International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 7– No.8, September 2014 – www.ijais.org

Use-Me Sort: A new Sorting Algorithm

Mohit Sehgal
Engineering (IT) Student,
GTBIT, Indraprastha University,
Delhi, India

Nihal Kumar
Engineering (IT) Student,
GTBIT, Indraprastha University,
Delhi, India

ABSTRACT
One of the fundamental issues in Computer Science is the ordering of a list of items - known as sorting. Sorting algorithms such as the Bubble, Insertion and Selection Sort, all have a quadratic time complexity $O(N^2)$ that limits their use when the number of elements is very large.

This paper presents Use-Me sort. It sorts a list by making the use of already sorted elements present in the list. Moreover, it provides a trade-off between Space and Time Complexity with better performance than the existing sorting algorithms of the $O(N^2)$ class.

General Terms
NOC - Number of Comparisons, Space complexity, Time Complexity.

Keywords
Algorithm, Complexity, Insertion Sort

1. INTRODUCTION
Information is growing rapidly in today's world and to search for this information, it should be systematically organized. Algorithms such as Bubble, Insertion and Selection Sort, have an O(N2) time complexity that limits its usefulness to small number of elements no more than a few thousand elements.[5]

This paper presents Use-Me Sort which makes use of already sorted elements present in the array. It iterates from one end to the other, leaving behind a sorted array. Whenever a group of ordered elements is encountered, it is merged recursively with the sorted array behind. It follows a methodology, similar to Insertion Sort.

The aim of Use-Me sort is to reduce the Number of Comparisons (NOC) by using binary search instead of linear search – at the time of merging. The difference is not big for small N but it becomes significant when N goes high to some few ten thousand elements or more.

For further references, we will use a term called UM array for Use-Me array. In an unsorted array, UM array refers to an ordered sub-array which is out of order with respect to the array. It is important to note that the elements of the sub-array should be in some order. In Figure 1, if the array A[1..6] is to be sorted in ascending order then, array A[3..5] is a UM array.

1	5	4	3	1	9

Fig 1: Example of UM array

2. APPROACH
Use-Me Sort uses an approach which is as follows:

1) Iterate from the first element till the last to find a UM array.
2) Merge the UM array recursively with the sorted array behind.
3) Repeat the steps 1 and 2 till the end.

2.1 Pseudo Code
Input: Array A[0…N-1], N is length of the array.

Output: Array A[0…N-1] in ascending order.

```
USE-ME(A)
i←0
while i < N-1
    if A[i] > A[i+1]
        then k←i+1
        while  A[k] > A[k+1] and  k < N-1
            do k++
        if  k ≠ i+1
            then  m ← k-i-1
            DESC(A, i, k, m)
            i ← k
        else
            while A[k]≤A[k+1] and  k<N-1 and A[k+1]<A[i]
                do k++
            m ← k-i-1
            ASC(A, i, k, m)
            i ← k
    else
        i++
```

```
DESC(A, i, k, m)
if  m = -1
   then return 0
temp ← A[i+m+1]
l ← DESC(A, i, k, m-1)
if  temp < A[0]
    then low ← -1
else
    high ← i-l
```

International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 7– No.8, September 2014 – www.ijais.org

```
9      low←0
10     do  med ← (low+high)/2
11          if  temp > A[med]
12             low ← med
13          else
14             high ← med
15      while  low < high-1 go to line 10
16      while  temp ≥ A[low+1]
17          do  low++
18  k ← k-l-m
19  i ← i-l

20  while i > low
21      do A[k] ← A[i]
22      i--, k--, l++
23   A[k] ← temp
24   return l
```

```
ASC(A, i, k, m)
1   if  m = -1
2     then return 0
3   temp ← A[k-m]
4   l  ← ASC(A, i, k, m-1)
5   if  temp < A[0]
6      then low ← -1
7   else
8      high ← i-l
9      low←0
10     do  med ← (low+high)/2
11          if  temp > A[med]
12             low ← med
13          else
14             high ← med
15     while  low < high-1 go to line 10
16     while  temp ≥ A[low+1]
17          do  low++
18  k ← k-l-m
19  i ← i-l
20  while i > low
21      do A[k] ← A[i]
22      i--, k--, l++
23  A[k] ← temp
24  return l
```

2.2 Example

Consider the list as below:

2 3 5 4 3 1

STEP 1: Iterate from the first element to the last to find a UM array.

