Bucket Sort Algorithm

In this article, we will discuss the bucket sort Algorithm. The data items in the bucket sort are distributed in the form of buckets. In coding or technical interviews for software engineers, sorting algorithms are widely asked. So, it is important to discuss the topic.

Bucket sort is a sorting algorithm that separate the elements into multiple groups said to be buckets. Elements in bucket sort are first uniformly divided into groups called buckets, and then they are sorted by any other sorting algorithm. After that, elements are gathered in a sorted manner.

The basic procedure of performing the bucket sort is given as follows -

- First, partition the range into a fixed number of buckets.
- Then, toss every element into its appropriate bucket.
- After that, sort each bucket individually by applying a sorting algorithm.
- And at last, concatenate all the sorted buckets.

The advantages of bucket sort are -

- Bucket sort reduces the no. of comparisons.
- It is asymptotically fast because of the uniform distribution of elements.

The limitations of bucket sort are -

- It may or may not be a stable sorting algorithm.
- It is not useful if we have a large array because it increases the cost.
- It is not an in-place sorting algorithm, because some extra space is required to sort the buckets.

The best and average-case complexity of bucket sort is **O(n + k)**, and the worst-case complexity of bucket sort is **$O(n^2)$**, where **n** is the number of items.

Bucket sort is commonly used -

- With floating-point values.
- When input is distributed uniformly over a range.

The basic idea to perform the bucket sort is given as follows -

```
bucketSort(a[], n)
1. Create 'n' empty buckets
2. Do for each array element a[i]
    2.1. Put array elements into buckets, i.e. insert a[i] into bucket[n*a[i]]
3. Sort the elements of individual buckets by using the insertion sort.
4. At last, gather or concatenate the sorted buckets.
End bucketSort
```

Now, let's see the algorithm of bucket sort.

Algorithm

```
Bucket Sort(A[])
1.  Let B[0....n-1] be a new array
2.  n=length[A]
3.  for i=0 to n-1
4.  make B[i] an empty list
5.  for i=1 to n
6.  do insert A[i] into list B[n a[i]]
7.  for i=0 to n-1
8.  do sort list B[i] with insertion-sort
9.  Concatenate lists B[0], B[1],....., B[n-1] together in order
End
```

Scatter-gather approach

We can understand the Bucket sort algorithm via scatter-gather approach. Here, the given elements are first scattered into buckets. After scattering, elements in each bucket are sorted using a stable sorting algorithm. At last, the sorted elements will be gathered in order.

Let's take an unsorted array to understand the process of bucket sort. It will be easier to understand the bucket sort via an example.

Let the elements of array are -

Now, create buckets with a range from 0 to 25. The buckets range are 0-5, 5-10, 10-15, 15-20, 20-25. Elements are inserted in the buckets according to the bucket range. Suppose the value of an item is 16, so it will be inserted in the bucket with the range 15-20. Similarly, every item of the array will insert accordingly.

This phase is known to be the **scattering of array elements**.

Now, **sort** each bucket individually. The elements of each bucket can be sorted by using any of the stable sorting algorithms.

At last, **gather** the sorted elements from each bucket in order

Now, the array is completely sorted.

Bucket sort complexity

Now, let's see the time complexity of bucket sort in best case, average case, and in worst case. We will also see the space complexity of the bucket sort.

1. Time Complexity

Case	Time	Complexity
Best Case	O(n + k)	
Average Case	O(n + k)	
Worst Case	$O(n^2)$	

- **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. In Bucket sort, best case occurs when the elements are uniformly distributed in the buckets. The complexity will be better if the elements are already sorted in the buckets.
  If we use the insertion sort to sort the bucket elements, the overall complexity will be linear, i.e., O(n + k), where O(n) is for making the buckets, and O(k) is for sorting the bucket elements using algorithms with linear time complexity at best case.
  The best-case time complexity of bucket sort is **O(n + k)**.
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. Bucket sort runs in the linear time, even when the elements are uniformly distributed. The average case time complexity of bucket sort is **O(n + K)**.
- **Worst Case Complexity -** In bucket sort, worst case occurs when the elements are of the close range in the array, because of that, they have to be placed in the same bucket. So, some buckets have more number of elements than others.

The complexity will get worse when the elements are in the reverse order.

The worst-case time complexity of bucket sort is **O(n^2)**.

2. Space Complexity

Space Complexity	O(n*k)
Stable	YES

- The space complexity of bucket sort is O(n*k).

Implementation of bucket sort

Now, let's see the programs of bucket sort in different programming languages.

