International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 04 Issue: 01 | Jan -2017 www.irjet.net p-ISSN: 2395-0072

A comparative Study of Sorting Algorithms Comb, Cocktail and Counting Sorting

Ahmad H. Elkahlout[1], Ashraf Y. A. Maghari[2].

[1] *Faculty of Information Technology, Islamic University of Gaza - Palestine*
[2] *Faculty of Information Technology, Islamic University of Gaza - Palestine*

Abstract – *The sorting algorithms problem is probably one of the most famous problems that used in abroad variety of application. There are many techniques to solve the sorting problem. In this paper, we conduct a comparative study to evaluate the performance of three algorithms; comb, cocktail and count sorting algorithms in terms of execution time. Java programing is used to implement the algorithms using numeric data on the same platform conditions. Among the three algorithms, we found out that the cocktail algorithm has the shortest execution time; while counting sort comes in the second order. Furthermore, Comb comes in the last order in term of execution time*. Future effort will investigate the memory space complexity.

Key Words: Sorting, Comb, Cocktail, Counting, Algorithm, Comparison, Experimental.

1. Introduction

Sorting algorithm is the process of placing elements from a collection in some kind of order. For example, a list of words could be sorted alphabetically or by length. A list of cities could be sorted by population, by area, or by zip code[1] . The most-used orders are numerical order and lexicographical order [2]. The sorting is a problem of central importance both in theoretical and practical computing [3].Efficient sorting is important for optimizing the use of other algorithms such as search and merge algorithms which require input data to be in sorted lists. Further, the data is often taken to be in an array, which allows random access, rather than a list, which only allows sequential access. The sorting n algorithms can be applied with suitable modification to either type of data. Since the dawn of computing, the sorting problem has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple, familiar statement. We have already seen a number of algorithms that were able to sort unsorted list by different ways and solve the sorting problem. Computer sorting algorithms can be classified into two categories; comparison-based sorting and non-comparison-based sorting. The purpose of this study is making experimental comparison between three sorting algorithm Comb, Cocktail and Counting Sort to determine experimentally which one is the fastest in term of execution time. Three sorting algorithms are implemented using by java programming language, in which every algorithm sort the same set of random numeric data. Then the execution time during the sorting process is measured for each algorithm.

The rest part of this paper is organized as follows. Section 2 gives a background of the three algorithms; comb, count, and cocktail algorithms. In section 3, related work is presented. Section4 comparison between the algorithms; section 5 discusses the results obtained from the evaluation of sorting techniques considered. The paper is concluded in section 6.

2. Background

2.1. Comb Sort

Comb sort is an enhanced version from bubble sort and rediscovered by Stephen Lacey and Richard Box in 1991. [9]. 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 shown in figure 2 until it reaches the value 1. Thus, Comb Sort removes more than one inversion counts with one swap and performs better than Bubble Sort. Figure 1 shows how Comb sorting algorithm works. In average it works better than Bubble Sort. Its worst case, however, remains $O(n^2)$.

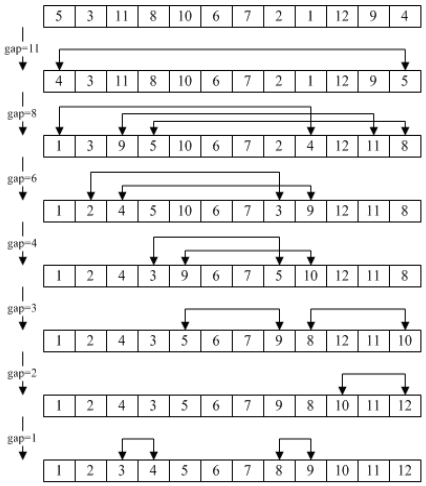

Figure 1: Comb sort

```
function comb sort(array input, int size)
        gap ← size
        shrink ← 1.3
  loop until gap = 1 and swapped = true
      gap ← int(gap / shrink)
            if gap < 1
             gap ← 1
            end if
            i ← 0
       swapped ← false
    loop until i >= size - gap
     if input[i] > input[i+gap]
    swap(input[i], input[i+gap])
        swapped ← true
           end if
         i ← i + 1
        end loop
       end  loop
      end function
```

Figure 2: Pseudo code: comb Algorithm

2.2. Cocktail Sort:

Cocktail sort is similar to the bubble sort in which both are even and comparison-based sorting algorithms as shown in Figure 4. The Cocktail algorithm differs from a bubble sort in that it sorts in both directions on each pass through the list as shown in Figure 3. This sorting algorithm is more difficult to implement than the bubble sort [4]. The first rightward pass will shift the largest element to its correct place at the end, and the following leftward pass will shift the smallest element to its correct place at the beginning. The second complete pass will shift the second largest and second smallest elements to their correct places, and so on.

