Faculty of Engineering & Technology Electrical & Computer Engineering Department

COMP2421, DATA STRUCTURES

PROJECT 4

Name and ID :
1192423-Raghad Afaghani

Section : 2

Instructor : Dr.Radi Jarrar

Date : 23.2.2023

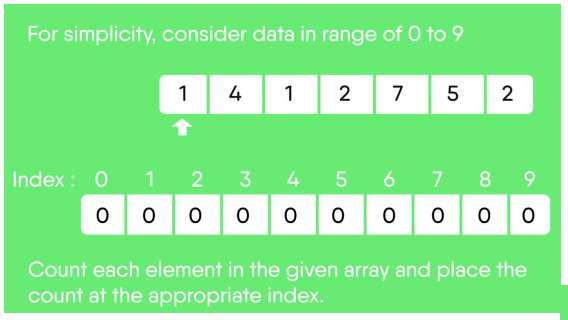
Table of content

Abstract

Sorting involves arranging items or data in a specific order, making it easier to find the desired data using a specific algorithm. In this report, I will be discussing five types of sorting algorithms written in the C programming language. The aim of sorting is to make it easier to search through a list of data in a more efficient way.

2.Theory
2.1 Counting Sort

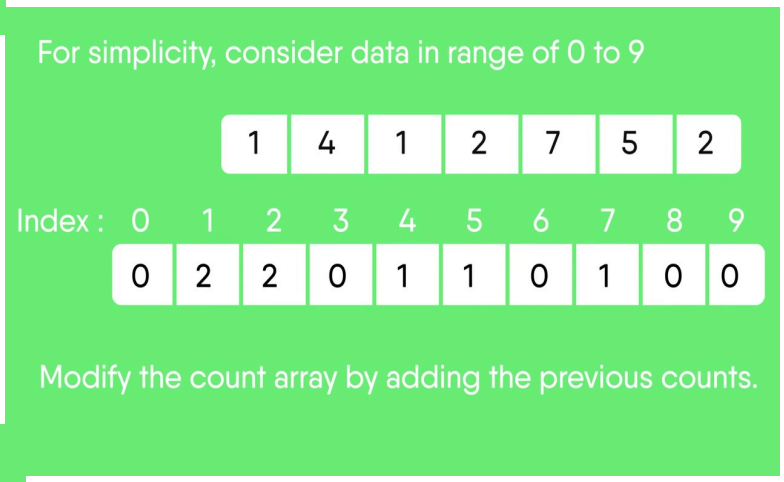
Counting sort is a way to sort things by their key value when the key values are within a certain range. It works by counting how many objects have each key value, and then using some math to figure out where each object should go in the final sorted list. It's kind of like putting things in different boxes based on their key, and then arranging the boxes in order to get the final sorted list.

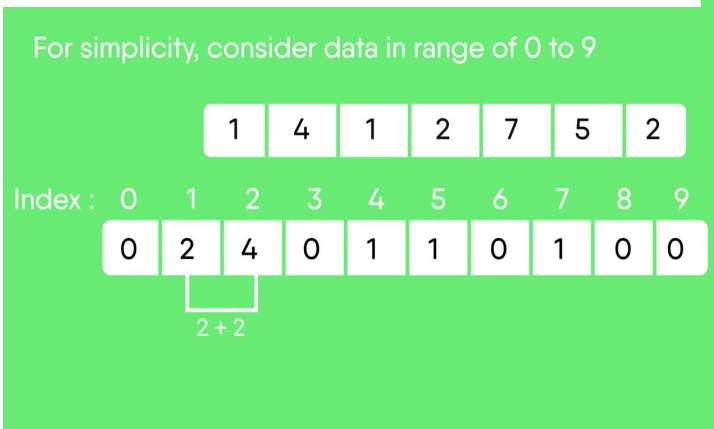


Step1:Make a count array to store the count of each unique object.

