International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 6, June - 2013

An Comparative Analysis Of NEET's (Negate Extended Elements Time In Sorting) Algorithm With Bubble Sort And Cocktail Sort In Matlab

Ayush Kumar Yogi
MCA Department, Govt. Engineering College, Ajmer

Abstract

NEET's (Negate Extended Elements Time in sorting) algorithm is new technique to sort numeric numbers in respect to reducing the time at operational level. NEET's algorithm is bidirectional as cocktail sort but dissimilarities are there in operational style and concept compared to bubble sort and cocktail sort. For measuring actual time that all three algorithms spend I have used MATLAB version R2007b. Bubble sort operates in one direction and cocktail sort adopt the mechanism of bubble sort but operates in respectively in two directions. Cocktail sort also named as bidirectional bubble sort, but due to its number of calculations in each pass it is not able to produce results on efficient time constraint. NEET's algorithm operates in two directions respectively by taking constant sets of two positions in array for each direction. NEET's performance is efficient on large number of elements comparing to bubble and cocktail as analyzed.

1. Introduction

In the real world as the technical support has merged in all area of profession the data has increased proportionally. There becomes a need to arrange the data in the specified incremented or decremented or any specified formulated manner. When the operation is performed to get this objective, then this process is called sorting of objects or sorting of elements.
NEET's is in-place algorithm. This algorithm operates on array elements in two directions. First direction of operating if the total number of elements (N) in array is even (N%2==0) is first position to last position in array that I have named left pass and the second direction is from second last position to second position in array named right pass. If N is odd (N%2! =0) then left pass runs by excluding last position and right pass runs from last position to second position in array. This algorithm creates sets for comparing the elements. Numbers of elements in sets are fixed with two elements in both left and right passes but order of sets are not same in both passes. Bubble sort and cocktail sort follow the incremental approach as 1, 2, 3….N-1, N for comparing the elements but NEET's follow the concept to partition the array in sets of two elements. NEET's algorithm provides a way that has smooth moves when it reaches from starting to last element in the left pass. As the control reaches at last element it does not make a jump to first or second element at the staring position, it just take a turn to the second last element that is the right pass. So as quick left pass ends, right pass gets started without the overhead of time to go back to first element. For small value of N this time is low but for larger N it may take effective time as in bubble sort, which is unnecessary increasing execution time. Creating sets of only two elements give advantage in time efficiency because set of three or greater than three elements produces extra time to sort with additional option that implementer may use another algorithm on these set of elements.

2. Concept of NEET's

2.1. Left and right pass

NEET's work in two passes left pass and right pass. The process is responsible to take a smooth move at the end of both left and right pass. Here is the concept of NEET's that improves the efficiency in respect to time to sort the numeric elements. Elements that are sorted in the order of sets in one pass (left pass) are again compared with elements of the set of another pass (right pass). It reduces the time of jumping at the end of each pass (left pass) and starts second pass (right pass). It means until all the elements get sorted the NEET's algorithm flows in left to right and right to left without any jump. Here I introduce the left and right pass:

2.1.1. Left pass. This pass is responsible for sorting the numeric elements from its first position to last position in the array. It is necessary to check that total number of elements (N) is even or odd. If N is even then left pass runs from first to last elements without excluding

any position but if N is odd that is N%2!=0 then left pass runs from first to second last element, excluding last position element. Sets are consistent relative to their position based on the N (even or odd) in NEET's algorithm.

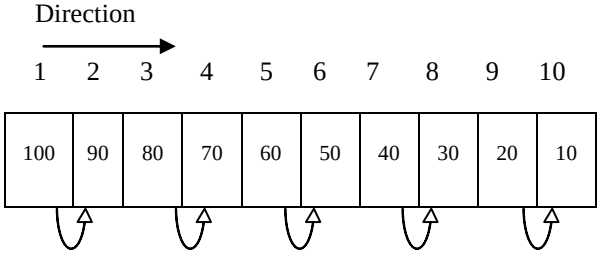

Figure 1. Comparison between elements when n is even in left pass

Sets in this scenario are created as group of two positions with even order as 1, 2 and 3, 4 and so on. -

Table 1. Sets of elements when n is even in left pass

Elements	Position
(100,90)	[1,2]
(80,70)	[3,4]
.	.
(20,10)	[N-1,N]