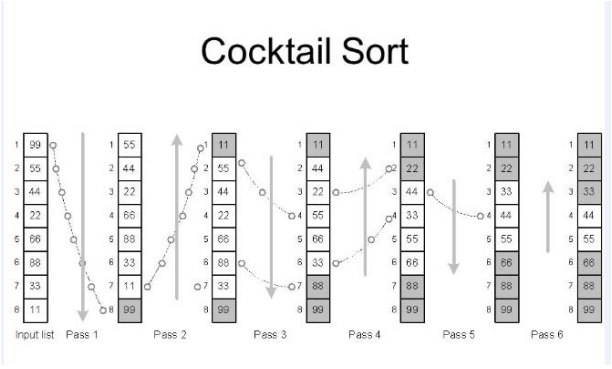

Figure 3: cocktail sort

```
procedure cocktail Sort( A : list of sortable items ) defined as: do
        swapped ← false
    if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] )
       swapped ← true
          end if
        end for
     if swapped = false then
      break do-while loop
        end if
        swapped ← false
      for each i = length( A ) - 1 to 0 do:
          if A[ i ] > A[ i + 1 ] then
swap( A[ i ], A[ i + 1 ] )
       swapped := true
         end if
         end for
       while swapped
      end procedure
```

Figure 4: Pseudo code: Cocktail Algorithm

2.3. Counting Sort:

Counting sort is an algorithm for sorting a collection of objects according to keys that are small integers. Figure 5 shows a pseudo code for the counting sort. It operates by counting the number of objects that have distinct key value, and using arithmetic on those counts to determine the positions of each key value in the output sequence [5]. An important property of counting sort is that it is stable: numbers with the same value appear in the output array in the same order as they do in the input array as shown in Figure 6. That is, it breaks ties between two numbers by the rule that whichever number appears first in the input array it also appears first in the output array. Normally, the property of stability is important only when satellite data are carried around with the element being sorted. Counting sort's stability is important for another reason: counting sort is often used as a subroutine in radix sort, in order for radix sort to work correctly, counting sort must be stable [3].

```
1.  Counting-Sort(A, B, k)
2.  Let C[0.....k] be a new array
3.  for i=0 to k                          [Loop 1]
4.      C[i]= 0;
5.  for j=1 to A.length  or n             [Loop 2]
6.      C[ A[j] ] = C[ A[j] ]  + 1;
7.  for i=1 to k                          [Loop 3]
8.      C[i] = C[i] + C[i-1];
9.  for j=n or A.length down to 1         [Loop 4]
10.     B[  C[ A[j] ]  ] = A[j];
11.     C[ A[j] ] = C[ A[j] ]  - 1;
```

Loop 1 and 3 takes O(k) time

Loop 2 and 4 takes O(n) time

Figure 5: pseudo code: counting sort

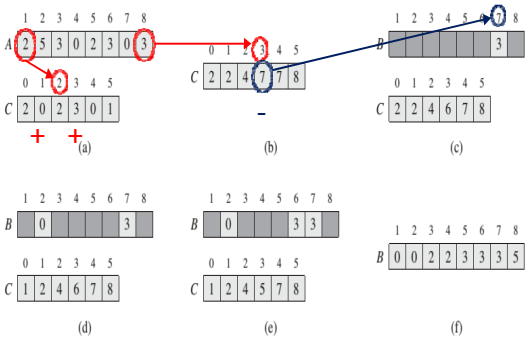

Figure 6: Counting sort

3. Related work

Computer sorting algorithms can be classified into two categories, comparison-based and non-comparison-based sorting. In recent years, some improved algorithms have been proposed to speed up comparison-based sorting algorithms, such as advanced quick sort according to data distribution characteristics [6]. Mishra, A.D. and Garg, D. [3] reported that the execution of any counting or computation refers to the sorting algorithm performance. Like any computational problem, similar results are given from many solutions. In [7], the author evaluates the performance of three algorithms; insertion, selection, and bubble sorting algorithms. Their work is evaluated in terms of CPU execution time and memory space. The most efficient algorithm was the insertion sort. Selection sort came in the second order and in the last order came the bubble sort. Khalid Suleiman Al-Kharabsheh [9][10] et al. suggested a GCS (Grouping Comparison Sorting) algorithm which makes further comparison with others conventional sorting algorithms like Bubble sort, Merge sort, Insertion sort, Quick sort and Selection sort, to show how these algorithms reduce time of execution.

Our work, however, is different in the context that the experimental comparison is conducted between three sorting algorithms; Comb, Cocktail and Counting Sort in term of execution time.

