().getAsDouble();

    // find the range between the max and min elements
    double range = (maxEle - minEle) / noOfBuckets;

    // create a list of empty lists to store elements based on their bucket in
    List<List<Double>> temp = new ArrayList<>();
    for (int i = 0; i < noOfBuckets; i++) {
      temp.add(new ArrayList<>());
    }

    // distribute the elements of the array into the appropriate bucket based
    for (int i = 0; i < arr.length; i++)
    {
      double diff = (arr[i] - minEle) / range - (int)((arr[i] - minEle) / rang

      // check if the difference is 0, and the element is not the min element
      if (diff == 0 && arr[i] != minEle) {
        temp.get((int)((arr[i] - minEle) / range) - 1).add(arr[i]);
      } else {
        temp.get((int)((arr[i] - minEle) / range)).add(arr[i]);
      }
    }

    // sort each non-empty bucket using the internal sort method
    for (int i = 0; i < temp.size(); i++) {
      if (!temp.get(i).isEmpty()) {
        temp.get(i).sort(Double::compare);
      }
    }
```

```
        if (!lst.isEmpty()) {
          for (double i : lst) {
            arr[k] = i;
            k++;
          }
        }
      }
    }

    public static void main(String[] args) {

      double[] arr = { 9.8, 0.6, 10.1, 1.9, 3.07, 3.04, 5.0, 8.0, 4.8, 7.68 };
      int noOfBuckets = 5;

      // call the bucket sort method
      bucketSort(arr, noOfBuckets);

      // print the sorted array
      System.out.print("Sorted array: ");
      for (double i : arr) {
        System.out.print(i + " ");
      }
      System.out.println();
    }
}

// This code is contributed by chinmaya121221
```

Python3

```
# Python program for the above approach

# Bucket sort for numbers
# having integer part
def bucketSort(arr, noOfBuckets):
    max_ele = max(arr)
    min_ele = min(arr)

    # range(for buckets)
    rnge = (max_ele - min_ele) / noOfBuckets

    temp = []

    # create empty buckets
    for i in range(noOfBuckets):
        temp.append([])
```

```
        diff = (arr[i] - min_ele) / rnge -  int((arr[i] - min_ele) / rnge)

        # append the boundary elements to the lower array
        if(diff == 0 and arr[i] != min_ele):
            temp[int((arr[i] - min_ele) / rnge) - 1].append(arr[i])

        else:
            temp[int((arr[i] - min_ele) / rnge)].append(arr[i])

    # Sort each bucket individually
    for i in range(len(temp)):
        if len(temp[i]) != 0:
            temp[i].sort()

    # Gather sorted elements
    # to the original array
    k = 0
    for lst in temp:
        if lst:
            for i in lst:
                arr[k] = i
                k = k+1


# Driver Code
arr = [9.8, 0.6, 10.1, 1.9, 3.07, 3.04, 5.0, 8.0, 4.8, 7.68]
noOfBuckets = 5
bucketSort(arr, noOfBuckets)
print("Sorted array: ", arr)

# This code is contributed by
# Vinita Yadav
```

C#

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace BucketSort
{
  class Program
  {
    static void Main(string[] args)
    {
      List<double> arr = new List<double> { 9.8, 0.6, 10.1, 1.9, 3.07, 3.04, 5
      int noOfBuckets = 5;
```

```
// bucketSort method that sorts an array of double values using bucket sor
static void BucketSort(List<double> arr, int noOfBuckets)
{

  // find the max and min elements in the array
  double max_ele = arr.Max();
  double min_ele = arr.Min();

  // range (for buckets)
  double rnge = (max_ele - min_ele) / noOfBuckets;

  List<List<double>> temp = new List<List<double>>();

  // create empty buckets
  for (int i = 0; i < noOfBuckets; i++)
  {
    temp.Add(new List<double>());
  }

  // scatter the array elements into the correct bucket
  for (int i = 0; i < arr.Count; i++)
  {
    double diff = (arr[i] - min_ele) / rnge - (int)((arr[i] - min_ele) / r

    // append the boundary elements to the lower array
    if (diff == 0 && arr[i] != min_ele)
    {
      temp[(int)((arr[i] - min_ele) / rnge) - 1].Add(arr[i]);
    }
    else
    {
      temp[(int)((arr[i] - min_ele) / rnge)].Add(arr[i]);
    }
  }

  // Sort each bucket individually
  for (int i = 0; i < temp.Count; i++)
  {
    if (temp[i].Count != 0)
    {
      temp[i].Sort();
    }
  }

  // Gather sorted elements to the original array
  int k = 0;
  foreach (List<double> lst in temp)
  {
```

```
            k++;
          }
        }
      }
    }
  }
}
// This code is contributed by divyansh2212
```

