International Journal of Computer Science and Network (IJCSN)
Volume 1, Issue 1, February 2012 www.ijcsn.org ISSN 2277-5420

# A Statistical Analysis of Bubble Sort in terms of Serial and Parallel Computation

[1]Sunil Kumar Panigrahi, [2]Dr. Soubhik Chakraborty, [3]Dr. jibitesh Mishra

[1] Dept of Computer Science and Engineering, APEX Institute of Technology & Management, Pahal, Bhubaneswar-752101, Orissa, India

[2] Department of Applied Mathematics, Birla Institute of Technology, Mesra, Ranchi-835215, Jharkhand, India.

[3] Head of Department of Information Technology College of Engg. Technology, Ghatika Bhubaneswar, Orissa, INDIA

**Abstract**
In some recent papers, the weight based statistical bounds have arguably explained time complexity better than the count based mathematical bounds. This is definitely true for average case where for an arbitrary code it is difficult to identify the pivotal operation or pivotal region in the code for taking the expectation and/or when the probability distribution, over which expectation is taken, becomes unrealistic over the problem domain. In worst case, it can certify whether a mathematical bound is conservative or not. Here we revisit the results on Bubble sort in sequential mode and make an independent study of the same algorithm in parallel mode using statistical bound

***Keywords:*** *Weight, Statistical bound, Complexity, Pivotal reason, Worst case.*

## 1. Introduction

Not Everything that can be counted counts and not everything that counts can be counted – **George Gallop**

The American Heritage Dictionary defines “analysis” as “*The separation of an intellectual or substantial whole into its constituent parts for individual study*”. But the term “analysis of algorithms” is usually used in a narrow technical sense to mean an investigation of algorithm efficiency with respect to two resources: running time and memory space. There are various ways to find the complexity of an algorithm. The mathematical bound is not sufficient to find the complexity of an algorithm, because there are several factors that affect complexity of an algorithm as given in Table-1. The statistical bound is best suitable to find the complexity of an algorithm, as it gives weights to computing operations rather than merely counting them. The mathematical bounds are operation specific. In a statistical bound, the operations can be collectively taken due to their weighing. This is very realistic as in actual implementation; the operations would be, after all, performing collectively.

Despite many years of intensive study, average case complexity of algorithms, has always been intriguing for computer scientists. For an arbitrary code it is difficult to identify the pivotal operation or pivotal region in the code for taking the expectation and/or it may happen that the probability distribution, over which expectation is taken, becomes unrealistic over the problem domain. On the other hand, average case complexity is an important area of research given that several algorithms with bad worst case complexity like Quick Sort perform better on an average. It is time to study the different cases of algorithm complexity, so that a better conclusion can be drawn. Following is a list of literature review on average case complexity:-

Chakraborty and Choudhury (2000) proposed the use of time as a weight. Chakraborty and Sundararajan (2007) experimented with Winograd’s algorithm with this philosophy. Chakraborty and Sourabh showed an empirical O($n^2$) complexity is gettable with two dense matrices in Amir Schoor’s algorithm in square matrix multiplication. The idea of statistical bound and the merging of two disciplines of research, algorithmic complexity and computer experiments have been recently documented in the book by Chakraborty and Sourabh (2010). For a general overview on average complexity, we consulted the works of Bogdanovl and Trevisan (2006), Vitanyi (2002), Impaggliazzo(1995), Wang (1992), Gurevich (1991), Li and Vitanyi (1992), David et. al. (1989). That statistical bound is ideal in average case complexity is strongly assessed in Chakraborty, Hatwal and Rastogi (2008). For worst case complexity, we consulted Liu and Lee (1994). For simulation works and computer experiments, we consulted Kennedy and Gentle (1980) and Sacks et. al. (1989) respectively.

The CPU time in complexity analysis of algorithms can be taken as a weight or in other words the weight of a computing operation is the corresponding time it consumes. Since the statistical approach is ideal in average cases, in the present paper, we examine the complexity of sort in terms of statistical bound in serial computing which is a survey of our previous work with weights and extend our approach to parallel computing where, we argue, with every change of processor or other factors of parallel computing (like scalability of the system), it is the weight of the operation that changes. Hence if the bound itself is based on weights, we are done.

In this paper Section-2 contain the comparison between statistical and mathematical bound and defining of statistical bound. Section-3 finding of statistical bound for a sort algorithm by conceptually. And finding the statistical bound for parallel computing using some of the factors describe in section-4.

## 2. Comparisons of Statistical & mathematical Bounds

Some of the common factors that are influenced during Empirical Estimation, table-1 contains some of the factors that are ideal for time complexity in mathematical bound.