4. Experimental Result

In our experiments, the three sorting algorithms have been implemented using Java programming. Then, time measuring method is developed to determine the execution time for each algorithm. A set of random numeric data is between 10 and 3000. The algorithms were implemented in Java-language. These were run on Intel core (R) i5-2410m, CPU 20.30GHz, RAM 4 G, operating system windows 7 64bt. Table 2 shows the CPU execution time for the three algorithms. We use numeric data between 10 and 3000. Figure 7 shows a graphical representation of the execution time of the three algorithms. The results demonstrate that the cocktail algorithms have the shortest execution time; while counting sort comes in the second order. Furthermore comb comes in the last order in term of execution time.

Table 1: Execution time measured with CPU with various data

Number	Procession Time		
	Comb	Cocktail	Counting
15	6	1	3
53	19	1	5
99	39	2	10
190	100	4	28
370	220	9	50
550	422	13	79
750	499	17	106
850	850	22	122
950	1250	50	266
3000	1980	66	380
Average	**538.5**	**18.5**	**104.9**

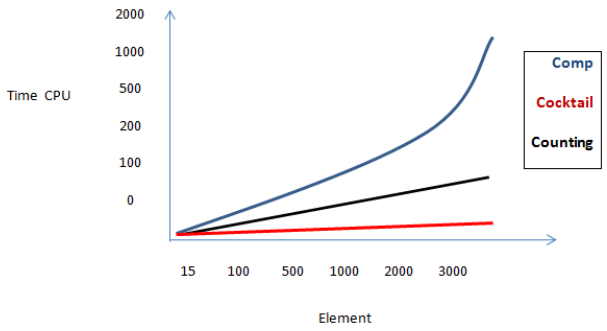

Figure 7: CPU Time vs. Data Size

Table 2: running time for Comb, Cocktail and Counting

Algorithm	Average case	Worst case	Best case
Cocktail	$O(n2)$	$O(n2)$	$O(n)$
Count	$O(n+k)$	$O(n+k)$	$O(n+k)$
Comb	$O(n2)$	$O(n2)$	$\Theta(n \log n)$

5. Discussion

This section presents the discussion of results obtained from evaluation of comb, cocktail, and count sort algorithms. Cocktail have variation from comb sort and count sort in which it passes alternately from bottom to top and then from top to bottom [8]. It can achieve slightly better performance than count and comb sort. The reason for this is that comb and count only passes through the list in one direction.

6. Conclusion

We have evaluated the performance of comb, cocktail, and count sort techniques using CPU time. This was achieved by reviewing literatures of relevant works. We also formulated architectural model which serves as guideline for implementing and evaluating the sorting techniques. The techniques were implemented with Java language. Empirical results were tabulated and graphically presented. The results obtained show that in majority of the cases considered, cocktail sort technique is faster. While counting sort comes in the second order and Comb comes in the last order in term of execution time. Future efforts should be made to compare the memory usage of sorting algorithms.

7. References

1. Huang, Z., S. Kannan, and S. Khanna. *Algorithms for the generalized sorting problem*. in *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. 2011. IEEE.
2. Demuth, H.B., *Electronic data sorting.* IEEE transactions on computers, 1985. **100**(4): p. 296-310.
3. Mishra, A.D. and D. Garg, *Selection of best sorting algorithm.* International Journal of Intelligent Information Processing, 2008. **2**(2): p. 363-368.
4. Cormen, T.H., et al., *Introduction to algorithms*. Vol. 6. 2001: MIT press Cambridge.
5. Adhikari, P., *Review On Sorting Algorithms A comparative study on two sorting algorithms.* Mississippi State, Mississippi, 2007. **4**.
6. Horsmalahti, P., *Comparison of Bucket Sort and RADIX Sort.* arXiv preprint arXiv:1206.3511, 2012.
7. Aremu, D., et al., *A Comparative Study of Sorting Algorithms.* African Journal of Computing & ICT, 2013. **6**(5).
8. Cunningham, J.G., et al. *Methods for identifying systematic differential reflectivity (ZDR) biases on the operational WSR-88D network*. in *36th Conference on Radar Meteorology*. 2013.
9. Khalid Suleiman Al-Kharabsheh, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani, and Nabeel Imhammed Zanoon, “Review on Sorting Algorithms A Comparative Study”, International Journal of Computer Science and Security (IJCSS), Vol. 7, No. 3, 2013
10. I. trini, k. kharabsheh, and A. trini, "Grouping Comparison Sort", Australian Journal of Basic and Applied Sciences, pp. 221-228, May 2016.

BIOGRAPHIES

Mr. Ahmad H. Elkahlout is the corresponding author of this paper and a master student at the Faculty of Information Technology, Islamic University of Gaza, Palestine.
Phone Number: +97299604246
e-mail: a_h_k80@hotmaill.com