Javascript

```
function bucketSort(arr, noOfBuckets) {
  // find the max and min elements in the array
  let maxEle = Math.max(...arr);
  let minEle = Math.min(...arr);
  // find the range between the max and min elements
  let range = (maxEle - minEle) / noOfBuckets;

  // create an array of empty arrays to store elements based on their bucket i
  let temp = [];
  for (let i = 0; i < noOfBuckets; i++) {
    temp.push([]);
  }

  // distribute the elements of the array into the appropriate bucket based on
  for (let i = 0; i < arr.length; i++) {
    let diff = (arr[i] - minEle) / range - Math.floor((arr[i] - minEle) / rang
    // check if the difference is 0, and the element is not the min element
    if (diff === 0 && arr[i] !== minEle) {
      let flr = Math.floor((arr[i] - minEle) / range);
      temp[flr - 1].push(arr[i]);
    } else {
      let flr = Math.floor((arr[i] - minEle) / range);
      temp[flr].push(arr[i]);
    }
  }

  // sort each non-empty bucket using the Array.sort method
  for (let i = 0; i < temp.length; i++) {
    if (temp[i].length !== 0) {
      temp[i].sort((a, b) => a - b);
    }
  }

  // combine the sorted elements from each non-empty bucket into a single arra
  let k = 0:
```

```
            k++;
          }
        }
      }

    return arr;
}


let arr = [9.8, 0.6, 10.1, 1.9, 3.07, 3.04, 5.0, 8.0, 4.8, 7.68];
let noOfBuckets = 5;

// call the bucket sort method
console.log("Sorted array:", bucketSort(arr, noOfBuckets));
//This Code is Contributed by chinmaya121221
```

Output

```
Sorted array:  [0.6, 1.9, 3.04, 3.07, 4.8, 5.0, 7.68, 8.0, 9.8, 10.1]
```

Time Complexity:

The time complexity of bucket sort is **O(n + k)**, where n is the number of elements and k is the number of buckets.

Auxiliary Space :

The Auxiliary Space of bucket sort is **O(n + k)**. This is because we need to create a new array of size k to store the buckets and another array of size n to store the sorted elements.

[Bucket Sort To Sort an Array with Negative Numbers]

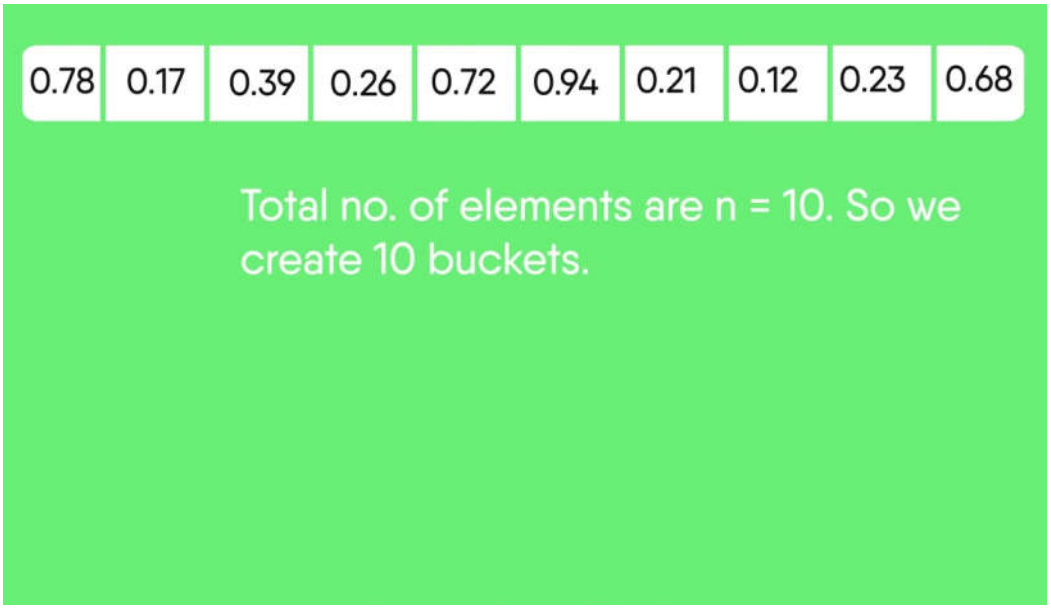
References:

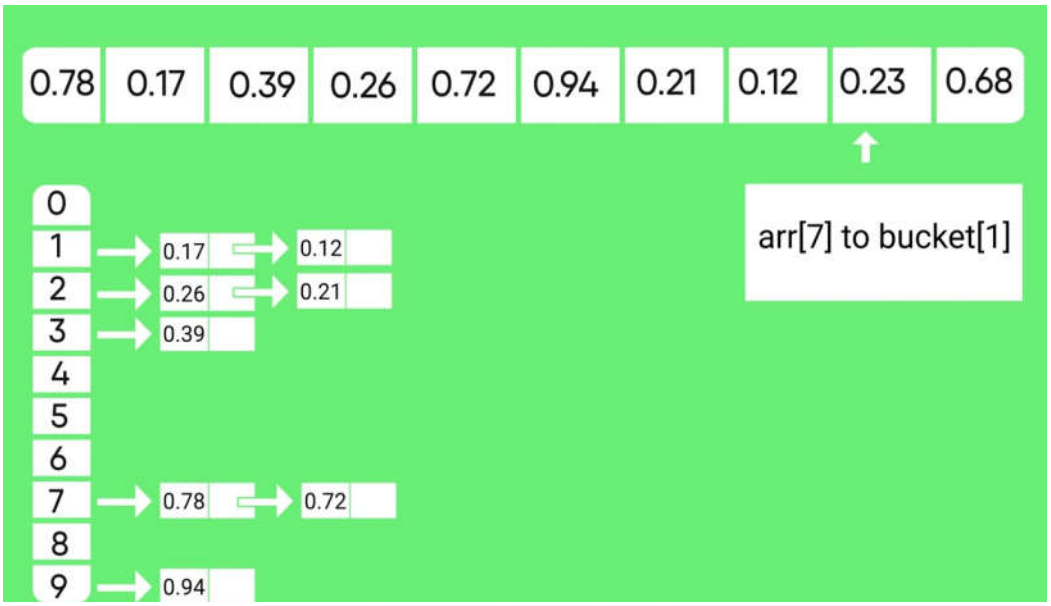
Introduction to Algorithms 3rd Edition by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest

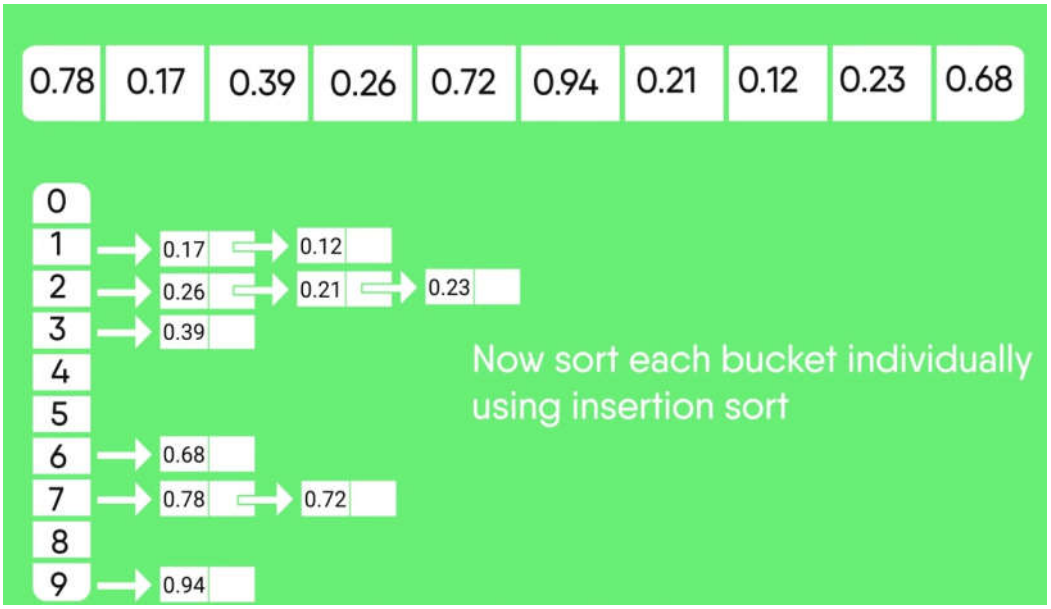
http://en.wikipedia.org/wiki/Bucket_sort