Table 1: Ideal for all the problem cases

| Factors | Example |
|---|---|
| Instructions are not distinguished | i. Sum=a+b<br>ii. If(a>b)<br>both instructions are counted equally |
| Selection statements counting are difficult | If(a>b) Stat-1<br>Whether the Stat-1 is counted. |
| System with different speed | Giga Hertz processors are faster than Mega Hertz processors |
| Program environment | Different programming language compiler like Java, C etc. |
| Different values | Multiple variable having values |
| Data size are not consider | The variable contain n=10 is not same as n=10000000 |
| Data type | The variable is Integer or float or double etc. |
| Data transfer rate | Form processor to memory or vice-versa |
| Size of memory | In MB or GB |
| Data arrangement | N= 1, 2, 3, 4, 5<br>N= 5, 4, 3, 2, 1<br>N= 3, 4, 2, 5, 1<br>Different arrangement of data |
| No of time memory reference | Cache hit or miss |
| Data input by user | Based on user typing speed |
| Outputted data to device | Whether it send to monitor or printer or any other device which far from CPU |
| Types of OS | UNIX or Windows or any other |
| Ideal for multi programming | If simultaneously many programs run in same environment |

### 2.1 Definition of Statistical bound ($O_{Stat}$)

*If $w_{ij}$ is the weight of (a computing) operation of type i in the j-th repetition (generally time is taken as a weight) and y is a "stochastic realization" (Pwhich may not be stochastic [4] [9]) of the deterministic*

$$T = 1.\ w_{ij}$$

*where we count one for each operation repetition irrespective of the type, the statistical bound of the algorithm is the asymptotic least upper bound of y expressed as a function of n where n is the input parameter characterizing the algorithm's input size. If interpreter is used, the measured time will involve both the translation time and the execution time but the translation time being independent of the input will not affect the order of complexity. The deterministic model in that case is*

$$T = P\ 1.\ w_{ij} + \text{translation time.}$$

*For parallel computing, summation should be replaced by maximum. Empirical O (written as O with a subscript emp) is an empirical estimate of the statistical bound over a finite range, obtained by supplying numerical values to the weights, which emerge from computer experiments.*

For more insight, the performance factors as per the Kai Hwang [1]. Let $I_c$ be the number of instructions in a given program, or the instruction count. The CPU time (T in seconds/program) needed to execute the program is estimated by finding the product of three contributing factors

$$\mathbf{T = I_c * CPI * \tau}$$

Where $\tau$ clock time of a processor, the execution of an instruction requires going through a cycle of events involving the instruction fetch, decode, operand(s) fetch execution and store results. In this cycle, only the instructions decode and execution phases are carried out in the CPU. CPI (Cycle Per Instruction) is varies from instruction to instruction. The CPI of an instruction type can be divided into two component terms corresponding to the total processor cycles and memory cycles needed to complete the execution of the instruction. Depending on the instruction type, the complete instruction cycle may involve one to four memory references

$$\mathbf{T = I_c * (\mathit{P} + \mathit{M} * \mathit{K}) * ج}$$

Where $P$ is the number of processor cycles needed for the instruction decoded and execution, $M$ is the number of memory references needed, $K$ is the ratio between memory cycle and processor cycle.

Now for analytically comparing statistical bound and mathematical bound, Let ج be Constant for the both bounding, then

Empirical 'O'

$$\mathbf{T_{Emp} = I_c} \text{ (No of Instruction that executed)}$$

For Mathematical bound CPI is 1,

Statistical 'O'

But for Statistical

$$\mathbf{T_{Stat} = I_c * CPI}$$

For Statistical bound CPI becomes the Weight ($W_{ij}$)

$$\mathbf{T_{Stat} = I_c * W_{ij}}$$

In general if $W_{ij}$=1, Then

$$\mathbf{T_{Emp} = T_{Stat}}$$

$W_{ij}$ vary from algorithm to algorithm and chosen from the concept that based on how the problem is solved and some other factors, the value of $W_{ij} \geq 1$ . In algorithm the lowest or less time taken for execution of an instruction is to be taken as 1. The table-2 shows some of the basic difference between mathematical and statistical bound [7]. The experiment carried over P-IV machine with Window2007 Operating System and DEV 'C' compiler.