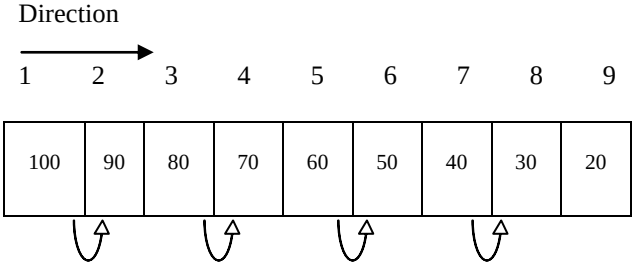

Figure 2. Comparisons between Elements When N is Odd in Left Pass

Sets in this scenario are created as shown in figure 2, the sets are same as figure1 but in odd sequence last element has excluded. Exclusion of element in one pass is handled in second pass. So the responsibility to sort the elements is not only on one pass, if one pass leave

Table 2. Sets of elements when n is odd in left pass

Elements	Position
(100,90)	[1,2]
(80,70)	[3,4]
.	.
(40,30)	[N-2,N-1]

specified position element in operation then second pass takes responsibility to include that previous pass excluded element in the sorting operation.

2.1.2. Right pass. In right pass the direction of process is from right to left in the array, in other words process executes to sort the elements from second last to second element in case of total number of elements N are even that is N%2==0. It is the effective pass of NEET's algorithm.

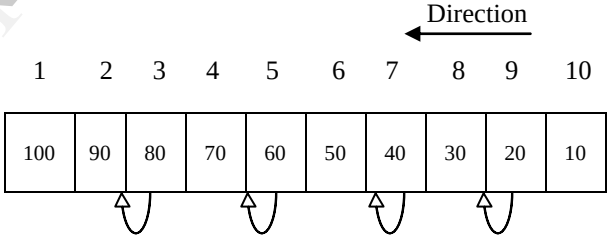

Figure 3. Comparison between elements when n is even in right pass

Sets in this scenario are created as-

Table 3. Sets of elements when n is even in right pass

Elements	Position
(20,30)	[9,8]
(40,50)	[7,6]
.	.
(80,90)	[3,2]

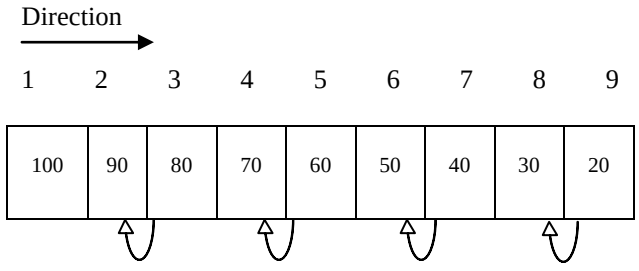

Figure 4. Comparisons between elements when n is odd in right pass

Sets in this scenario are created as-

Table 4. Sets of elements when n is odd in right pass

Elements	Position
(20,30)	[9,8]
(40,50)	[7,6]
. . .	. . .
(80,90)	[3,2]

So this right pass compares the elements that are recently replaced in left pass with the elements that get its position in another set just nearby of it. So shifting of elements in previous pass either in left or in right successively gets its right place during the operation.

3. Algorithm of NEET's sort, cocktail sort and bubble sort

3.1. NEET's algorithm

Step1. Set CountL=0, CountR=0, Total_Count=0, LS=1 and RE=2.

Step2. Input elements in array ARR.

Step3. N= Count (ARR).

Step4. If (N%2==0) then

LE= N and RS= N-1

Else

LE= N-1 and RS= N

Step5. Initialize Left Pass:

(i) For i= LS to LE

If (ARR[i] > ARR [i+1]) then

Swap (ARR[i], ARR [i+1])

CountL=CountL+1;

End If

i=i+2;

End For

Step6. Initialize Right Pass:

(i) For i= RS to RE

If (ARR[i] < ARR [i-1]) then

Swap (ARR[i], ARR [i-1])

CountR= CountR+1;

End If

i= i-2;

End For

Step7. Calculate Total_Count as:

Total_Count= CountL+CountR;

Step8. If (Total_Count! = 0) then

Repeat steps 5 to 7

Else

Go to Step 9

Step9.Exit

3.2. Cocktail sort algorithm

Step1. Enter elements in array ARR.

Step2. N=Count (ARR)

Step3. For i= 0 to (n/2) – 1

(i) Set swapped=false

(ii) For j= i to N-i-1

If (ARR[j] > ARR [J+1]) then

Swap (ARR[j], ARR [j+1])
Swapped=true

j= j+1

End If

End For

(iii) For j= n-2-i to j>i

If (ARR[j] < ARR [j-1]) then

Swapped= true

j=j-1

End If

End For

Step4. If (Swapped) then

Repeat Steps 3

Else

Go to Step 5

Step5. Exit

3.3. Bubble sort algorithm

Step1. Enter elements in the array ARR.