- As 5 < 4, set Indexes i = 3 and k = 4

  2 3 5(i) 4(k) 3 1

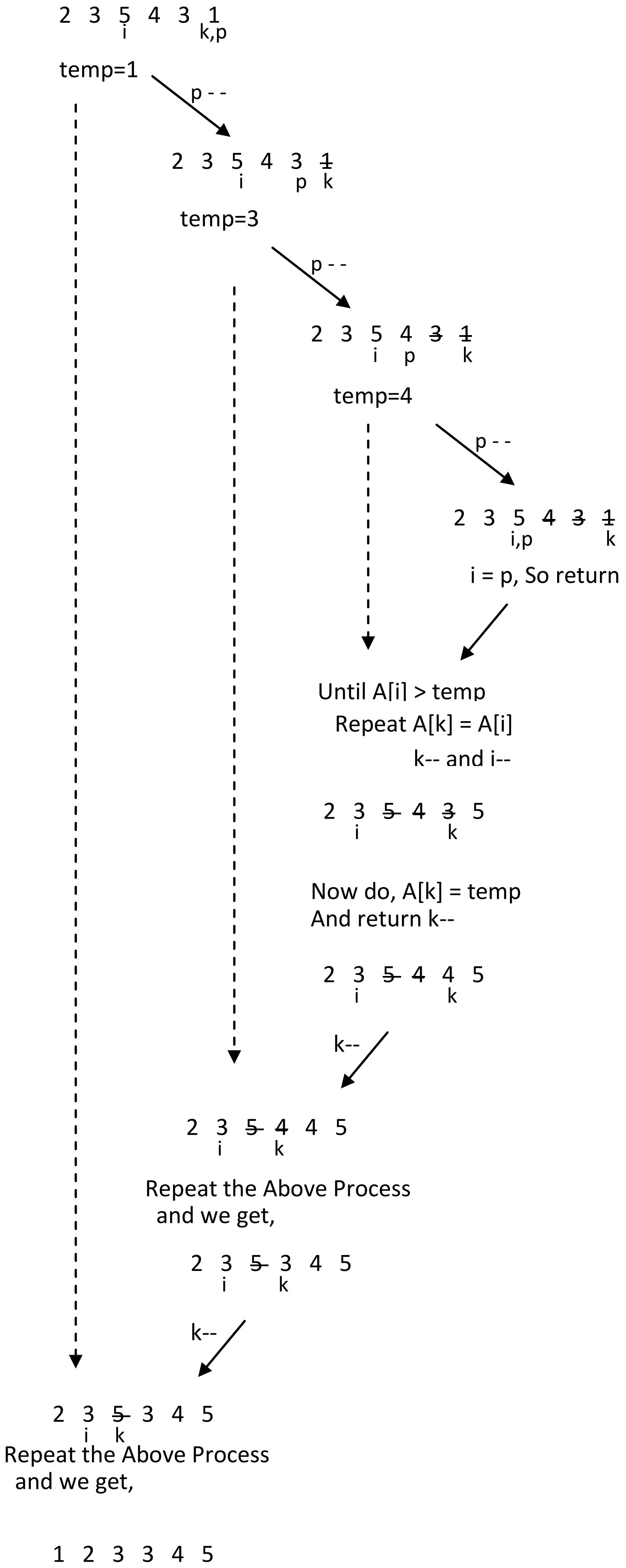
- Again 3 < 4, so k++

  2 3 5(i) 4 3(k) 1

- Again 1 < 3, so k++

  2 3 5(i) 4 3 1(k)

STEP 2: Recursively merge the already sorted array, A[1 to i] and UM array, A[i+1 to k]



International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 7– No.8, September 2014 – www.ijais.org

3. EVALUATION

3.1 Use of Binary Search

In Use-Me sort, the NOC is minimized by using binary search in merging two arrays. As we know the fact that, two arrays to be merged – will be already in order. Therefore binary search will be an effective way to minimize the NOC in merging the sub-arrays.

