KOTEBE METROPOLITAN UNIVERSITY

***ALGORITHM ANALYSIS***

**CoSc 2092**

Group Assignment

***Members name Id number***

***1.Gacheno Mamo…………………CNCS/UR15140/12***

***2.Biruk binbesa………………………CNCS/UR15024/12***

***3.Fasil Israel……………………………CNCS/UR15128/12***

***4.surafel Gashawtena…………….026/08***

***5.sintayehu kassa……………………027/08***

x

Submission date :dec 5,2021

Submitted to: Mr. Mulugeta M

## *BUCKET SORT ALGORITHM*

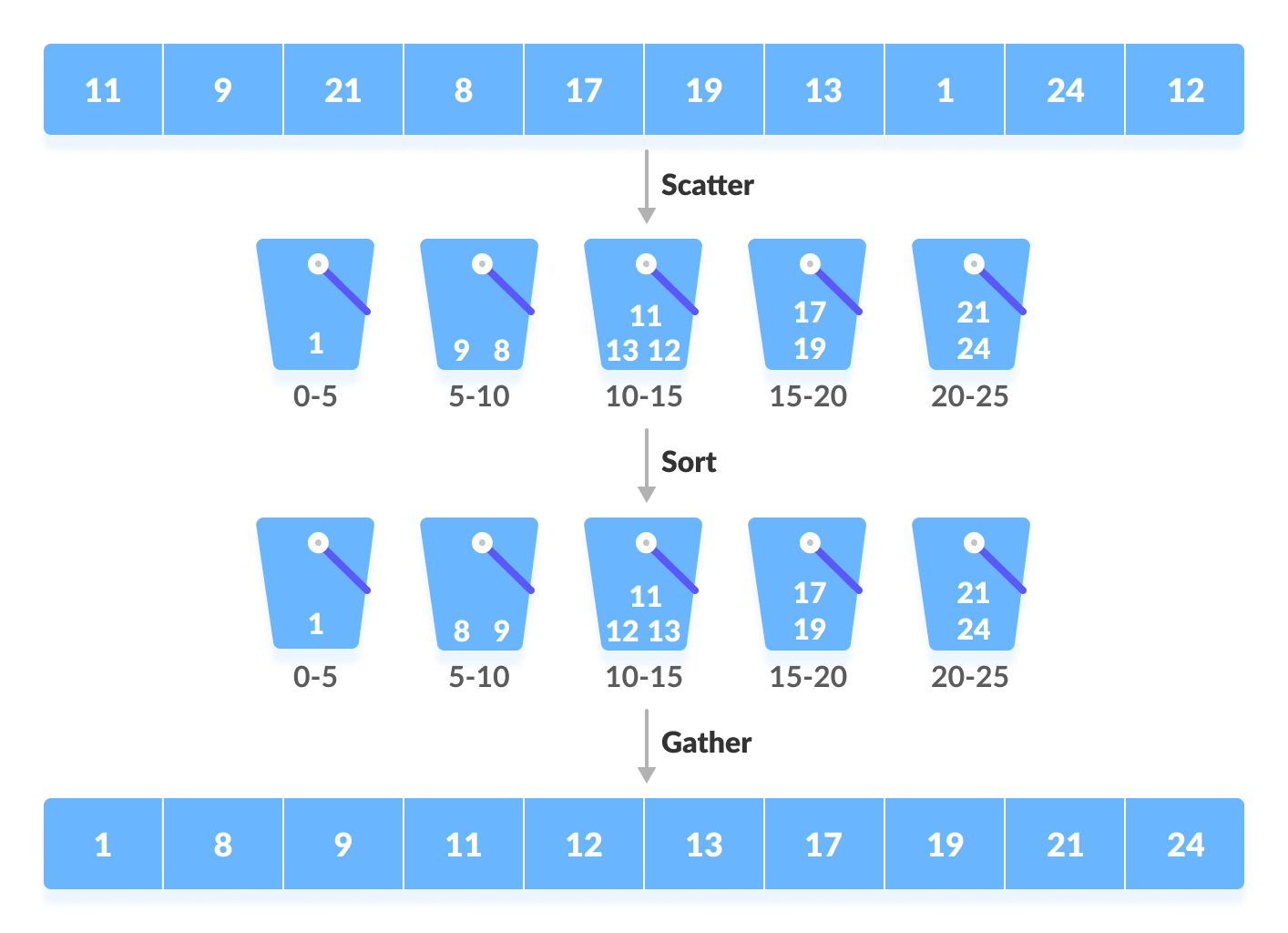
Bucket sort

**What is bucket sort algorithm?**

* Bucket Sort is a sorting algorithm that divides the unsorted array elements into several groups called buckets.
* Each bucket is then sorted by using any of the suitable [sorting algorithms](https://www.programiz.com/dsa/sorting-algorithm) or recursively applying the same bucket algorithm.
* Finally, the sorted buckets are combined to form a final sorted array.