Table2: Comparison between Mathematical and statistical bounds

| **Mathematical Bounds** | **Statistical Bounds** |
|---|---|
| Operations are counted. | Operations are Weighted. |
| A separate bound is provided for a specific operation type | Weighting permits collective considerations for determining the bound |
| Theoretical derivable | They are conceptual |
| They may be unrealistic at times | Guaranteed to be realistic |
| They are system independent. | They can only system invariant |
| They are ideal for analyzing worst case behavior (as the bounds have guarantee) | They are suitable for average case |
| They are exact | They are exact provided only they are system invariant |

N.B. See Chakraborty and Sourabh (2010) for an insight.

## 3. Bubble Sort in Statistical bound

A simple introductory example of the application of the incompressibility method is the average-case analysis of sorting algorithm. The classical approach can be found in [11]. It is well-known that Bubble sort uses $O(n^2)$ comparisons/exchanges on the average. We present a very simple proof of this fact. The proof is based on the following intuitive idea: There are $n!$ different permutations. Given the sorting process (the insertion paths in the right order) one can recover the correct permutation from the sorted list. Hence one requires $n!$ Pair wise different sorting processes. This gives a lower bound on the minimum of the maximal length of a process. We formulate the proof in the crisp format of incompressibility.

In a comparisons and swapping sort we make passes from left to right over the permutation to be sorted and always move the currently largest element right by exchanges between it and the right-adjacent element, if that one is smaller. We make at most $n$ ¡ 1 passes, since after moving all but one element in the correct place the single remaining element must be also in its correct place (it takes two elements to be wrongly placed). The total number of exchanges is obviously at most $n^2$, so we only need to consider the lower bound. We can describe the total number of exchanges by it's initial distance of element $n$. Note that in every pass more than one element may \bubble" right but that means simply that in the future passes of the sorting process an equal number of exchanges will be saved for the element to reach its position. That is, every element executes a number of exchanges going right that equals precisely the initial distance between its start positions to its position. The model suggested for how to evaluate the statistical bound for an algorithm. The figure-1 consists algorithm and weight that combine to form the Statistical bound. The weight is conceptually built as fit to algorithm.

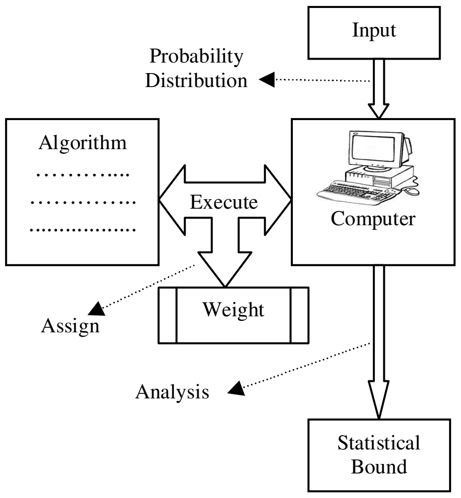

Fig. 1 Model of the Statistical Bounds

Here we are considering for bubble sort to find the complexity by using above model, and its experimental result.

## 3.1 Analysis of Sorting Algorithm in Serial Computing using statistical bound's

```
1.  ALGORITHM Sort(A[ ],n)
2.      FOR ( i = 1 to n - 1 and i = i + 1) DO
3.      Count=count+1
4.          FOR ( j = 1 to n - i and j = j + 1) DO
5.          Count=count+1
6.              Compare
7.                  Swap
8.  END
```

Now let us consider for n=10 elements of array. For each shuffling positions best case, worst case and average cases are derived as shown in the table-3.

Table3: Different cases of Mathematical bound for n=10

| Comparisons | Loop | swapping | | | Total Time | | |
|---|---|---|---|---|---|---|---|
| For all cases | For all Cases | best | worst | average | best | worst | average |
| 45 | 110 | 0 | 0 | 0 | 155 | 155 | 155 |
| 45 | 110 | 1 | 17 | 7 | 158 | 206 | 176 |
| 45 | 110 | 2 | 16 | 11 | 161 | 203 | 188 |
| 45 | 110 | 2 | 30 | 15 | 161 | 245 | 200 |
| 45 | 110 | 3 | 29 | 19 | 164 | 242 | 212 |
| 45 | 110 | 3 | 39 | 22 | 164 | 272 | 221 |
| 45 | 110 | 4 | 38 | 27 | 167 | 269 | 236 |
| 45 | 110 | 4 | 44 | 31 | 167 | 287 | 248 |
| 45 | 110 | 5 | 43 | 35 | 170 | 284 | 260 |
| 45 | 110 | 5 | 45 | 39 | 170 | 290 | 272 |

Complexity for the above sorting algorithm, where n=10. The numbers of comparisons are 45 in all cases and swapping is 0 to 45 and the looping for all cases are 110 but the total time for the sort algorithm is 155 to 290. And the average case is towards the worst case. The average case complexity of sorting algorithm is $O(n^2)$.

