Software Complexity: A Statistical Case Study Through Insertion Sort

Anchala Kumari
Department of Statistics
Patna University
Patna – 800005, INDIA
anchalak@yahoo.com

Soubhik Chakraborty
Department of Statistics
Marwari College
T.M. Bhagalpur University
Bhagalpur-812007, INDIA
soubhikc@yahoo.co.in

ABSTRACT:

The present paper makes use of factorial experiments to assess software complexity using insertion sort as a trivial example. We next propose to implement the methodology in quicksort and other advanced algorithms.

Key Words: Insertion sort, average complexity, interchanges, parameters of input distribution, stochastic modeling of deterministic computer experiment.

The organization of the paper is as follows. Section I is the introduction. Section II gives the code we used. Sections III to V give the results and statistical analysis while section VI is the conclusion and suggestions for future work.

SECTION-I

Introduction

Over the past few decades Software Complexity has opened a new era in the field of Computer Science. It may be defined as a major driver of cost, reliability and functionality of software systems. High software costs and the fact that around 65% of the software life cycle time is spent on testing and maintenance has created considerable attention to the software complexity problem. Both the inherent complexity of the problem and additional complexity of implementation are important aspects and software size is included in this interpretation of complexity because unless special techniques are employed to combat it, the complexity increases with its size.

Over the decade researchers have proposed a wide range of complexity metrics that have been successfully transitioned into practice and have immediate benefits in terms of risk management, reliability predictions, cost containment and improving overall software quality. However, most of these measures suffer from the weakness that much of what is being measured is source text format, which is not an intrinsic attribute of software implementation. Most of the software complexity models have also been developed only in the recent past. One to be cited is

Axiomatic Model of complexity (see E. Jweyuker, ref[6]). The other one is a Stochastic Modeling of deterministic computer experiments proposed by Prof. J. Sacks and others (ref[1]). Based on some of these studies we have given an exposure to statistical approach for evaluating the complexity of a program (software). In this paper we have obtained the complexity of insertion sort when the n observations to be sorted come from Binomial Population B(m, p). To investigate the simultaneous effect of numbers to be sorted (n), Binomial parameters(m & p) and also their joint effects, a 3-cube factorial experiment was conducted with 3 levels of each of the factors n, m & p. Excitingly all the three factors were found to have significant effect on the average time complexity . The link between algorithmic complexity and computer experiments can be found in Chakraborty *et. al*. in ref[5].

Sorting, frequently used in programming, means the rearrangement of items into ascending or descending order. The order in which items are sorted in computer memory often has a profound influence on the speed and simplicity of algorithms that manipulate those items. In literature many sorting techniques are available which provide excellent illustrations of the general ideas involved in the analysis of algorithms. Mainly the sorting procedures are divided into two parts; Internal sorting and External sorting. Internal sorting takes place in the main memory of a computer whereas External sorting is necessary when the number of objects to be sorted is too large to fit in main memory.

There are several internal sorting methods. The simplest algorithms usually takes $O(n^2)$ time to sort n objects and are only useful for sorting short lists. One of the most popular sorting algorithm is quick sort, which takes O(nlogn)time on average and $O(n^2)$ in worst case. The other popular method of sorting is "Insertion sort ." It is so called because in the i-th pass we insert the ith element A[i] into its right place among A[1],A[2], A [3]…….,A[i-1] which were previously placed in sorted order. To make the process of moving A[i] easier, it helps to introduce an element A[0] whose key has a value smaller than that of any key among A[1], A[2],…..A[n]. A QBASIC program for sorting an array of n observations is given in Section II. We first verify the $O(n^2)$ complexity for discrete uniform[1, 2…m] input of size n before we move on to the more exciting Binomial case. For theoretical details on insertion sort, see ref [9]. Although the first study is a routine work, it is of interest as the statistical approach "weighs" rather than counts the operations (see Chakraborty and Choudhury, ref. [10]). The weight of an operation (which in our case is the corresponding time it consumes on implementation) depends on the operation type, the system and even the operands. Another reason is to ensure that the constant "c" in the definition of the big - O does not take an impractical value (remember this "c' depends on implementation). For more on big – O, any standard text on algorithms such as ref [9] may be consulted.