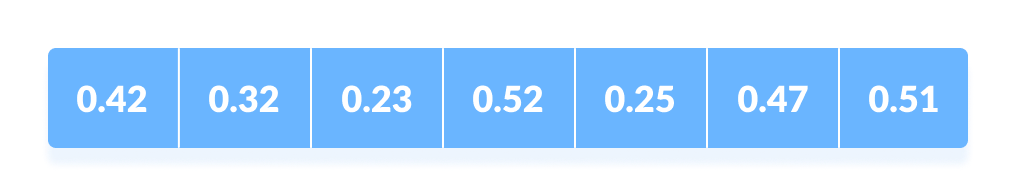
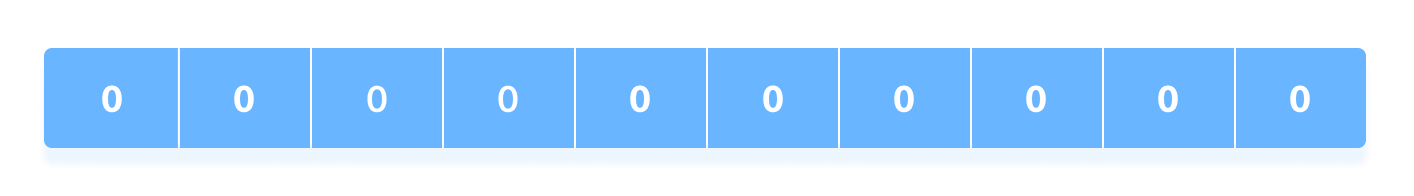
**Working of bucket sort algorithm**

* The process of bucket sort can be understood as a **scatter-gather approach**.
* Here, elements are first scattered into buckets then the elements in each bucket are sorted. Finally, the elements are gathered in order.