For n=10 As per the mathematical bound time complexity $\approx O(n^2)$ As per the experimental from above table-3, we conclude that

For comparison or swapping the complexity = $45 \approx n^{1.653213} \approx n^{1.65}$

For looping = $110 \approx n^{2.041393} \approx n^2$

For Best case = $155 \approx n^{2.190332} \approx n^{2.20}$

For Worst case = $290 \approx n^{2.462398} \approx n^{2.46}$

For Average case = $272 \approx n^{2.434569} \approx n^{2.43}$

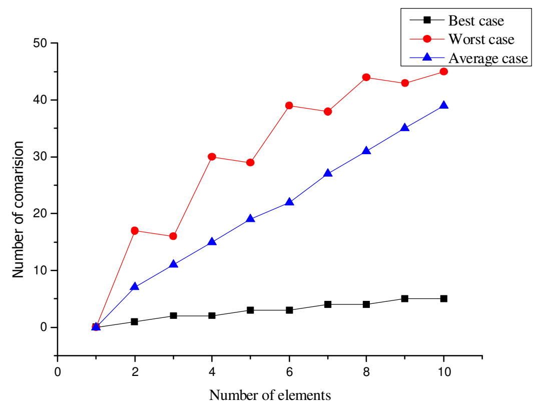

Fig. 2. Shows the number of time swapping for 10 different elements in different shuffling, and its best, worst, and average cases

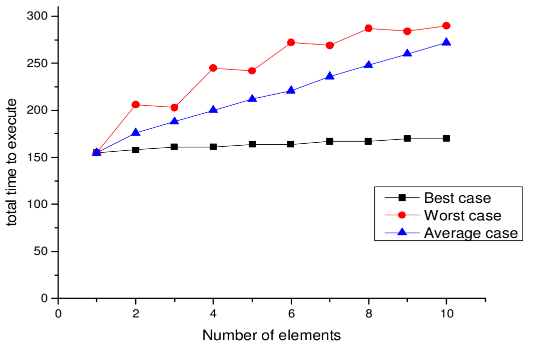

Fig3. Shows the number of time total execution of instructions for 10 different elements in different shuffling, and its best, worst, average cases

The figure-2 and figure-3, shows the graphical representation of the complexity of bubble sort algorithms and average case are more towards the worst case.

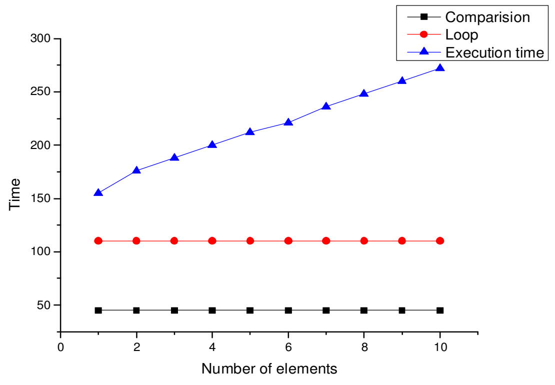

Fig4. Shows the number of time total execution of instructions for 10 different elements in the loops, comparisons and total execution time.

But if we consider about the comparison or swapping then it is less than the mathematical bounds. Figure-4 clearly distinguishes between the loops, comparisons and total execution time. So the complexity of bubble sort algorithm in statistical bound is O($n^{2.46}$), when we apply the counting as the weight.

But average is a statistical term, and the sorting algorithms are depending upon on the following three pivotal regions.

1. Comparison
2. Swapping
3. Number of time the loop conditions checked and incremented

In sorting algorithm we are consider the weight as comparisons, swapping and the loop executed. So there are three weighting factor as follows.

$$T_{stat} = W_1\, I_{Compare} + W_2\, I_{Swapping} + W_3\, I_{Looping}$$

$W_1$ = The instruction is used for the Comparisons.
$W_2$ = The instruction used for swapping
$W_3$ = The instructions are used for the loop condition checked and incremented

In statistical bound the complexity of algorithms is measured by experimentally i.e. here the weight is consider for time required to execute the pivotal factors. The algorithm is tested over the system configuration such as

```
1.  ALGORITHM Sort(A[ ],n)
2.      W3 = start-timer
3.      FOR ( i = 1 to n - 1 and i = i + 1) DO
4.          FOR ( j = 1 to n - i and j = j + 1) DO
5.              W1 = start-timer
6.              Compare
7.                  W2 = start-timer
8.                  Swap
9.                  W2 = stop-timer
10.             W1 = stop-timer
11.     W3 = stop-timer
```

Time efficiency estimates depend on what we define to be a step. For the analysis to correspond usefully to the actual execution time, the time required to perform a step must be guaranteed to be bounded above by a constant. One must be careful here; for instance, some analyses count an addition of two numbers as one step. This assumption may not be warranted in certain contexts. For example, if the numbers involved in a computation may be arbitrarily large, the time required by a single addition can no longer be assumed to be constant.

