

**Data Structure**

**Project#4**

**Name : Sondos sarraj**

**ID : 1140275**

* Bucket sort :

# Bucket sort, is a [sorting algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm) that works by distributing the elements of an [array](https://en.wikipedia.org/wiki/Array_data_structure) into a number of [buckets](https://en.wikipedia.org/wiki/Bucket_%28computing%29). Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the bucket sorting algorithm. It is a [distribution sort](https://en.wikipedia.org/wiki/Distribution_sort) .

# Another advantage of bucket sort is that you can use it as an external sorting algorithm. If you need to sort a list that is so huge you can't fit it into memory, you can stream the list through RAM, distribute the items into buckets stored in external files, then sort each file in RAM independently. It is fast and stable .

**Bucket Sort algorithm:**

1-in sorting algorithm we create buckets and put elements in to them .

divider=ceil((max+1)/bucket);

b[j]=arr[i];

j=floor(arr[i]/divider);

## 2- then we apply some sorting algorithm (insertion sort )to sort the element in each bucket .

# 3-finally we take the elements out and join them to get sorted result .

**Time and Space Complexity:**

**K = number of buckets .**



## Counting sort

## [Counting sort](http://en.wikipedia.org/wiki/Counting_sort) is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.

## Counting Sort algorithm:

## 1-find the min and max value.

# 2- create index from min to max .

# 3-create an array that will hold count of each number count .

# 4-count the number of times each appeared in the out put .

## 5-we have counted all the numbers in the in put we can fill the empty space in count array with 0 .

# 6-Now create sum count array that will hold sum of counts for given index.

# 7-count the numbers in input and create a position and fill those position with given number per sum count .

**Time and Space Complexity:**

##

## Comb Sort :

 Comb Sort is mainly an improvement over Bubble Sort. Bubble sort always compares adjacent values. So all [inversions](http://www.geeksforgeeks.org/counting-inversions/) are removed one by one. Comb Sort improves on Bubble Sort by using gap of size more than 1. The gap starts with a large value and shrinks by a factor of 1.3 in every iteration until it reaches the value 1. Thus Comb Sort removes more than one [inversion counts](http://www.geeksforgeeks.org/counting-inversions/) with one swap and performs better than Bubble Sort.

The shrink factor has been empirically found to be 1.3 (by testing Combsort on over 200,000 random lists)

Although, it works better than Bubble Sort on average, worst case remains O(n2)

## Comb Sort algorithm:

1. Calculation of the *gap* value.
2. Iterating over the *data set* comparing each item with the item that is *“gap”* elements further down the list and swapping them if required.
3. Checking to see if the gap value has reached one and no swaps have occurred. If so, then the set has been sorted.

**Time and Space Complexity :**

Worst case complexity of this algorithm is O(n2) and the Best Case complexity is O(n log n).

**Auxiliary Space :**O(1).

* **Cocktail Sort :**

is a variation of [bubble sort](https://en.wikipedia.org/wiki/Bubble_sort) that is both a [stable](https://en.wikipedia.org/wiki/Stable_sort)[sorting algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm) and a [comparison sort](https://en.wikipedia.org/wiki/Comparison_sort). The algorithm differs from a [bubble sort](https://en.wikipedia.org/wiki/Bubble_sort) in that it sorts in both directions on each pass through the list. This sorting algorithm is only marginally more difficult to implement than a bubble sort, and solves the problem of [turtles](https://en.wikipedia.org/wiki/Bubble_sort#Rabbits_and_turtles) in bubble sorts.

## Cocktail Sort algorithm:

# Compare first two numbers – swap if necessary – rewrite the remainder of the list.

# Compare second and third number – swap if necessary – rewrite the remainder of the list. Now compare the first two numbers –swap if necessary and re-write the list….IF not swap needed start the next pass.

# The pass which starts by comparing the second to last and last numbers. Make sure you complete the pass indicating comparisons made.

**Time and Space Complexity :**

worst case = O(N2) , best case = O(N) ,Avg case =O(N2).

Auxiliary Space : O(1). Just like bubble sort this is also an in-place algorithm.

* **Odd-Even Sort :**

Odd –even sort is a relatively simple [sorting algorithm](https://en.wikipedia.org/wiki/Sorting_algorithm), developed originally for use on parallel processors with local interconnections. It is a [comparison sort](https://en.wikipedia.org/wiki/Comparison_sort) related to [bubble sort](https://en.wikipedia.org/wiki/Bubble_sort), with which it shares many characteristics. It functions by comparing all odd/even indexed pairs of adjacent elements in the list and, if a pair is in the wrong order (the first is larger than the second) the elements are switched. The next step repeats this for even/odd indexed pairs (of adjacent elements). Then it alternates between odd/even and even/odd steps until the list is sorted.

**Time and Space Complexity :**

Worst case complexity of this algorithm is = O(n2)

 the Best Case complexity is O(n).

**Auxiliary Space :**O(1).

**Referances :**

**Stack over flow**

**Growing with the web**

# Rosetta Code

**Wikipedia**