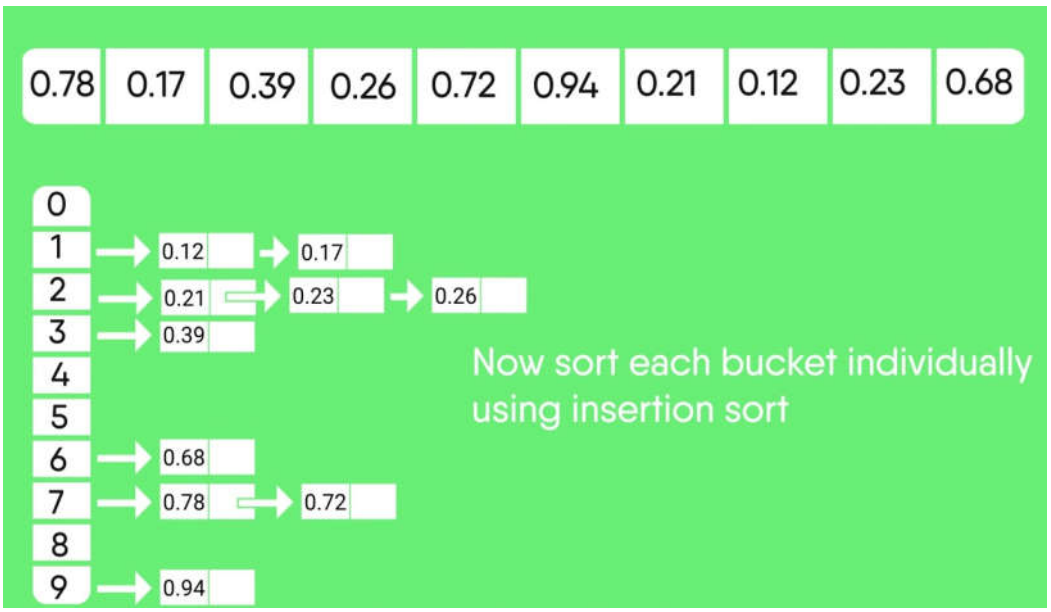
https://youtu.be/VuXbEb5ywrU

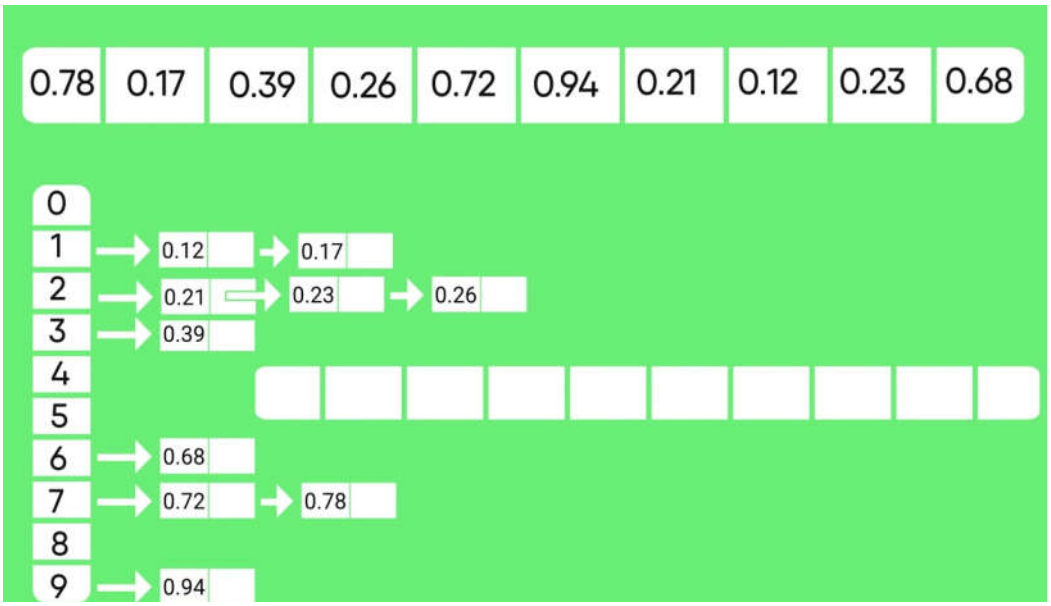
Snapshots:

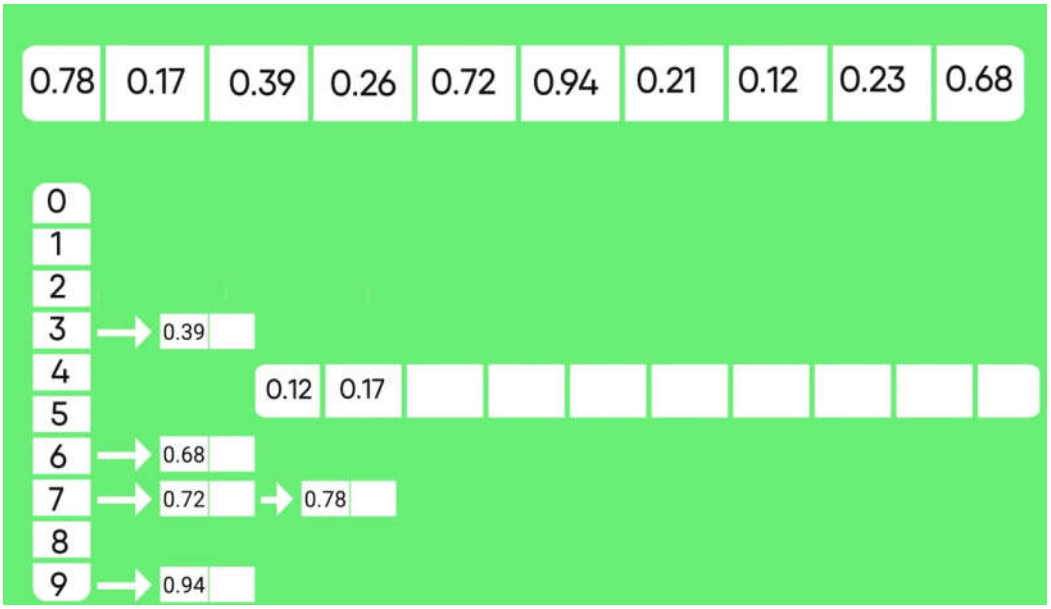












Quiz on Bucket Sort

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:

- Selection Sort
- Bubble Sort
- Insertion Sort
- Merge Sort
- Heap Sort
- QuickSort
- Radix Sort
- Counting Sort

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

Last Updated : 24 Mar, 2023 98

Similar Reads

1. Bucket Sort To Sort an Array with Negative Numbers
2. Radix Sort vs Bucket Sort
3. Bucket Sort Visualization Using Javascript
4. Minimum number of pigs required to find the poisonous bucket
5. Comparison among Bubble Sort, Selection Sort and Insertion Sort
6. C/C++ Program for Odd-Even Sort / Brick Sort
7. Java Program for Odd-Even Sort / Brick Sort
8. Insertion sort to sort even and odd positioned elements in different orders
9. Sort an Array which contain 1 to N values in O(N) using Cycle Sort
10. sort() vs. partial_sort() vs. nth_element() + sort() in C++ STL

Related Tutorials

1. Learn Data Structures with Javascript | DSA Tutorial
2. Introduction to Max-Heap – Data Structure and Algorithm Tutorials
3. Introduction to Set – Data Structure and Algorithm Tutorials

5. What is Dijkstra’s Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm

Previous Next

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : Akanksha_Rai, vinita07, OneilHsiao, himangshushekharjha, rutvik_56, architbubber, arorakashish0911, unknown2108, asmitwrites, divyansh2212, factworx4i2, chinmaya121221

Article Tags : DSA, Sorting

Practice Tags : Sorting

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

GoLang
SQL
R Language
Android Tutorial

Data Structures

Array
String
Linked List
Stack
Queue
Tree
Graph

Algorithms

Sorting
Searching
Greedy
Dynamic Programming
Pattern Searching
Recursion
Backtracking

Web Development

HTML
CSS
JavaScript
Bootstrap
ReactJS
AngularJS
NodeJS

Write & Earn

Write an Article
Improve an Article
Pick Topics to Write
Write Interview Experience
Internships
Video Internship

Computer Science

GATE CS Notes
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design

Data Science & ML

Data Science With Python
Data Science For Beginner
Machine Learning Tutorial
Maths For Machine Learning
Pandas Tutorial
NumPy Tutorial

Company Preparation

Preparation for SDE

Company Interview Corner

Experienced Interview

Internship Interview

Competitive Programming

Aptitude

Python Tutorial

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

UPSC/SSC/BANKING

SSC CGL Syllabus

SBI PO Syllabus

IBPS PO Syllabus

UPSC Ethics Notes

UPSC Economics Notes

UPSC History Notes