3.2 Trade-Off

It supports a Space-Time trade-off, in which run-time of an algorithm can be improved by making it use more memory or vice versa. To improve the runtime, we trade memory and for less memory usage we trade runtime.[2]

In Use-Me Sort, by restricting the size of the ordered sub-array, we can reduce the memory usage. As the memory decreases, the execution time(time to sort the array) will increase and hence, we can achieve Space-Time trade-off.

3.3 Stability

Stability simply means, maintaining the relative order of elements with equal value. In other words, a sorting algorithm is stable if whenever there are two records X and Y with the same value and with X appearing before Y in the original list, then X will always appear before Y in the sorted list. Use-Me algorithm is a stable algorithm.[8]

4. ANALYSIS

4.1 Complexity

The complexity of Use-Me algorithm is O(N) in the best case. In average case and worst case, when the UM array becomes small, the approximate complexity is O(Nlog(N)). This is because it will take N comparisons to iterate the array and utmost logN comparisons to find the correct location of an element of UM array. After summing all of it together the complexity becomes O(Nlog(N)). In case when the array is in decreasing order, the time complexity becomes less than O(Nlog(N)) because of the fact that the whole array will become UM array, apart from the first element. Therefore it will take operations, less than normal, to sort the array.

4.2 Comparison

When compared the proposed algorithm with Insertion Sort, it was inferred that the Use-Me algorithm shows a better results. The Table 1 given below shows the complexities of different sorting algorithms with the Use-Me algorithm[1]

Table 1. Comparison with other Algorithms

	Use-Me Sort	Bubble Sort	Selection Sort	Insertion Sort
Best Case Complexity	O(N)	$O(N^2)$	$O(N^2)$	O(N)
Average Case Complexity	O(Nlog(N))	$O(N^2)$	$O(N^2)$	$O(N^2)$
Worst Case Complexity	O(Nlog(N))	$O(N^2)$	$O(N^2)$	$O(N^2)$

The efficiency of the Use-Me sort algorithm will be measured in CPU time using the system clock [5] on a computer with minimal background processes running, with respect to the size of the input array, and compared with the insertion sort algorithm. Both the algorithms were executed with array size: 3k, 6k, 9k, 12k and 15k.The data sets used, include system generated random number. Algorithms were executed in C++. A comparison graph of CPU time Vs Array Size of Use-Me sort with Insertion sort has been shown in Figure 2 below.

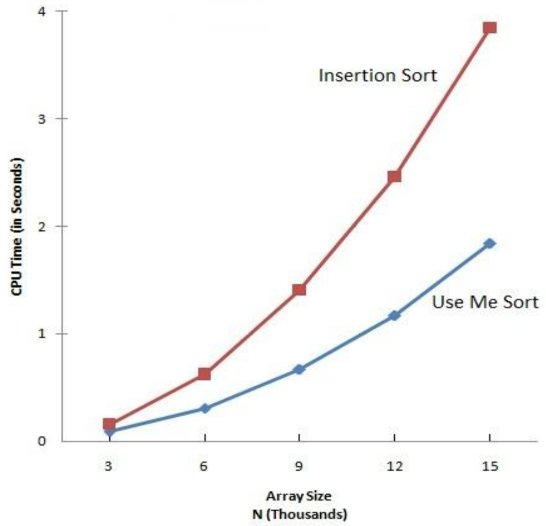


Fig 2: Comparison of Use-Me and Insertion Sort

The trade-off between Memory and Time Complexity is shown in Figure 3 below by a graph. This graph represents a comparison of Use-Me algorithm executed for a Data Set of 15000 elements for different allocated memory.