Step2. Calculate N= Count (ARR)

Step3. For i= 1 to N-1, repeat step 4

Step4. For j=0 to N-i, repeat

if ARR[j] > ARR[j+1] then

temp:= ARR[j]

ARR[j]:= ARR[j+1]

ARR[j+1]:= temp

End if

End For (Step4)

End For (Step3)

Step5. Exit

4. Performance analysis

To check the performance of NEET's compared to bubble sort and cocktail sort I have used set of numbers ranging from 500 to 4000. The performance is analyzed in worst case and average case. Also why MATLAB is used for analysis is because of its nature of operation and its functioning features.

4.1. About MATLAB R2007b

MATLAB (R2007b) provides a high-level language and development tools that let you quickly develop and analyze algorithms and applications. The MATLAB language provides native support for the vector and matrix operations that are fundamental to solving engineering and scientific problems, enabling fast development and execution.

MATLAB (R2007b) uses processor-optimized libraries for fast execution of matrix and vector computations. For general-purpose scalar computations, MATLAB uses its just-in-time (JIT) compilation technology to provide execution speeds that rival those of traditional programming languages. All the comparison is performed between cocktail and NEET's by taking smaller to larger number of elements.

4.2. System platform

It is important that on which hardware and software configured platform performance have been checked. Here in analysis of all three algorithms the system used with its configuration is described in table 5.

Table 5. System configuration

Components	Details
Processor	Intel(R) Core(TM) i3 CPU M 370 @ 2.40GHz 6.7 4.3
RAM	3.00 GB 5.5
Primary Hard Disk	283 GB Total
Operating System	Windows 7 Home Basic
Manufacturer	Hewlett-Packard
Model	HP G42 Notebook PC
System Type	64-bit Operating System
Number of Processor Cores	2

4.3. Performance on worst and average case

Here we go for two sections for performance check, worst case and average case. The term worst case is used, when approximately all the numbers need to be operated. As in sorting, worst case is the section of operation where all elements need to be placed at their right position that is changing array elements order from completely descending order to ascending order. The average case is where approximately half of the total number of elements that is N/2 needs to be placed at their correct position. So here I have checked on different sets of numbers in both cases ranging 500 to 4000 numbers.

4.3.1. Worst case time of NEET's, cocktail and bubble sort.

Table 6. Time spent to sort in worst case

Total Number of Elements (N)	Time in Second		
	NEET's	Cocktail	Bubble
500	0.0071	0.2256	0.2036
1000	0.0185	0.8822	0.826
1500	0.0409	1.931	1.7995
2000	0.0676	3.3995	3.1659
2500	0.1041	5.3328	4.9335
3000	0.1523	7.578	7.096
4000	0.2607	13.2179	12.4093

As listed in table 6 the worst case performance of NEET is much higher compared to both bubble and cocktail sort. As calculative results NEETs is 28.676 times and 31.77465 times approx. higher then bubble and cocktail sort respectively on 500 numbers in worst case.

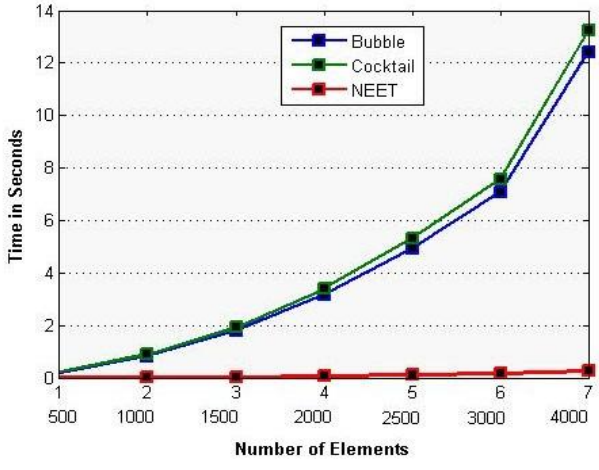

Figure 5. Time spent to sort in worst case up to 4000 numbers

Also as the number of elements increases this ratio of result increases as also shown in figure. From 1000 to 4000 numbers in worst case NEET's performs approx. 46 times faster than bubble sort and approx. 49 to 50 times faster than cocktail. So negation in sorting of elements time is achieved at its best level compared to both sorting techniques by NEET's.

Both bubble and cocktail sort on average performs same but NEET's performance is much higher compared to both. Figure 5 shows if complete descending numbers need to be in ascending order then based on time constraint NEET's replaces bubble and cocktail sort.

4.3.2. Average case time of NEET's, cocktail and bubble sort. Any algorithm performance can be predictable based on the worst case performance. If algorithm performance is high enough means in small amount of time algorithm produces desired output then in average case it will perform well. As listed in table 7 the time sequence of bubble sort and cocktail sort algorithms shows good performance measure compared to worst case.