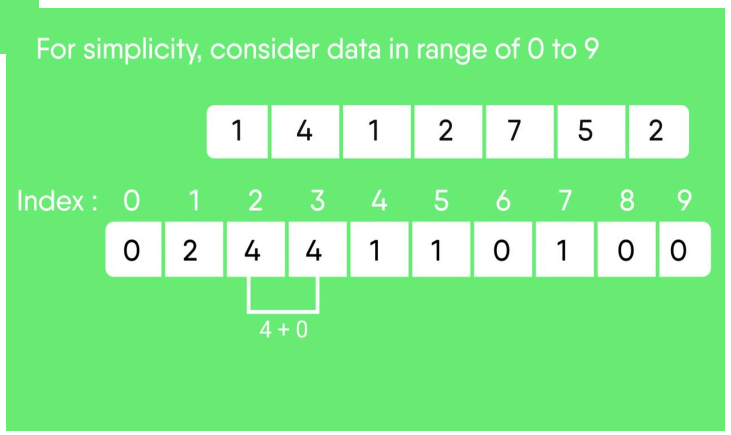
Step2:Now, store the count of each unique element in the count array

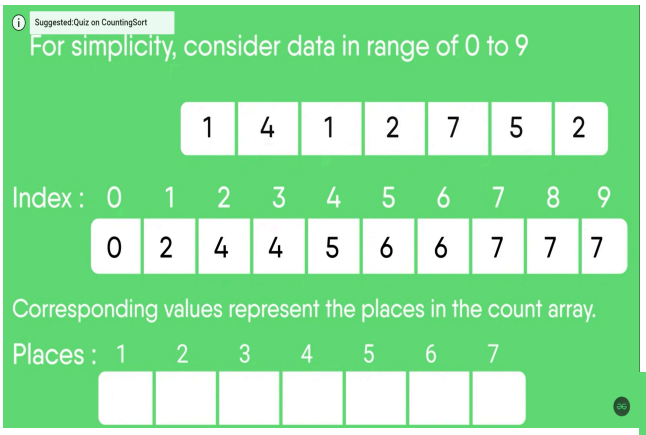
If any element repeats itself, simply increase its count.





Step3:Modify the count array by adding the previous counts.



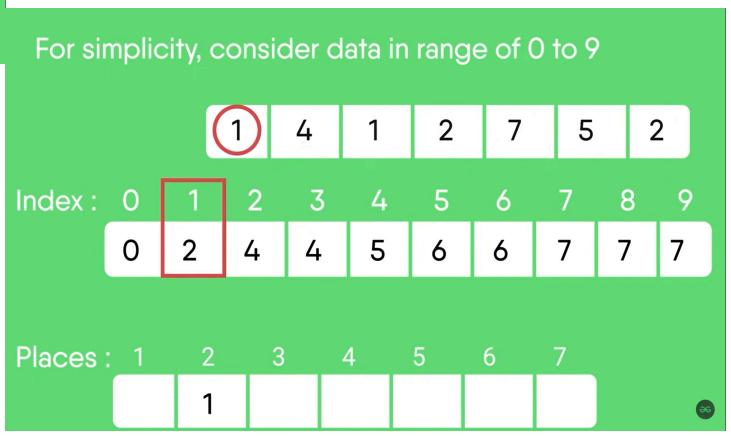


Step4:Create an empty array with the same size as the base array.

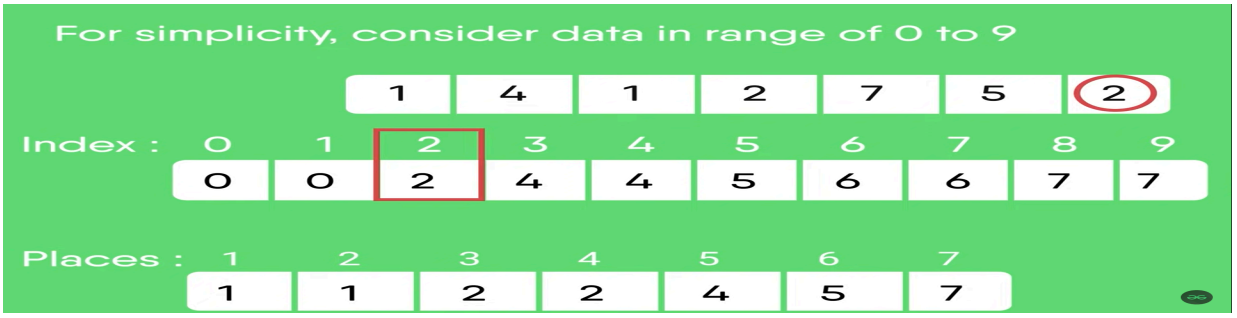
Since we have 7 inputs, We create an array with seven Places

Step5:We place the objects in their correct position and decrease the count by one

Note that:Corresponding values represent the places in the count array



Array is now sorted



Time Complexity

- ❖ Worst case time O(n)
- ❖ Best case time O(n)
- ❖ Average case time O(n)
- ❖ Space complexity O(n) .

