Bucket sort is mainly useful when input is uniformly distributed over a range. The lower bound for Comparison-based sorting algorithm is $\Omega(nLog_2n)$. How can we improve such time complexity?

procedure bucketSort(arr[], n):

1. create n empty buckets (or lists).
2. do following for every array element arr[i]:

   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.

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

    // 2) Put array elements in different buckets
    for (int i=0; i<n; i++)
    {
       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 funtion */
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;
}
```