In table-4 the result that obtained by the algorithm is compare with some power of 'n' and we are consider for the elements form n=1 to 30 and its result. Mathematical bound is suitable for, when n is small.

Table 4. Different cases of statistical bound for n=1 to 30

| Worst **case** | | | | | |
|---|---|---|---|---|---|
| | Compa risons | Total time | O($n^2$) | Total time in power of n | Comparison in power of n |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 2 | 1 | 10 | 4 | 3.321928 | 0 |
| 3 | 3 | 24 | 9 | 2.892789 | 1 |
| 4 | 6 | 44 | 16 | 2.729716 | 1.292481 |
| 5 | 10 | 70 | 25 | 2.639739 | 1.430677 |
| 6 | 15 | 102 | 36 | 2.581246 | 1.511392 |
| 7 | 21 | 140 | 49 | 2.539502 | 1.564575 |
| 8 | 28 | 184 | 64 | 2.507854 | 1.602452 |
| 9 | 36 | 234 | 81 | 2.482824 | 1.63093 |
| 10 | 45 | 290 | 100 | 2.462398 | 1.653213 |
| 11 | 55 | 352 | 121 | 2.445324 | 1.671188 |
| 12 | 66 | 420 | 144 | 2.430777 | 1.686041 |
| 13 | 78 | 494 | 169 | 2.41819 | 1.698555 |
| 14 | 91 | 574 | 196 | 2.407159 | 1.709269 |
| 15 | 105 | 660 | 225 | 2.397385 | 1.718565 |
| 16 | 120 | 752 | 256 | 2.388647 | 1.726723 |
| 17 | 136 | 850 | 289 | 2.380772 | 1.733952 |
| 18 | 153 | 954 | 324 | 2.373627 | 1.740412 |
| 19 | 171 | 1064 | 361 | 2.367103 | 1.746229 |
| 20 | 190 | 1180 | 400 | 2.361115 | 1.7515 |
| 21 | 210 | 1302 | 441 | 2.355593 | 1.756304 |
| 22 | 231 | 1430 | 484 | 2.350479 | 1.760706 |
| 23 | 253 | 1564 | 529 | 2.345723 | 1.764758 |
| 24 | 276 | 1704 | 576 | 2.341286 | 1.768504 |
| 25 | 300 | 1850 | 625 | 2.337133 | 1.77198 |
| 26 | 325 | 2002 | 676 | 2.333234 | 1.775216 |
| 27 | 351 | 2160 | 729 | 2.329564 | 1.778239 |
| 28 | 378 | 2324 | 784 | 2.326101 | 1.781071 |
| 29 | 406 | 2494 | 841 | 2.322826 | 1.783732 |
| 30 | 435 | 2670 | 900 | 2.319722 | 1.786237 |

It is clear that from the above table the total time of sort in statistical bound is gradually decrease and the number of comparision is increased. As shown on figure-4 that the total no of instruction executed is gradually decreased to wards $n^2$ .

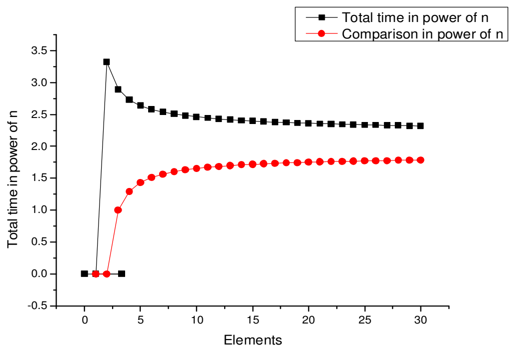

Fig. 5. It shows the number of time total execution of instructions and comparisons for 1 to 30 elements in polynomial time.

From the above figure-5 it is found that the complexity of the sort algorithm comparisons/swapping is less than from mathematical bound and the total no instruction executed is greater than the mathematical bound, for the short range of element

**Comparisons/swapping < mathematical bound < total no instruction executed**

Now we are considering for the larger elements and in terms of time in mili second. Time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform. Thus the amount of time taken and the number of elementary operations performed by the algorithm differ by at most a constant factor.