Stability?

Counting sort is a stable sorting algorithm, which means that it preserves the relative order of equal elements. The stability of the algorithm depends on the range of values in the array, which is denoted by the variable k.

Running Time

Regardless of whether the array is sorted in ascending, descending, or unsorted order, counting sort has the same time complexity because it works with a consistent range of values (k) and number of elements (n). Therefore, the best, worst, and average cases have the same time complexity.

Comparison-based or non-comparison-based?

Non Comparison-based algorithm

In place or not?

out of space

- ★ Counting sort cannot sort arrays that contain negative numbers because there are no negative array indices.
- ★ In summary, counting sort is a fast and efficient algorithm that works best for small data sets with a limited range of values. It's especially useful when the data is already partially sorted or has a limited range. However, it's not suitable for large data sets, and it requires additional space for storing the sorted data.

2.2 Comb Sort

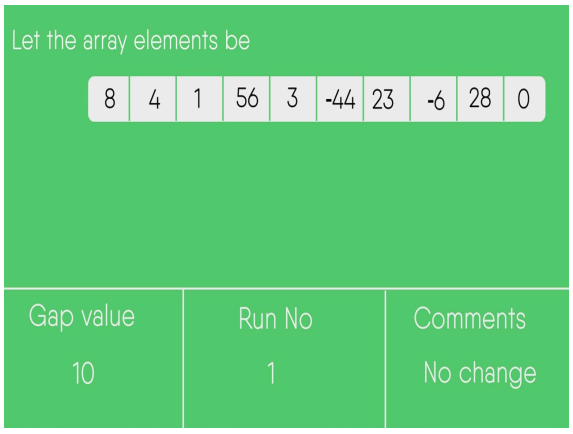
Comb Sort is a sorting algorithm that is similar to Bubble Sort. However, instead of comparing adjacent values, it uses a gap that starts with a large value and shrinks by a factor of 1.3 in each iteration until it reaches the value of 1. This allows Comb Sort to remove more than one inversion count with each swap and perform better than Bubble Sort. The shrink factor of 1.3 has been determined empirically by testing Combsort on over 200,000 random lists.

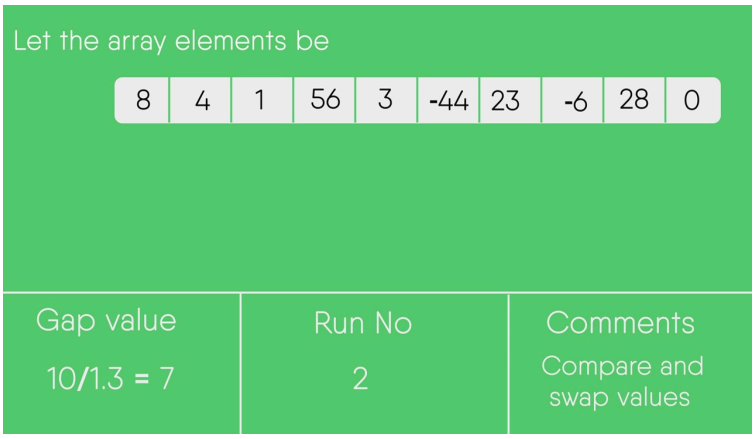
While Comb Sort works better than Bubble Sort on average, it still has a worst-case time complexity of O(n2), which means that it can take a long time to sort large datasets.

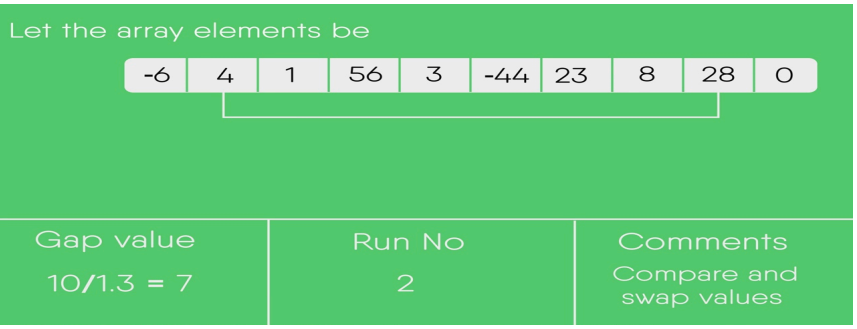
Step1:Start with a gap size of n, where n is the length of the list.