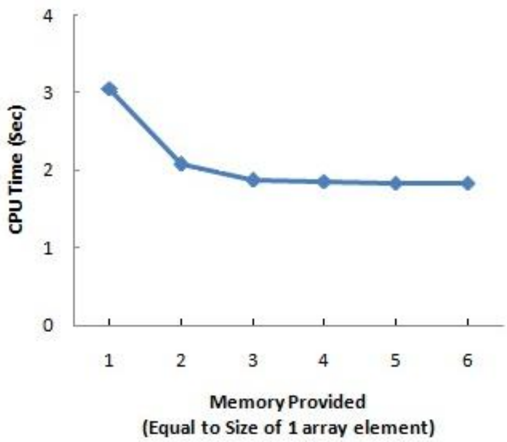


Fig 3: Comparison at different Allocated Memory

5. CONCLUSION

This paper introduced Use-Me Sort with Time Complexity O(Nlog(N)). It makes use of already ordered elements in the list and recursively merge them with the sorted list behind.

It also provides a trade-off between Memory and Time Complexity. This trade-off allows Use-Me sort to perform better for list having large number of elements.

In Striking contrast, Bubble, Insertion and Selection sort have quadratic time complexity that limit its use to a small number of elements. Use-Me is slightly faster than insertion sort when N is small but is much faster as N grows.[5]

From the results in Table 1, Figure 1 and Figure 2, we conclude that Use-Me algorithm has performed well in average case because it avoids as well as reduces the number of comparisons and number of swaps in comparison to the other algorithms.

6. ACKNOWLEDGEMENT

I would like to express heartfelt thanks to my friend Nihal Kumar for providing valuable feedback and comments. My family for a constant source of encouragement to me. I am deeply obliged to Mrs. Basantipal Ma'am of our College (GTBIT) who urged me to think that "how difficult it can be to develop your own Sorting Algorithm" and hence motivated me for it.

7. REFERENCES

[1] Knuth D.E., "The art of programming- sorting and searching". Addison-Wesley.

[2] Cormen T. , Leiserson C., Rivest R., and Stein C., "Introduction to Algorithms," McGraw Hill.

[3] Sedgewick, Algorithms in C++, pp.98-100, ISBN 0-201-51059-6, Addison-Wesley.

[4] Seymour Lipschutz, G A Vijayalakshmi Pai (2006), "Data Structures", Tata McGraw-Hill Publishing Company Limited.

[5] Song Qin, "Merge Sort Algorithm" Florida Institute of Technology.

[6] Vandana Sharma, Satwinder Singh and Dr.K.S.Kahlon, "Performance Study of Improved Heap Sort Algorithm and Other Sorting Algorithms on Different Platforms (2008), "IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.4.

[7] You Yang, Ping Yu, Yan Gan, "Experimental Study on the Five Sort Algorithms", International Conference on Mechanic Automation and Control Engineering (MACE), 2011.

[8] Kaur S., Sodhi T. S., Kumar P., (2012) "Freezing Sort". International Journal of Applied Information Systems (IJAIS), vol. 2, no. 4, pp. 18–21.

[9] Gurram, H.K., GovardhanaBabuKolli, (2011). Average Sort. International Journal of Experimental Algorithms (IJEA), vol. 2, no. 2, pp. 48–54.

[10] Wang Min "Analysis on 2-Element Insertion Sort Algorithm", International Conference on Computer Design And Appliations (ICCDA), 2010.

[11] Blum, M., Floyd, R.W., Pratt, V., Rivest, R.L., Tarjan, R.E., (1973). Time Bounds for Selection. Journal of Computer and System Sciences, vol. 7, no. 4, pp. 448–461.