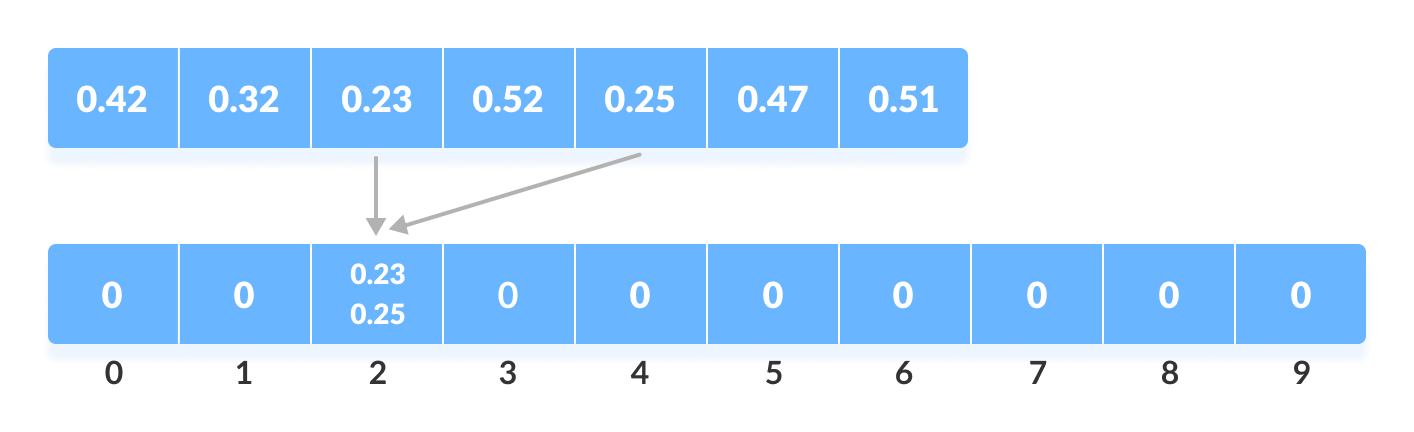


**Working of Bucket Sort**

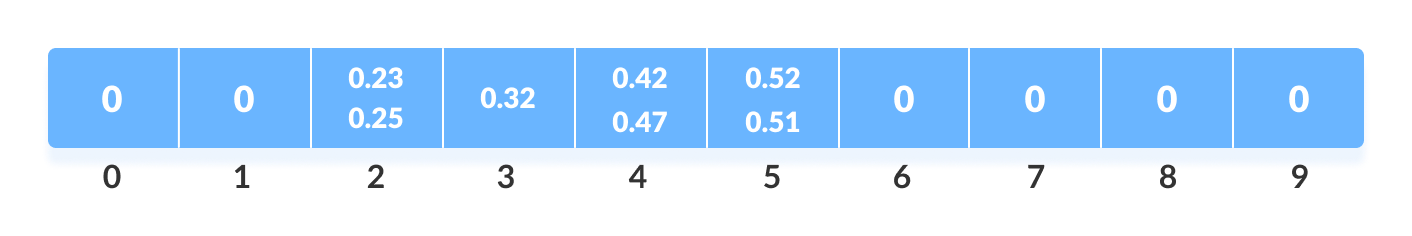
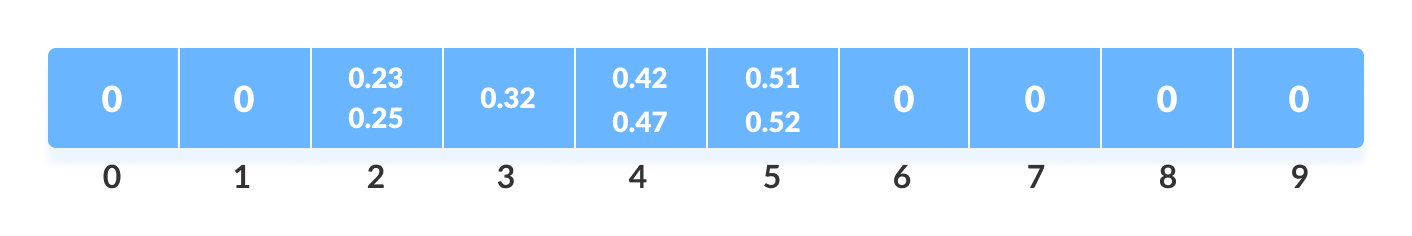
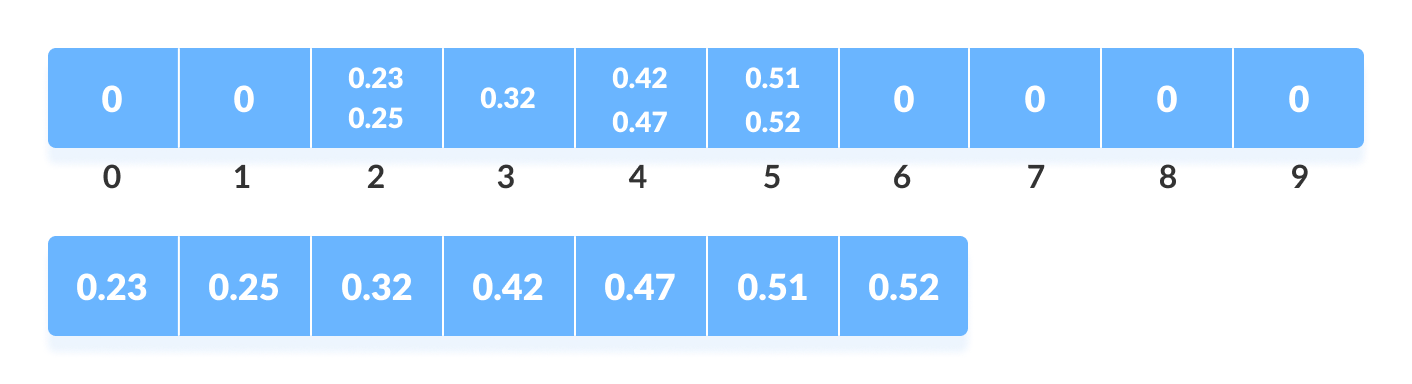
**What ever the number of elements are more than 10 we will take 10 buckets to sort the elements.**

* Suppose, the input array is: input array  
  Create an array of size 10. Each slot of this array is used as a bucket for storing elements.