Initially gap value = 10 [number of the element in the array]

After shrinking gap value => 10/1.3 = **7**; [number of the element in the array/1.3]

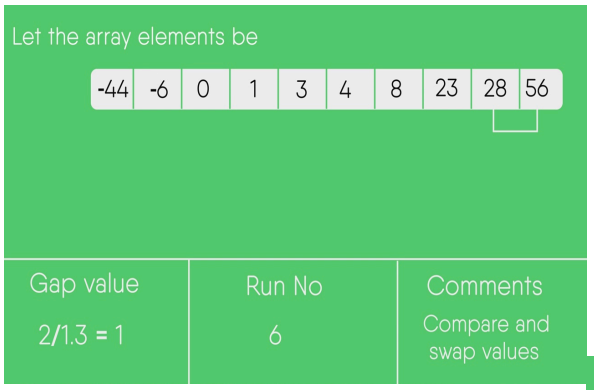




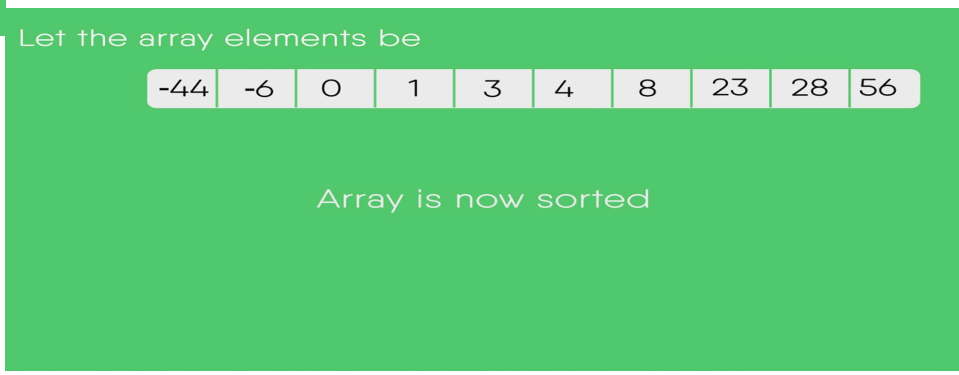


Step 2: Compare elements that are n positions apart and swap them if they are in the wrong order.

Step 3: Reduce the gap size by a shrink factor, typically 1.3 (this value was empirically determined to be effective).



Step 4:Repeat steps 2-3 until the gap size is less than or equal to 1.



Finally, perform a standard bubble sort on the list to ensure that any remaining out-of-order elements are sorted correctly.

Time Complexity

- Worst case time O(n^2)
- Best case time O(n log n)
- Average case time O(N^2/2^p)
- Space complexity O(1) .

Stability?

stability: it's not stable

Running Time

- When the array is already sorted in ascending order, the time complexity of comb sort is at its minimum, which is O(n log n).
- However, when the array is sorted in descending order, the time complexity becomes O(n^2) because the algorithm has to iterate through all the numbers in nested loops.
- For an unsorted array, the time complexity of comb sort is O(n^2/2^p), where p is the number of increments. However, on average, the algorithm breaks out of the while loop earlier than all numbers, making the average case time complexity better than worst-case but still quadratic.

Comparison-based or non-comparison-based?