Table 7. Time spent to sort in average case

Total Number of Elements (N)	Time in Second		
	NEET's	Cocktail	Bubble
500	0.0053	0.1177	0.1079
1000	0.0199	0.4389	0.4415
1500	0.0327	0.9597	0.9461
2000	0.0564	1.6986	1.6936
2500	0.0837	2.6261	2.6033
3000	0.1192	3.6988	3.709
4000	0.2101	6.5409	6.5676

As listed in table 7 for 500 to 4000 numbers all three NEET's, cocktail and bubble sort algorithms reduced their time constraint in average case. But cocktail and bubble sort have not achieved at better level their performance compared to NEET's.

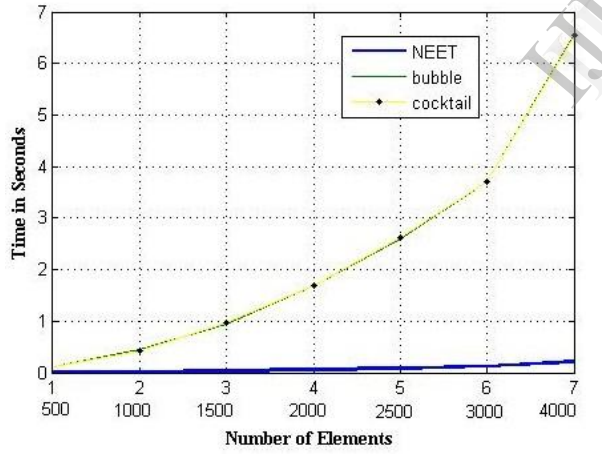

Figure 6. Time spent to sort in average case up to 4000 elements

In the figure 6 bubble sort and cocktail sort performs on same time scale. Bubble sort's green lines in figure following the same time sequence and fluctuation in time as the number of elements exceed. For industrial purpose where bubble and cocktail is used NEET's can give best output to sort numeric data. If we see the fluctuation in time for NEET's in figure 6, it show that there is not a big jump as in bubble sort and cocktail sort from 3000 to 4000 numbers. Compared to bubble and cocktail sort, NEET's algorithm is approx. 20 times faster to sort 500 numbers in average case. For 2000 numbers NEET's takes 33% (approx.) time on time of bubble and cocktail sort. As to take final set of 4000 random numbers bubble and cocktail sort produce output approx. 31 times slower than NEET's.

5. Conclusion and future work

NEET's concept is simple enough to be programmed in any language for students' purpose to industrial purpose where bubble and cocktail sorts are used. Performance of NEET's is best comparing to both algorithms. Average case and worst case performance show that there is a great time difference to be efficient compared to NEET's of both bubble and cocktail sorts. Simple and division of operation in right and left pass of NEET's gives flexibility for testing that what are the new elements get placed after run of the current (left or right) pass. Objective of NEET's algorithm is to reduce operating time of elements to place them at their right place has achieved at its best level compared to bubble sort and cocktail sort.
Related to domain of time constraint NEET's performs well. Improvement in NEET's algorithm will be proposed in future, by adding advanced and intellectual methodology.

6. References

[1] Programming and Algorithm Development -MATLAB–MathWorksIndia, http://www.mathworks.in/products/matlab/description4.html
[2] Vandana Sharma, Satwinder Singh and Dr. K. S. Kahlon, "Performance Study of Improved Heap Sort Algorithm and Other Sorting Algorithms on Different Platforms", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.4, April 2008
[3] Dr. Leena Bhatia, Dr. Bindu Jain, "Ash Sorting: Easy & Less Time Consuming Sorting Algorithm", IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.12, December 2010
[4] James D. Fix and Richard E. Ladner, Member, IEEE, "Sorting by Parallel Insertion on a One-Dimensional Subbus Array", IEEE TRANSACTIONS ON COMPUTERS, VOL. 47, NO. 11, NOVEMBER 1998
[5] Stephan Olariu, Member, IEEE, M. Cristina Pinotti, Member, IEEE Computer Society, and Si Qing Zheng, Senior Member, IEEE, "An Optimal Hardware-Algorithm for Sorting Using a Fixed-Size Parallel Sorting Device", IEEE TRANSACTIONS ON COMPUTERS, VOL. 49, NO. 12, DECEMBER 2000

[6] Dr. Anupam Shukla and Rahul Kala, "Predictive Sort", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.6, June 2008
[7] Dr. Vipin Saxena, Ajay Pratap, "Genetic Algorithm Based Bayesian Classification Algorithm for Object Oriented Data", IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS), ISSN: 2249-9555, Vol. 2, No.6, December 2012
[8] http://rosettacode.org/wiki/Sorting_algorithms

Author Profile

Ayush Kumar Yogi has received BCA degree in 2008 from University of Kota and MCA from Rajasthan Technical University in 2011. At present he is faculty of MCA Department, Govt. Engineering College, Ajmer, Rajasthan.