Bucket Sort

Last Updated : 30 Sep, 2025

Bucket sort is a sorting technique that involves dividing elements into various groups, or buckets. These buckets are formed by uniformly distributing the elements. Once the elements are divided into buckets, they can be sorted using any other sorting algorithm. Finally, the sorted elements are gathered together in an ordered fashion.

- Works well when the input array elements are uniformly distributed across a range.
- A stable algorithm because we use Insertion Sort (which is stable) to sort the individual buckets.

Bucket Sort Algorithm:

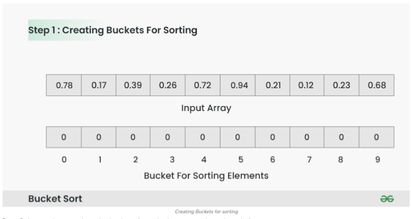
Create **n** empty buckets (Or lists) and do the following for every array element arr[i].

- Insert arr[i] into bucket[n*array[i]]
- Sort individual buckets using insertion sort.
- Concatenate all sorted buckets.

Step by Step Illustration

To apply bucket sort on the input array **{0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21, 0.12, 0.23, 0.68}**, we follow these steps:

Step 1: Create an array of size 10, where each slot represents a bucket.

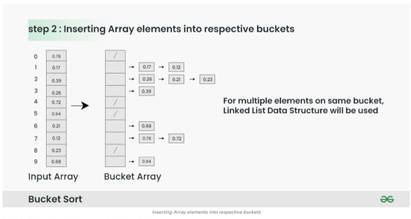


Creating Buckets for sorting

Step 2: Insert elements into the buckets from the input array based on their range.

Inserting elements into the buckets:

- Multiply each element by the size of the bucket array (10 in this case). For example, for element 0.23, we get 0.23 * 10 = 2.3.
- Convert the result to an integer, which gives us the bucket index. In this case, 2.3 is converted to the integer 2.
- Insert the element into the bucket corresponding to the calculated index.
- Repeat these steps for all elements in the input array.

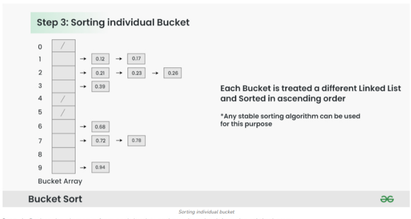


Inserting Array elements into respective buckets

Step 3: Sort the elements within each bucket.

Sorting the elements within each bucket:

- Apply a stable sorting algorithm (e.g., Insertion Sort) to sort the elements within each bucket.
- The elements within each bucket are now sorted.

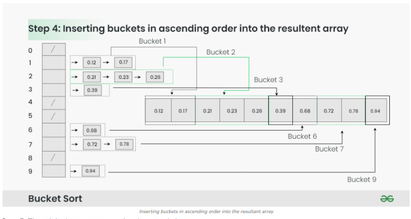


Sorting individual bucket

Step 4: Gather the elements from each bucket and put them back into the original array.

Gathering elements from each bucket:

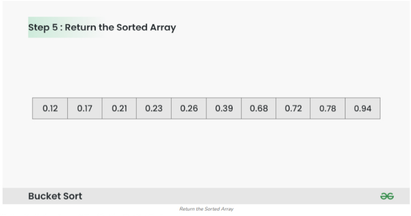
- Iterate through each bucket in order.
- Insert each individual element from the bucket into the original array.



Inserting buckets in ascending order into the resultant array

Step 5: The original array now contains the sorted elements.

The final sorted array using bucket sort for the given input is [0.12, 0.17, 0.21, 0.23, 0.26, 0.39, 0.68, 0.72, 0.78, 0.94].



Return the Sorted Array

Below is the implementation for the Bucket Sort:

C++ Java Python C# JavaScript

```python
def insertion_sort(bucket):
    for i in range(1, len(bucket)):
        key = bucket[i]
        j = i - 1
        while j >= 0 and bucket[j] > key:
            bucket[j + 1] = bucket[j]
            j -= 1
        bucket[j + 1] = key

def bucket_sort(arr):
    n = len(arr)
    buckets = [[] for _ in range(n)]

    # Put array elements in different buckets
    for num in arr:
        bi = int(n * num)
        buckets[bi].append(num)

    # Sort individual buckets using insertion sort
    for bucket in buckets:
        insertion_sort(bucket)

    # Concatenate all buckets into arr[]
    index = 0
    for bucket in buckets:
        for num in bucket:
            arr[index] = num
            index += 1

arr = [0.897, 0.565, 0.656, 0.1234, 0.665, 0.3434]
bucket_sort(arr)
print("Sorted array is:")
print(" ".join(map(str, arr)))
```

Output

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

Complexity Analysis of Bucket Sort Algorithm:

Worst Case Time Complexity: $O(n^2)$ The worst case happens when one bucket gets all the elements. In this case, we will be running insertion sort on all items which will make the time complexity as $O(n^2)$. We can reduce the worst case time complexity to O(n Log n) by using a O(n Log n) algorithm like Merge Sort or Heap Sort to sort the individual buckets, but that will improve the algorithm time for cases when buckets have small number of items as insertion sort works better for small arrays.

Best Case Time Complexity : O(n + k) The best case happens when every bucket gets equal number of elements. In this case every call to insertion sort will take constant time as the number of items in every bucket would be constant (Assuming that k is linearly proportional to n).

Auxiliary Space: O(n+k)

Article Tags : DSA BucketSort

Explore

DSA Fundamentals
Data Structures
Algorithms
Advanced
Interview Preparation
Practice Problem