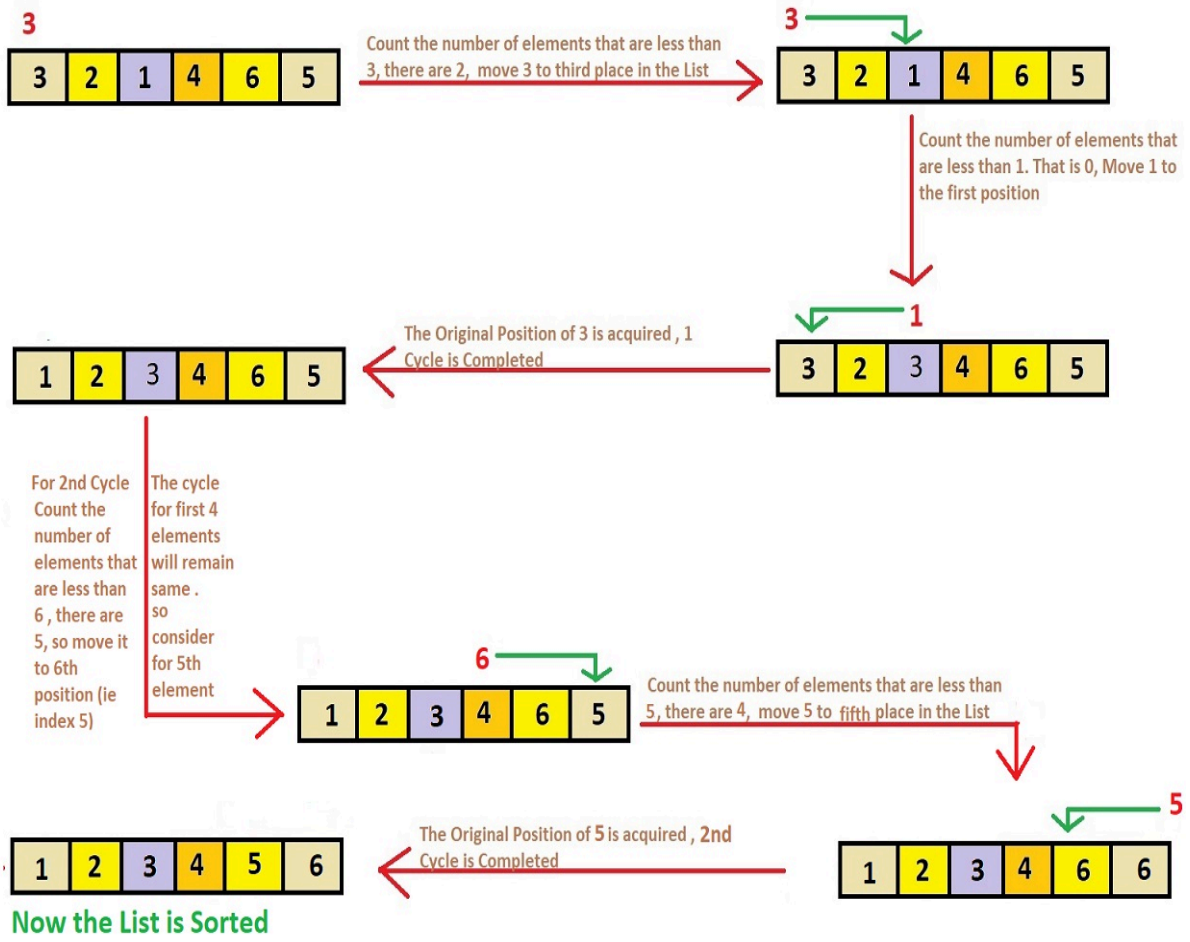
Comparison-based algorithm

In place or not?

in place

2.3 Cycle Sort

Cycle sort is a sorting algorithm that divides an array into cycles and rotates each cycle to produce a sorted array. It is an unstable, in-place sorting algorithm that minimizes the number of memory writes needed to sort the array. Each value is either written zero times if it's already in the correct position or written once to its correct position. The algorithm works by treating the array as a graph with nodes and edges, where each edge represents the correct position of an element. The algorithm then sorts the array by moving elements to their correct positions along the edges of the graph.



Working Principle

The cycle sort algorithm involves iterating through the list of distinct elements to sort them. For each element, we check if it is in its correct position. If it is, we move on to the next element. If it is not in the correct position, we must move it to its correct

position. We do this by moving the element that is currently in the correct position to its correct position, and so on, until each item is in its correct index. We repeat this cycle for each item in the list until the resulting list is fully sorted.

Time Complexity

- ❖ Worst case time O(n^2)
- ❖ Best case time O(n^2)
- ❖ Average case time O(n^2)
- ❖ Space complexity O(1) .

Running Time

- When the data is already sorted in ascending order and we want to sort it in ascending order, the best-case scenario is used with a time complexity of O(n^2), but this scenario is optimistic and not realistic as the program may stop running at a certain threshold.

- When the data is sorted in descending order and we want to sort it in ascending order, the worst-case scenario is used with a time complexity of O(n^2). The program will give the longest runtime for the algorithm given the size n and will complete eventually.

