IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Impact Factor: 6.252

Volume 2, Issue 6, June 2022

Improvement of Bubble Sort Time Complexity in Average Case

Mohit Sharma
Student, Department of CSE
Dronacharya College of Engineering, Gurgaon, Haryana, India
mohitsharma26900@gmail.com

Abstract: *The Time Complexity of Bubble Sort Algorithm is O(n^2) in all the three cases (best case, average case and worst case) but after optimization we can improve the bubble sort Time Complexity to O(n) in best case and average case..*

Keywords: Bubble Sort, Algorithm, Array Data Structure

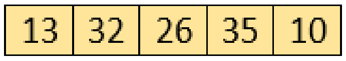
I. INTRODUCTION

Bubble sort works on the repeatedly swapping of adjacent elements until they are not in the intended order. It is called bubble sort because the movement of array elements is just like the movement of air bubbles in the water. Bubbles in water rise up to the surface; similarly, the array elements in bubble sort move to the end in each iteration.
Although it is simple to use, it is primarily used as an educational tool because the performance of bubble sort is poor in the real world. It is not suitable for large data sets. The average and worst-case complexity of Bubble sort is **$O(n^2)$,** where **n** is a number of items.

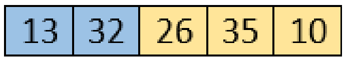
II. PROPOSED ALGORITHM

2.1 Working of Bubble Sort

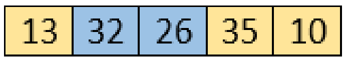
Now, let's see the working of Bubble sort Algorithm.
To understand the working of bubble sort algorithm, let's take an unsorted array. We are taking a short and accurate array, as we know the complexity of bubble sort is **$O(n^2)$.**
Let the elements of array are -



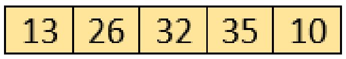
First Pass
Sorting will start from the initial two elements. Let compare them to check which is greater.



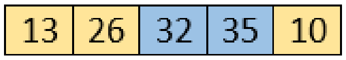
Here, 32 is greater than 13 (32 > 13), so it is already sorted. Now, compare 32 with 26.



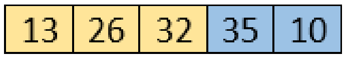
Here, 26 is smaller than 36. So, swapping is required. After swapping new array will look like -



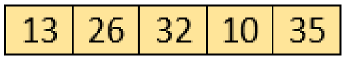
Now, compare 32 and 35.



Here, 35 is greater than 32. So, there is no swapping required as they are already sorted.
Now, the comparison will be in between 35 and 10.

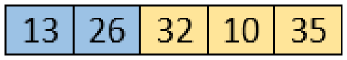


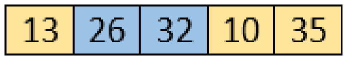
Here, 10 is smaller than 35 that are not sorted. So, swapping is required. Now, we reach at the end of the array. After first pass, the array will be -

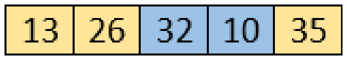


Now, move to the second iteration.

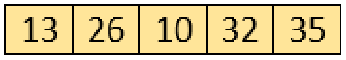
Second Pass
The same process will be followed for second iteration.

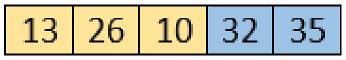






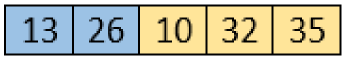
Here, 10 is smaller than 32. So, swapping is required. After swapping, the array will be -

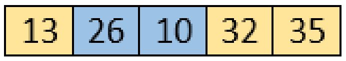




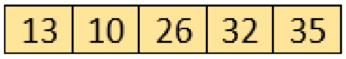
Now, move to the third iteration.

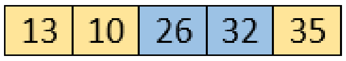
Third Pass
The same process will be followed for third iteration.

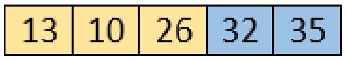




Here, 10 is smaller than 26. So, swapping is required. After swapping, the array will be -

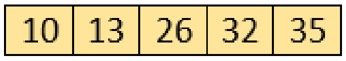






Now, move to the fourth iteration.

Fourth pass
Similarly, after the fourth iteration, the array will be -



Hence, there is no swapping required, so the array is completely sorted.

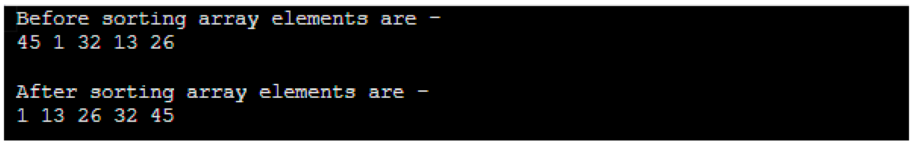
2.2 Working of Optimized Bubble Sort

In the bubble sort algorithm, comparisons are made even when the array is already sorted. Because of that, the execution time increases.
To solve it, we can use an extra variable ***swapped.*** It is set to **true** if swapping requires; otherwise, it is set to **false.**
It will be helpful, as suppose after an iteration, if there is no swapping required, the value of variable **swapped** will be **false.** It means that the elements are already sorted, and no further iterations are required.
This method will reduce the execution time and also optimizes the bubble sort.

III. EXPERIMENT AND RESULT

Program and Output of Optimized Bubble Sort Algorithm

```
void bubbleSort(int *arr, int n)
{
   for(int i=0; i<n; i++)
   {
     bool flag = false;
      for(int j=0; j<n-i-1; j++)
      {
        if(array[j]>array[j+1])
        {
          flag = true;
           int temp = array[j+1];
           array[j+1] = array[j];
           array[j] = temp;
        }
      }
     // No Swapping happened, array is sorted
     if(!flag){
        return;
     }
   }
}
```



```
Before sorting array elements are -
45 1 32 13 26

After sorting array elements are -
1 13 26 32 45
```

IV.CONCLUSION

Note, that if all the passes are performed, then our optimized algorithm will in fact perform a little slower than the original one. But for the best case (Array already sorted) it will be O(n), For average case also the performance will see an improvement. Whereas the original algorithm was O(n^2) for all the cases.

REFERENCES

[1]. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Problem 2-2, pg.40.

[2]. Sorting in the Presence of Branch Prediction and Caches

[3]. Fundamentals of Data Structures by Ellis Horowitz, Sartaj Sahni and Susan Anderson-Freed ISBN 81-7371-605-6

[4]. Owen Astrachan. Bubble Sort: An Archaeological Algorithmic Analysis

[5]. Computer Integrated Manufacturing by Spasic PhD, Srdic MSc, Open Source, 1987.[1]