Faculty of Engineering & Technology Electrical & Computer Engineering Department

Data Structure and Algorithms- COMP2421

Project No.4 – Counting Sort, Pigeonhole sort, and Gnome Sort

- **Name:** Yousef Ahmad Jallad
- **ID:** 1190030
- **Instructor:** Ahmad Abusnaina
- **Date:** 30-12-2021

Introduction

Sorting is generally known as arranging data in a particular form, but in data structure we use algorithms which is the way used to arrange the data in ascending or descending manner. There are many known sorting algorithms and each one of them is used for particular problem. Anyway, this report focused on three sorting types and discussed them in depth, and they are: Pigeonhole sort, Gnome sort, and Counting sort.

➔Counting Sorting

⇨ Overview:

Counting sort is a non-comparison-based sort, it only handle arrays that have nonnegative integer values and nothing else, however this restriction allows it to be much more efficient than more generic comparison-based sorting algorithms, moreover, it doesn't require the use of a dynamic array as other non-comparison algorithms like Radix sort, in addition it is only useful where the max number isn't significantly larger than the number of elements. The way it works is that first it counts up how many times each distinctive element appears in the array, then we use these counts to put it in temporary array that has length of the max element in the original array (unsorted) incremented by one, after that we fill this array by the number of appearances of each index on the new index of temporary array, then we use the prefix sum to figure out how many elements are less than or equal to each particular index in the unsorted array, finally we create the final array (sorted) moving from right to left, we will go through each element in the original array (unsorted) starting at the end moving towards the front, to see the value of the temporary array at that index then take the value (decremented by one) as an index for the final array(sorted) and save the index of the value of temporary array into the final array(sorted).

⇨ Algorithm:

1- Find the max element from the original array.
2- Create a temporary array with size max element + 1, and make the initial value for each one is zero.
3- Store the number of appearances of each element in the original array in their respective index in the temporary array.
4- we use the prefix sum to figure out how many elements are less than or equal to each particular index in the unsorted array.
5- Start from the last index of original array, and find it's respective index in the temporary array then decrement it by one to find the index in the final array and store the counts' elements index into the index of the final array.

⇨ Algorithm Analysis:

- Time Complexity:
  this type of sort doesn't concern how the initial elements are sorted as it makes no difference in the difference of time complexity. n is array size and k is the max number of the array, as follows:
  →Worst case: O(n + k)
  →Average case: O(n + k)
  →Best case: O(n + k)

- Space Complexity:
  In this type of sorting, there is use of extra space, space complexity is O(n + k).

⇨ Results:

1- It doesn't require a dynamic array as another non-comparison-based sorting algorithm.
2- Only work with nonnegative integers, which make it much more efficient.
3- Counting sorting is stable, which means that two equal values will have the same order before and after sorting.
4- Counting sorting is out-of-place, uses extra spaces to complete the sorting of elements.

➔Pigeonhole Sorting

- ⇨ Overview:
  Pigeonhole sorting is non-comparison-based algorithm. It only handle integer arrays, which is similar to radix sorting in terms of implementation as both can be executed with a dynamic array, which in our case is an array of linked lists, also it is similar to counting sort both of them use the keys of elements into an auxiliary array, the difference is that while counting sort indexes into this array to keep track of the counts of each element, pigeonhole sort actually copies over the element itself and appends it to a list that is contained within each index of auxiliary array. so, the way it works is like this: first elements Are stored in an array then moved to an array of linked lists where each key value has a cell of its own (number of cells depends on the range of possible key values in original array), after every key is allocated in the array of linked list, we move the elements back to the original array. Finally, we can conclude that this kind of algorithm is only efficient when the range of elements is close to the number elements.

- ⇨ Algorithm:
  1- Find the minimum element and maximum element in the array.
  2- Find the range, Range= ($Max_{element} - Min_{element}$) +1.
  3- Allocate an array of linked lists with the range size and make it empty.
  4- Each cell has a key value as the values range from 0 to (Range-1).
  5- Each element is stored into the list with key = element – $Min_{element}$.
  6- Move the elements back to the original array in order.