Since an algorithm may take a different amount of time even on inputs of the same size, the most commonly used measure of time complexity, the worst-case time complexity of an algorithm, denoted as ***T*(*n*)**, is the maximum amount of time taken on any input of size *n*. Time complexities are classified by the nature of the function *T*(*n*). For instance, an algorithm with *T*(*n*) = O(*n*) is called a linear time algorithm, and an algorithm with *T*(*n*) = O($n^2$) is said to be an exponential time algorithm.

Table 5. Different cases of statistical bound for n in medium range

| | Best case | | | Worst case | | |
|---|---|---|---|---|---|---|
| Elements | Min | Max | Average | Min | Max | Average |
| 1000 | 0 | 0 | 0 | 6 | 11 | 8.5 |
| 2000 | 0 | 0 | 0 | 15 | 17 | 16 |
| 3000 | 15 | 30 | 22.5 | 31 | 47 | 39 |
| 4000 | 31 | 40 | 35.5 | 47 | 62 | 54.5 |
| 5000 | 35 | 47 | 41 | 78 | 94 | 86 |
| 6000 | 63 | 79 | 71 | 105 | 119 | 112 |
| 7000 | 78 | 91 | 84.5 | 140 | 156 | 148 |
| 8000 | 93 | 109 | 101 | 187 | 203 | 195 |
| 9000 | 122 | 138 | 130 | 234 | 250 | 242 |
| 10000 | 150 | 166 | 158 | 280 | 297 | 288.5 |

Table-5 contain the minimum to maximum rang value in terms of mili second and Figure-6 shows the different cases for the sorting algorithm and its graphical representation for range value 1000 to 10000. And it shows the polynomial bound.

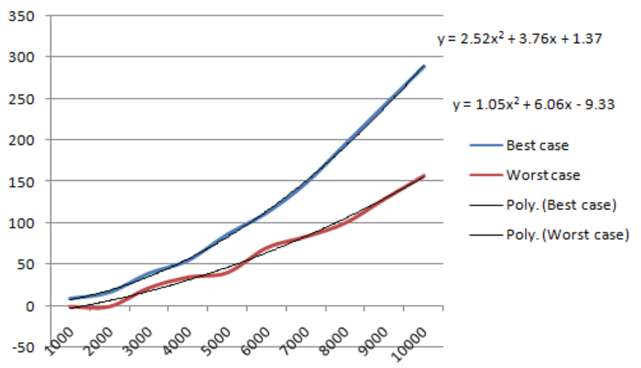

Fig. 6. It shows the number of time total execution of instructions for the 1000 to 10000 elements in its best, worst cases and finding of its best case, worst case polynomial time.

Now apply the elements 10000 to 100000 on sort algorithm and analysis using statistical bounding. The table-6 shows the experimental result of sorting elements in mili second and its best case, worst case analysis, and figure-7 shows the graphical representation of the table-6 result.

Table 6. Different cases of statistical bound for large n

| Elements | Best case | | | Worst case | | |
|---|---|---|---|---|---|---|
| | Min | Max | Average | Min | Max | Average |
| 10000 | 150 | 166 | 158 | 280 | 297 | 288.5 |
| 20000 | 555 | 576 | 565.5 | 1123 | 1149 | 1136 |
| 25000 | 858 | 886 | 872 | 1732 | 1757 | 1744.5 |
| 30000 | 1233 | 1258 | 1245.5 | 2496 | 2532 | 2514 |
| 40000 | 2190 | 2210 | 2200 | 4430 | 4466 | 4448 |
| 50000 | 3417 | 3452 | 3434.5 | 6911 | 6956 | 6933.5 |
| 60000 | 4930 | 4956 | 4943 | 9952 | 9998 | 9975 |
| 70000 | 6692 | 6721 | 6706.5 | 13556 | 13592 | 13574 |
| 80000 | 8751 | 8778 | 8764.5 | 17690 | 17742 | 17716 |
| 90000 | 11076 | 11107 | 11091.5 | 23057 | 24005 | 23531 |
| 100000 | 13674 | 13699 | 13686.5 | 27705 | 27796 | 27750.5 |