- When the data is not sorted, we use the average-case scenario with a time complexity of O(n^2) because the order is random.

Stability?

No,it’s not stable

Comparison-based or non-comparison-based?

Comparison-based algorithm

In place or not?

in place

2.4 Pigeonhole Sort

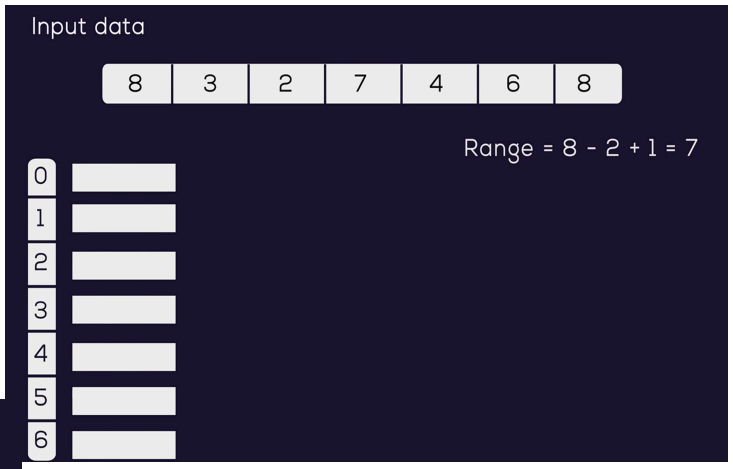
Pigeonhole sorting is a type of sorting algorithm used to sort lists of items with a range of possible values. It is suitable when the number of items and the range of values are similar. It takes O(n + N) time to sort the items, where n is the number of items and N is the length of the range of values.

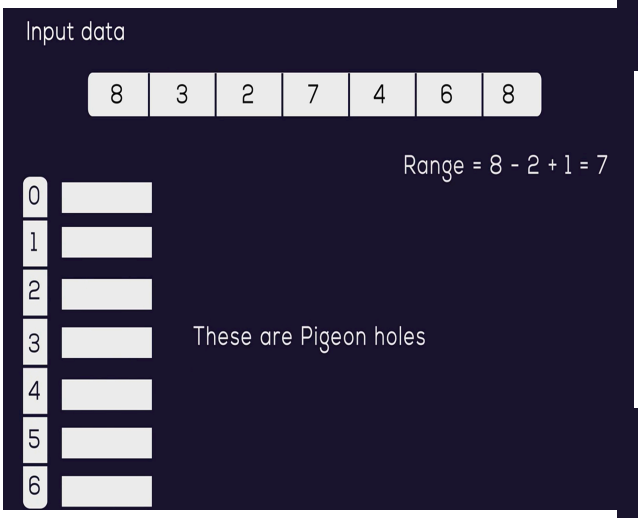
This algorithm is similar to counting sort, but with a small difference. Pigeonhole sort moves the items twice, first to a bucket array and then to their final destination, whereas counting sort builds an auxiliary array and uses it to compute each item's final destination and then moves the item there.

1.Find minimum and maximum values in the array.

Let the minimum and maximum values be ‘min’ and ‘max’ respectively.

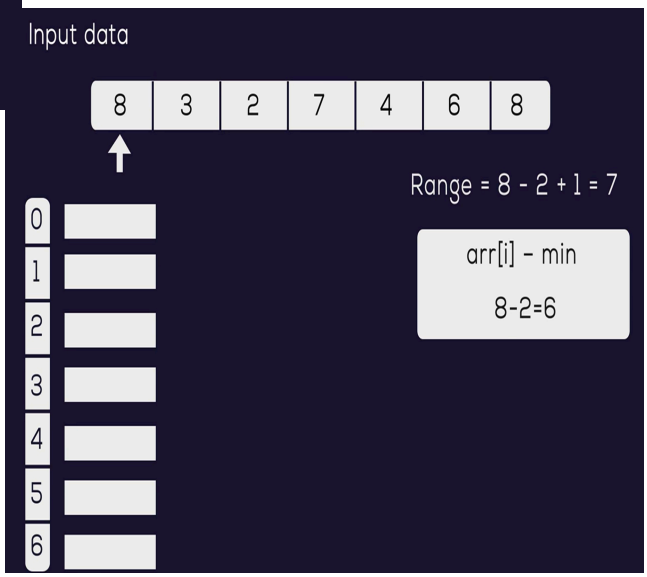
Also find the range as ‘max-min+1’.