Array in which each position is a bucket

* Insert elements into the buckets from the array.
* The elements are inserted according to the range of the bucket.  
    
  In our example code, we have buckets each of ranges from 0 to 1, 1 to 2, 2 to 3,...... (n-1) to n.
* Suppose, an input element is .23 is taken. It is multiplied by size = 10 (ie. .23\*10=2.3). Then, it is converted into an integer (ie. 2.3≈2). Finally, .23 is inserted into **bucket-2**.

Insert elements into the buckets from the array

* Similarly, .25 is also inserted into the same bucket. Everytime, the floor value of the floating point number is taken.  
    
  **If we take integer numbers as input, we have to divide it by the interval (10 here) to get the floor value.**  
    
  Similarly, other elements are inserted into their respective buckets.Insert all the elements into the buckets from the array
* The elements of each bucket are sorted using any of the stable sorting algorithms. Here, we have used quicksort (inbuilt function).Sort the elements in each bucket
* The elements from each bucket are gathered.  
    
  It is done by iterating through the bucket and inserting an individual element into the original array in each cycle. The element from the bucket is erased once it is copied into the original array.Gather elements from each bucket

**Pseudocode of Bucket Sort Algorithm**

bucketSort()

create N buckets each of which can hold a range of values

for all the buckets

initialize each bucket with 0 values

for all the buckets

put elements into buckets matching the range

for all the buckets

sort elements in each bucket

gather elements from each bucket

end bucketSort

**#include<iostream>**

**#include<vector>**

**#include<algorithm>**

**using namespace std;**

**void display(float \*array, int size) {**

**for(int i = 0; i<size; i++)**

**cout << array[i] << " ";**

**cout << endl;**

**}**

**void bucketSort(float \*array, int size) {**

**vector<float> bucket[size];**

**for(int i = 0; i<size; i++) { //put elements into different buckets**

**bucket[int(size\*array[i])].push\_back(array[i]);**

**}**

**for(int i = 0; i<size; i++) {**

**sort(bucket[i].begin(), bucket[i].end()); //sort individual vectors**

**}**

**int index = 0;**

**for(int i = 0; i<size; i++) {**

**while(!bucket[i].empty()) {**

**array[index++] = \*(bucket[i].begin());**

**bucket[i].erase(bucket[i].begin());**

**}**

**}**

**}**

**int main() {**

**int n;**

**cout << "Enter the number of elements: ";**

**cin >> n;**

**float arr[n]; //create an array with given number of elements**

**cout << "Enter elements:" << endl;**

**for(int i = 0; i<n; i++) {**

**cin >> arr[i];**

**}**

**cout << "Array before Sorting: ";**

**display(arr, n);**

**bucketSort(arr, n);**

**cout << "Array after Sorting: ";**

**display(arr, n);**

**}**

**Bucket Sort Code in/C++**

**Bucket Sort Complexity**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Time Complexity** |  |  |
|  |  |  |  |
|  |  |  |
|  | Best |  | O(n+k) |
|  |  |  |  |
|  | Worst |  | O(n2) |
|  |  |  |  |
|  | Average |  | O(n) |
|  | **Space Complexity** |  | O(n+k) |
|  |  |  |  |
|  | **Stability** |  | Yes |

|  |
| --- |
|  |

**Worst Case Complexity:** O(n2)

* When there are elements of close range in the array, they are likely to be placed in the same bucket.
* This may result in some buckets having more number of elements than others.
* It makes the complexity depend on the sorting algorithm used to sort the elements of the bucket.
* The complexity becomes even worse when the elements are in reverse order. If insertion sort is used to sort elements of the bucket, then the time complexity becomes O(n2).
* **Best Case Complexity:** O(n+k)
* It occurs when the elements are uniformly distributed in the buckets with a nearly equal number of elements in each bucket.  
  The complexity becomes even better if the elements inside the buckets are already sorted.
* complexity in the best case will be linear ie. O(n+k). O(n) is the complexity for making the buckets and O(k) is the complexity for sorting the elements of the bucket using algorithms having linear time complexity at the best case.
* **Average Case Complexity:** O(n)  
  It occurs when the elements are distributed randomly in the array. Even if the elements are not distributed uniformly, bucket sort runs in linear time. It holds true until the sum of the squares of the bucket sizes is linear in the total number of elements.

**Bucket Sort Applications**

Bucket sort is used when:

* input is uniformly distributed over a range.
* there are floating point values