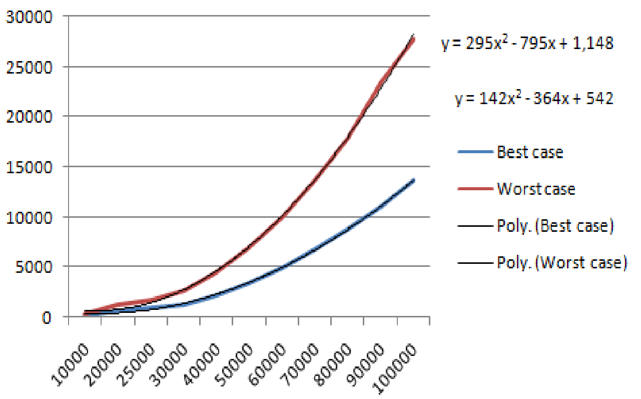

Fig. 7. It shows the number of time total execution of instructions for the 10000 to 100000 elements in its best, worst cases and finding of its best case, worst case polynomial time.

From the above experimental result it conclude that when N is large the complexity of sorting algorithm is polynomials bounded to $N^{2.x}$ where 'x' value is gradually decrease when N is increased.

$$T_{Stat} \approx O_{stat}(295\ N^2) \approx O_{stat}(N^{2.4})$$

## 4. Factors for bubble sort in parallel mode

A *parallel computer* is a set of processors that are able to work cooperatively to solve a computational problem. This program may access local memory and may send and receive messages over the network. Messages are used to communicate with other computers or, equivalently, to read and write remote memories. In the idealized network, the cost of sending a message between two nodes is independent of both node location and other network traffic, but does depend on message length.

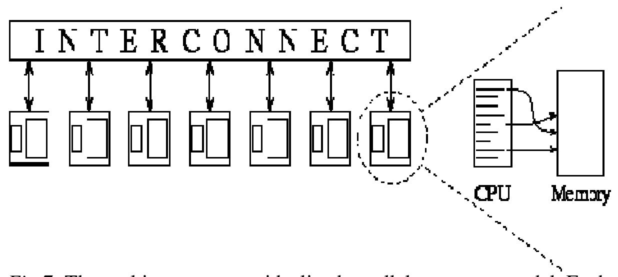

*Fig 7:* The multicomputer, an idealized parallel computer model. Each node consists of a von Neumann machine: a CPU and memory.

*Message passing.* Message passing is probably the most widely used parallel programming model today. Message-passing programs, like task/channel programs, create multiple tasks, with each task encapsulating local data. Computational algorithms are traditionally executed sequentially on uni-processors. Parallel algorithms are those specially devised for parallel computers. The idealized parallel algorithms are those written for the PRAM models if no physical constraints or communication overheads are imposed. Some characteristics direct affect to parallelism

1. Deterministic versus nondeterministic
2. Computational granularity
3. Parallelism profile
4. Communication pattern and synchronization requirements
5. Uniformity of the operations
6. Memory requirements
7. Data structures

Some factors that indirect affect

1. Machine size
2. Clock rate
3. Problem size
4. CPU time
5. I/O bound
6. Memory capacity
    a. Shared memory
    b. Distributed memory
7. Communication overhead
8. Programming overhead
9. Scalability
10. Architecture
    a. Symmetric
    b. Asymmetric
11. Dependency
    a. Data dependency
    b. Control dependency
12. Parallel programming models
    a. Shared variable
    b. Message passing
    c. Data parallel
    d. Object-oriented
    e. Functional & logical
13. Complier used
    a. Implicit
    b. explicit

Basically the time factor in parallel mode is evaluated by determining by number processor that used for solving the problem. The bubble sort in parallel mode is as follows and its statistical bound.

```
1.  ALGORITHM Parallel_Bubble_Sort ( A[ ], n )
2.  BEGIN
3.      INITIALISE the number of processor P_i //
            may be 'n' number of processor
4.      SEND values to P_i // each contain an element
            of A[ ]
5.      counter = counter + n // as to communicate
            all thye processor
6.      FOR ( i = 1 to n and i= i+1) DO
7.      BEGIN
8.          counter1 = counter1 + 1 // number of
                time loop is executed.
9.          IF ( i % 2 == 1) THEN
10.         BEGIN
11.             Communicate to all odd P_i
```

```
12.          counter = counter + n / 2 //
             communicate to only odd processor to
             initiate
13.                IF (P_i == odd) THEN
14.          IF ( P_i > P_i+1 ) THEN
15.          counter2 = counter2 + n / 2 // only
             odd processors to compare
16.          EXCHANGE ( P_i, P_i+1 )
17.          END
18.          ELSE
19.          BEGIN
20.          Communicate to all even P_i
21.          counter = counter + n / 2 //
             communicate to only even processor to
             initiate
22.          IF (P_i == even ) THEN
23.          IF ( P_i > P_i+1 ) THEN
24.          counter2 = counter2 + n / 2 // only
             even processors to compare
25.          EXCHANGE ( P_i, P_i+1 )
26.          END
27.      END
28.      RECEIVE all the values from P_i
29.      counter = counter + n // as to receive from all
             the processor
30. END
```

$T_{stat}$ = Number of time the loop executed + Number of time it communicate to other processor + number of time it compare and swapping in single execution.

$$\mathbf{T_{stat}} = W_1 \, \mathbf{I_{Compare}} + W_2 \, \mathbf{I_{Communication}} + W_3 \, \mathbf{I_{Looping}}$$

$W_1$ = The 14th number instruction is used for the Comparisons.