- ⇨ Algorithm Analysis:
  - Time Complexity:
    In this type of sort, the order of data doesn't affect the change of time complexity as it keeps the same, because we have to transfer the data from and to the array, as well as filling the array of linked list with the elements using multiple loops that take time as following:
    →Worst case: O(n +k)
    →Average case: O(n + k)
    →Best case: O(n + k)

  - Space Complexity:
    This type of Algorithm needs an extra space because of the array of linked list that is being allocated as it takes: O(n + k), where k is the number of the pigeon holes for the key values and n is the number of elements.

⇨ Results:

1- It is more efficient when the range of elements is close to the number of elements, because when the range of elements is much larger than the number of elements it is considered to be a waste of space.
2- It only handles with integers, negative integers can cause issues.
3- Pigeonhole sorting is stable, which means that two equal values will have the same order before and after sorting.
4- Pigeonhole sorting is out-of-place, uses extra spaces to complete the sorting of elements.

➔Gnome Sorting

- ⇨ Overview:
  Gnome sort or stupid sort as it was called is a comparison-based sort method, it is very similar to Insertion and Bubble sort, as it only works with one element at one time while getting the element in the right place with a series of swaps like the bubble sort does, the main difference between Gnome and Bubble sort is that Gnome sort doesn't require the use of nested loops, but, finds the first place where two adjacent elements are in wrong order and swaps them, moreover, if the two elements on the right and the left are sorted then it moves forward if not it swaps them and moves a step backwards, other than that if we are at the first element we move one element forward accordingly, and if there is no element next to the current element then it is done.

- ⇨ Algorithm:
  1- Check if the element is the first element, if it is then we move to the next element.
  2- A) Compare the current element with the previous element, if equal or larger, then move to the next element.
  B) if the current element is smaller than we swap the two elements and move a step back to the previous element.

  3- Repeat step 2A and 2B until there is no next element.

- ⇨ Algorithm Analysis:
  - Time Complexity:
    In Gnome sorting, the sorted in ascending order is characterized as the best case, because the index keeps incrementing with no decrementing, on the contrary, average and worst cases are not sorted and sorted in descending order respectively, the reason is that the index may increment and decrement multiple times, as follows:
    →Worst case: $O(n^2)$
    →Average case: $O(n^2)$
    →Best case: $O(n)$

  - Space Complexity:
    In this method of sorting, neither extra space or recursion were required. As a result, space complexity is O(1).

- ⇨ Results:
  1- One of its advantages is that it is relatively easy to implement.
  2- Gnome sorting is stable, which means that two equal values will have the same order before and after sorting.
  3- Gnome sorting is out-of-place, uses extra spaces to complete the sorting of elements

➔Summary

The three sorting methods that were presented in this report are all critical methods that have added a lot to the sorting concept. Every single one of these sorting methods has its own way to sort the elements. These ways of sorting may have solved some problems or added new problems through their limitations, advantages, and disadvantages.

⇨ Properties Comparison:

Property/ Sort types	Count Sort	Pigeonhole Sort	Gnome sort
Time Complexity	Worst: O(n + k) Average: O(n + k) Best: O(n + k)	Worst: O(n + k) Average: O(n + k) Best: O(n + k)	Worst: $O(n^2)$ Average: $O(n^2)$ Best: O(n)
Space Complexity	O(n + k)	$O(n^2)$	O(1)
Stability	Yes	Yes	Yes
In-place	No	No	No

In conclusion, all three types are not practical, as every algorithm has its advantages and disadvantages, and every algorithm has a scenario that is best to use it in, because of their restrictions.

➔References:

1- GeeksForGeeks. Counting Sorting (online) https://www.geeksforgeeks.org/counting-sort/#:~:text=Counting%20sort%20is%20a%20sorting,object%20in%20the%20output%20sequence.
2- Programiz. Counting Sorting (online) https://www.programiz.com/dsa/counting-sort
3- Wikipedia. Counting sorting (online) https://en.wikipedia.org/wiki/Counting_sort
4- Wikipedia. Pigeonhole sorting (online) https://en.wikipedia.org/wiki/Pigeonhole_sort#:~:text=Pigeonhole%20sorting%20is%20a%20sorting,(n%20%2B%20N)%20time.
5- Xlinux.nist.gov. Pigeonhole Sorting. (online) https://xlinux.nist.gov/dads/HTML/pigeonholeSort.html
6- Wikipedia. Gnome Sorting (online) https://en.wikipedia.org/wiki/Gnome_sort
7- GeeksForGeeks. Gnome Sorting(online) https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/