Data Structures

COMP2321

Research Paper No. 2

Sorting Algorithms Report

Instructor: Dr. Radi Jarrar

Name: Anas Nimer

ID: 1180180

Section: 4

Contents:

1) Counting sort:

Counting sort is an efficient algorithm for sorting an array of elements that each have a nonnegative integer key, for example, an array, sometimes called a list, of positive integers could have keys that are just the value of the integer as the key, or a list of words could have keys assigned to them by some scheme mapping the alphabet to integers (to sort in alphabetical order, for instance). Unlike other sorting algorithms, such as merge sort, counting sort is an integer-sorting algorithm, not a comparison based algorithm. While any comparison based sorting algorithm requires O(nlogn) comparisons, counting sort has a running time of when the length of the input list is not much smaller than the largest key value , in the list. Counting sort can be used as a subroutine for other, more powerful, sorting algorithms such as radix sort.

- **Algorithm:**

```
Begin
   max = get maximum element from array.
   define count array of size [max+1]

   for i := 0 to max do
      count[i] = 0 //set all elements in the count array to 0
   done

   for i := 1 to size do
      increase count of each number which have found in the array
   done

   for i := 1 to max do
      count[i] = count[i] + count[i+1] //find cumulative frequency
   done

   for i := size to 1 decrease by 1 do
```

```
      store the number in the output array
      decrease count[i]
   done

   return the output array
End
```

- **Mechanism that Comb sort works :**

Find out the maximum of the element of the specified array and configure a set of maximum length + 1 for all elements 0. This array is used to store the number of elements in the array. Then store the number of each element in their index in the count array and store the cumulative sum of the elements of the count array. It helps put items in the correct index for the sorted array. Then Find the index of each element of the original array in the count array. and finally placing each element at its correct position, decrease its count by one.

- **Properties:**

Worst case time complexity	o(n+k)=o(n)
Average case time complexity	o(n+k)=o(n)
Best case time complexity	o(n+k)=o(n)
Space complexity	o(n+k)=o(n)
Stability	Yes , it is Stabil
In Place	Yes, it is in Place

- ❖ n is the number of values to be sorted, k is the largest number of the numbers.
- ❖ This sorting technique is effective when the difference between different keys are not so big, otherwise, it can increase the space complexity.

2) Comb sort:

Comb Sort is mainly an improvement over Bubble Sort. Bubble sort always compares adjacent values. So all 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 with one swap and performs better than Bubble Sort.

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

- **Algorithm:**

```
Begin
   gap := size
   flag := true
   while the gap ≠ 1 OR flag = true do
      gap = floor(gap/1.3) //the the floor value after division
      if gap < 1 then
         gap := 1
      flag = false

      for i := 0 to size – gap -1 do
         if array[i] > array[i+gap] then
            swap array[i] with array[i+gap]
            flag = true;
      done
   done
End
```

- **Mechanism that Comb sort works :**

Set the gap initially to the length of the array wanted to sort , Then Calculate the new gap by dividing gap by 1.3 , after that Compare array[i] with array[i + gap] , if the first is element is greater than the second then swap them and set the swap flag to 1, repeating The previous two steps until gap=1 and swap Flag=0.

- **Properties:**

Worst case time complexity	$o(n^2)$
Average case time complexity	$o(n^2/2^p)$
Best case time complexity	o(n)
Space complexity	o(1)/ constant
Stability	NO, it isn't Stabil
In Place	NO, it is not in Place

- ❖ n is the number of values to be sorted, p is the number of increments.
- ❖ This sorting technique is effective when the values are sorted.
- ❖ This sorting technique is defective when the values are arranged in reverse order and when the values not sorted.

Gnome sort:

Gnome Sort also called Stupid sort is based on the concept of a Garden Gnome sorting his flowerpots. A garden gnome sorts the flowerpots by the following method- Put items in order by comparing the current item with the previous item. If they are in order, move to the next item (or stop if the end is reached). If they are out of order, swap them and move to the previous item. If there is no previous item, move to the next item.

- **Algorithm:**

```
procedure gnomeSort(a[])
    pos := 1
    while pos < length(a)
        if (a[pos] >= a[pos-1])
            pos := pos + 1
        else
            swap a[pos] and a[pos-1]
            if (pos > 1)
                pos := pos - 1
            end if
        end if
    end while
end procedure
```

- **Mechanism that Comb sort works :**

He looks at the flowerpot next to him and the previous one; if they are in the right order he steps one pot forward, otherwise he swaps them and steps one pot backwards.
If there is no previous pot (he is at the starting of the pot line), he steps forwards; if there is no pot next to him (he is at the end of the pot line), he is done.

- **Properties:**

Worst case time complexity	$o(n^2)$
Average case time complexity	$o(n^2)$
Best case time complexity	o(n)
Space complexity	o(1)/ constant
Stability	Yes , it is Stabil
In Place	NO, it is not in Place

- ❖ n is the number of values to be sorted.
- ❖ This sorting technique is effective when the values are sorted.
- ❖ This sorting technique is defective when the values are arranged in reverse order and when the values not sorted.

Summary:

Name	Worst time complexity	Average time complexity	Best time complexity	Space complexity	Stable
Counting sort	O(n)	O(n)	O(n)	O(n	Yes
Comb sort	$O(n^2)$	**$O(n^2/2^p)$**	**O(n)**	O(1)/constant	NO
Gnome sort	$O(n^2)$	$O(n^2)$	O(n)	O(1)/constant	Yes

Note: From the above table we can see the counting sort is the best sorting algorithm between them. However, it cannot be used for float or negative numbers.

Gnome sort is also efficient, since it does not use any extra space, and it is fast if the data is nearly sorted.

References:

https://brilliant.org/wiki/counting-sort/

https://www.tutorialspoint.com/Counting-Sort

https://www.geeksforgeeks.org/comb-sort/

https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/