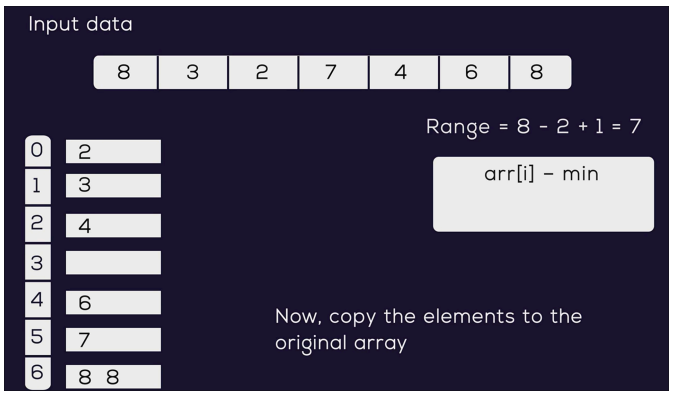


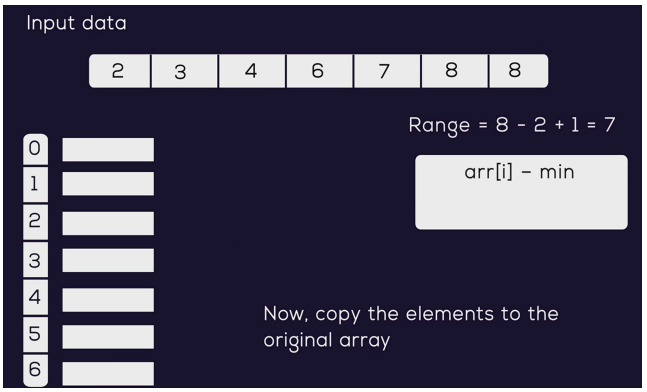
2.Set up an array of initially empty “pigeonholes” the same size as of the range.

3.Visit each element of the array and then put each element in its pigeonhole. An element arr[i] is put in hole at index arr[i] – min.



4.Now,Copy the elements to the original array





Time Complexity

- ❖ Worst case time O(n)
- ❖ Best case time O(n)
- ❖ Average case time O(n)
- ❖ Space complexity O(n) .

Running Time

- When the data is already sorted in ascending order and we want to sort it in the same order, the best-case scenario suggests that the algorithm will take O(n) time to complete. However, this is not a realistic scenario since the program may stop running after reaching a certain threshold.

- When the data is arranged in descending order and we want to sort it in ascending order, the worst-case scenario suggests that the algorithm will take O(n) time to complete. This is the longest possible runtime for the algorithm given the input size n.

- When the data is not sorted, we use the average-case scenario, which suggests that the algorithm will take O(n) time since the order of the input will be random.

Stability?

Stable sort algorithm.

Comparison-based or non-comparison-based?

Non Comparison-based algorithm

In place or not?

Out-place

2.5 Gnome Sort

Gnome Sort, also known as Stupid Sort, is like a garden gnome sorting his flower pots. The gnome looks at the pot next to him and the one before that. If they're in the right order, he moves forward. If they're not, he swaps them and moves backward. If he's at the beginning of the line, he moves forward, and if he's at the end, he's done.

Working Principle

- ***Underlined elements** are the pair under consideration.*
- ***"Red" colored** are the pair which needs to be swapped.*
- ***Result of the swapping is colored as "blue"***

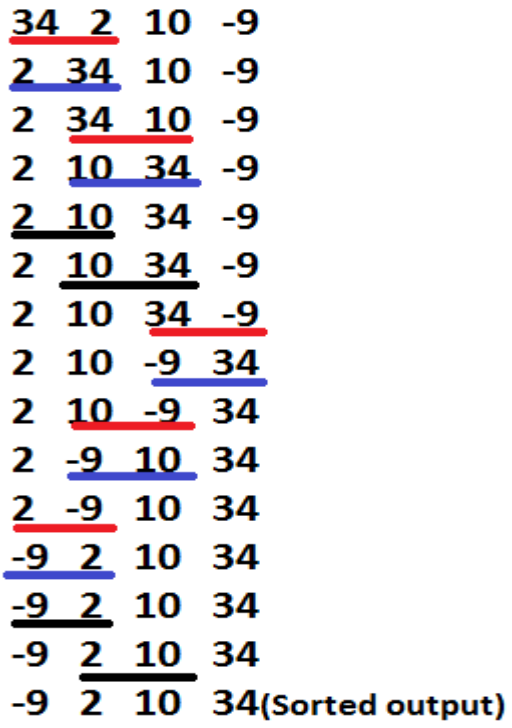
- If you are at the start of the array then go to the right element (from arr[0] to arr[1]).
- If the current element is greater than or equal to the previous element, move one step to the right.

```
if (arr[i] >= arr[i-1])
i++;
```