Program: Write a program to implement bucket sort in C language.

```
#include <stdio.h>
int getMax(int a[], int n) // function to get maximum element from the given array
{
  int max = a[0];
  for (int i = 1; i < n; i++)
    if (a[i] > max)
      max = a[i];
  return max;
}
void bucket(int a[], int n) // function to implement bucket sort
{
  int max = getMax(a, n); //max is the maximum element of array
  int bucket[max], i;
  for (int i = 0; i <= max; i++)
  {
    bucket[i] = 0;
```

```
  }
  for (int i = 0; i < n; i++)
  {
    bucket[a[i]]++;
  }
  for (int i = 0, j = 0; i <= max; i++)
  {
    while (bucket[i] > 0)
    {
      a[j++] = i;
      bucket[i]--;
    }
  }
}
void printArr(int a[], int n) // function to print array elements
{
  for (int i = 0; i < n; ++i)
    printf("%d  ", a[i]);
}
int main()
{
  int a[] = {54, 12, 84, 57, 69, 41, 9, 5};
  int n = sizeof(a) / sizeof(a[0]); // n is the size of array
  printf("Before sorting array elements are - \n");
  printArr(a, n);
  bucket(a, n);
  printf("\nAfter sorting array elements are - \n");
  printArr(a, n);
}
```

Output

After the execution of above code, the output will be -

```
Before sorting array elements are -
54  12  84  57  69  41  9  5
After sorting array elements are -
5  9  12  41  54  57  69  84
```

Program: Write a program to implement bucket sort in C++.

```
#include <iostream>

using namespace std;

int getMax(int a[], int n) // function to get maximum element from the given array
{
  int max = a[0];
  for (int i = 1; i < n; i++)
    if (a[i] > max)
      max = a[i];
  return max;
}
void bucket(int a[], int n) // function to implement bucket sort
{
  int max = getMax(a, n); //max is the maximum element of array
  int bucket[max], i;
  for (int i = 0; i <= max; i++)
  {
    bucket[i] = 0;
  }
  for (int i = 0; i < n; i++)
  {
    bucket[a[i]]++;
  }
```

```
  for (int i = 0, j = 0; i <= max; i++)
  {
    while (bucket[i] > 0)
    {
      a[j++] = i;
      bucket[i]--;
    }
  }
}
void printArr(int a[], int n) // function to print array elements
{
  for (int i = 0; i < n; ++i)
    cout<<a[i]<<" ";
}
int main()
{
  int a[] = {34, 42, 74, 57, 99, 84, 9, 5};
  int n = sizeof(a) / sizeof(a[0]); // n is the size of array
  cout<<"Before sorting array elements are - \n";
  printArr(a, n);
  bucket(a, n);
  cout<<"\nAfter sorting array elements are - \n";
  printArr(a, n);
}
```

Output

After the execution of above code, the output will be -

```
Before sorting array elements are -
34 42 74 57 99 84 9 5
After sorting array elements are -
5 9 34 42 57 74 84 99
```

Program: Write a program to implement bucket sort in C#.

```
using System;
class Bucket {
static int getMax(int[] a) // function to get maximum element from the given array
{
    int n = a.Length;
    int max = a[0];
    for (int i = 1; i < n; i++)
    if (a[i] > max)
    max = a[i];
    return max;
}
static void bucket(int[] a) // function to implement bucket sort
{
    int n = a.Length;
    int max = getMax(a); //max is the maximum element of array
    int[] bucket = new int[max+1];
    for (int i = 0; i <= max; i++)
    {
        bucket[i] = 0;
    }
    for (int i = 0; i < n; i++)
    {
        bucket[a[i]]++;

    }
    for (int i = 0, j = 0; i <= max; i++)
    {
        while (bucket[i] > 0)
        {
```

```
            a[j++] = i;
            bucket[i]--;
        }
    }
}
static void printArr(int[] a) /* function to print the array */
{
        int i;
    int n = a.Length;
    for (i = 0; i < n; i++)
    Console.Write(a[i] + " ");
}
static void Main() {
    int[] a = { 95, 50, 45, 15, 20, 10 };
    Console.Write("Before sorting array elements are - \n");
    printArr(a);
    bucket(a);
    Console.Write("\nAfter sorting array elements are - \n");
    printArr(a);
  }
}
```

Output

After the execution of above code, the output will be -

```
Before sorting array elements are -
95 50 45 15 20 10
After sorting array elements are -
10 15 20 45 50 95
```

Program: Write a program to implement bucket sort in Java.