$W_2$ = The instruction 4th, 11th, 20th and 24th are used for communication

$W_3$ = The 6th number instructions used for the loop condition checked and incremented

Time required to execute $\mathbf{T_{stat} = n / p_i + 3n + n / p_i}$. If number of processor is same as the number of elements in array and communication overhead is neglected then Conceptually the statistical bound of bubble sort in parallel algorithm is 'n'

## 5. Conclusion

In parallel computing, with every change of the processor, the weight of an operation changes. Thus the ideal bound here should be a weight based statistical bound and not a count based mathematical bound. Our case study in bubble sort substantiates this fundamental argument.

## References

[1] Kai Hwang, *Advanced Computer Architecture* Tata-McGrawHill p.14-16.

[2] S. Chakraborty, S. K. Sourabh , *On Why an Algorithmic Time Complexity Measure can be System Invariant rather than System Independent*, Applied Math. and Computation, Science Direct, Elsevier Vol. 190(1), 2007, p. 195-204.

[3] S. Chakraborty, S. K. Sourabh , *How robust are average complexity measures? A statistical case study*, Applied Math. and Computation, Science Direct, Elsevier Vol. 189(1), 2007, p. 1787-1797.

[4] S. Chakraborty, P. P. Choudhury, *A Statistical Analysis of an Algorithm's Complexity*, Applied Mathematics Letters 13(5), 2000, p.121-126.

[5] S. Chakraborty, S. K. Sourabh, A Computer Experiment Oriented Approach to Algorithmic Complexity, Lambert Academic Publishing, 2010.

[6] S. Chakraborty, Charu Wahi, S. K. Sourabh On the philosophy of statistical bounds: a case study on a determinant Algorithm. Medwell Journals 2007, p. 15 – 23.

[7] S. Chakraborty, R. Hatwal, T. Rastogi, *On Statistical Bounds and Their Empirical Estimates:A More Meaningful Approach to Average Case Complexity*, International Journal of Mathematical Modeling, Simulation and Applications, vol. 1, no. 2, 2008, 123-136.

[8] S. Chakraborty and P. P. Choudhury. A statistical analysis of an algorithm's complexity. *Applied Mathematics Letters*, 13(5):121–126, 2000.

[9] Soubhik Chakraborty, Divya Nupur Modi and Sunil Kumar Panigrahi, Will the Weight-based Statistical Bounds Revolutionize the IT? (HTTP://WWW.IJCC.US), VOL. 7, NO. 3, september 2009.

[10] C. Cotta and P. Moscato. A mixed evolutionary-statistical analysis of an agorithm's complexity. *Applied Mathematics Letters*, 2003 [available online at http://www.lcc.uma.es/»ccottap/papers/aml03eaa.pdf].

[11] S. Chakraborty, K. K. Sundararajan, *Winograd's Algorithm Statistically Revisited: It pays to weigh than to count!*, Applied Math. and Compu., vol. 190(1), 2007, p. 15-20.

[12] Alaa ismail el-nashar *Parallel performance of mpi Sorting algorithms on dual–core Processor windows-based systems* international journal of distributed and parallel systems (ijdps) vol.2, no.3, may 2011.

[13] Guy Blelloch "Programming Parallel Algorithms.". Comm - unications of the ACM, volume 39, number 3, March 1996.

[14] Felician ALECU Parallel Bubble Sort Economic Informatics Department, A.S.E. Bucharest.

[15] R. Wyrzykowski, *Parallel Processing And Applied Mathematics*, Springer, 2004.

[16] David R. Cheng Viral B. Shah John R. Gilbert Alan Edelman "A Novel Parallel Sorting Algorithm for Contemporary Architectures" May 20, 2007.

[17] S. Ben-David, B. Chor, O. Goldreich, and M. Luby, On the theory of average case complexity. J. Comp. Sys. Sci,. First appeared in STOC. ACM. 1989.