- If the current element is smaller than the previous element, swap the two elements and move one step to the left.

```
if (arr[i] < arr[i-1])
{
swap(arr[i], arr[i-1]);
i–;
}
```

- Repeat steps 2 and 3 until you reach the end of the array.
- Stop when you reach the end of the array, and the array will be sorted.



Time Complexity

- ❖ Worst case time O(n^2)
- ❖ Best case time O(n)
- ❖ Average case time O(n^2)
- ❖ Space complexity O(1) .

Running Time

- When the list is already sorted in ascending order, Gnome sort will have a time complexity of O(n), meaning that it only needs to walk through the elements once without changing any of them.
- However, if the list is sorted in descending order, the time complexity will be O(n^2), as Gnome sort will need to increase and decrease while iterating through the list.
- When the list is not sorted, the time complexity will also be O(n^2) because the algorithm will need to perform some comparisons and possibly move elements back and forth, resulting in some increases and decreases that will take up to n^2 time, depending on the length of the list.

Stability?

Stable sort algorithm.

Comparison-based or non-comparison-based?

Comparison-based algorithm

In place or not?

in place

Conclusion

Comparing the five-sorting algorithm

	Counting Sort	Comb Sort	Cycle sort	Pigeonhole sort	Gnome sort
Worst Case	O(n)	O(n^2)	O(n^2)	O(n)	O(n^2)
Best Case	O(n)	O(n log n)	O(n^2)	O(n)	O(n)
Average Case	O(n)	O(N^2/2^p)	O(n^2)	O(n)	O(n^2)
Space Complexity	O(n)	O(1)	O(1)	O(n)	O(1)
Stability	Stable	not stable	not stable	Stable	Stable
Comparison-based or non-comparison based	Non-comparison based	Comparison-based	Comparison-based	Non-comparison based	Comparison-based
Place in/out	Out	In	In	Out	In

In conclusion, the five sorting algorithms: counting sort, comb sort, cycle sort, pigeonhole sort, and gnome sort, all have their unique strengths and weaknesses.

Counting sort is particularly efficient when the range of key values is limited and can achieve a time complexity of O(n + k). It is stable and easy to implement.

Comb sort is a variation of bubble sort that can perform better than bubble sort, particularly when the array is already partially sorted. However, its worst-case time complexity is O(n^2), and it may not be as efficient as other sorting algorithms.

Cycle sort is an in-place sorting algorithm that minimizes the number of writes to memory. It can perform well on small arrays, but its worst-case time complexity is O(n^2), and it may not be as efficient as other sorting algorithms for larger arrays.

Pigeonhole sort is a sorting algorithm that is useful when the number of elements to be sorted is approximately the same as the range of key values. It has a time complexity of O(n + k), but its space complexity can be a limiting factor.

Gnome sort is a simple sorting algorithm that is easy to understand and implement. However, its worst-case time complexity is O(n^2), and it may not be as efficient as other sorting algorithms for larger arrays.

When we examine the table, we see that counting sort and pigeonhole sort are similar except for one thing. In counting sort, we require two arrays along with the given array to move the elements once, while in pigeonhole sort, we only need one additional array to move the elements twice. First, the elements are transferred from the given array to the second array, then sorted and finally returned to the original array.

In summary, the choice of sorting algorithm will depend on the specific requirements of the problem at hand, such as the size of the input array, the range of key values, and the need for stability. It is important to consider the strengths and weaknesses of each algorithm to make an informed decision.

References

https://www.geeksforgeeks.org/counting-sort/
https://www.geeksforgeeks.org/comb-sort/
https://www.geeksforgeeks.org/cycle-sort/
https://www.geeksforgeeks.org/pigeonhole-sort/
https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/