SECTION II

```
    REM insertion sort
    CLS
    INPUT "Numbers to be sorted"; n
    INPUT  "enter the parameter m"; m
    DIM a(n)
    RANDOMIZE TIMER
    FOR i=1 TO n
    a(i)=INT(m*RND)+1
    NEXT i
10 REM insertion sort begins
    st=TIMER
    a(0)=  -1: REM a(0) is less than every sorting number
    FOR i=2 TO n
    j=i
   WHILE a(j)<a(j-1)
   SWAP a(j),a(j-1)
   j=j-1
   WEND
   NEXT i
   et=TIMER
   PRINT "Run time in seconds="; et-st
   END
```

SECTION III

To study the effect of m & n , a 3-square factorial experiment was employed with three levels of m (500,1000,1500) and three levels of n (1000,2000,3000) and the results obtained are given in the table below.

Table-3.1

Sources	d.f	S .S	M.S	F	Table-F(5%)
m	2	0.002	0.001	0.54	3.55
n	2	148.064	74.032	4.2E+04	3.55
m*n	4	.004	0.001	0.55	2.93
error	18	.032	0.002		
total	26	148.102			

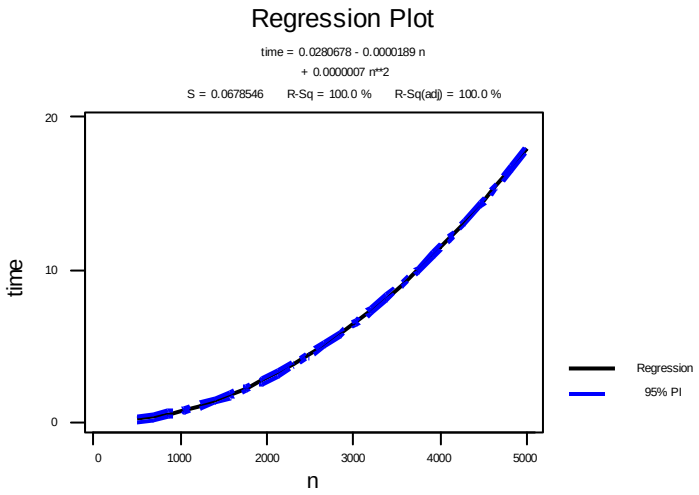
The numbers to be sorted (n) proves to be highly significant whereas the parameter m of uniform discrete distribution comes out to be insignificant We are therefore interested in studying the behavior of n only on complexity for the present case.
The following table gives average run time y (average taken over ten readings) for different values of the arguments n for fixed m=1000.

Table 3.2

n	y
500	.2213542
1000	.6953126
1500	1.628907
2000	2.855469
2500	4.483073
3000	6.389323
3500	8.786459
4000	11.38932
4500	14.6276
5000	17.8112

The values of y when plotted against the values of n using Minitab statistical package indicates a quadratic fit which verifies the theoretical result : "Average time complexity of insertion sort is $o(n^2)$".

Fig 3.1



The line of regression of y on n is $y= 0.280678-.0000189n+0.0000007n^2$

Table-3.3

The Analysis of variance table is given below :-

Sources	df	ss	ms	F	Remarks
Regression	2	335.720	167.860	36457.6	Highly significant
Error	7	0.032	0.005		
Total	9	335.752			

R^2 = 335.720/335.752= .9999 . $100R^2$ = 99.99≈100%. Thus we find that the coefficient of determination is approximately 100 percent: stating thereby that average time complexity of insertion sort can be well explained by quadratic function of input size.

We will now further investigate the topic by simulating n numbers to be sorted from the Binomial population B(m, p), m being the parameter of the distribution and p the probability of success and study its effect on sorting efficiency.
A subprogram incorporating the Binomial (m, p) input is given below.

```
REM generating n Binomial (m, p) elements
CLS
INPUT n
INPUT m, p
DIM a(n)
RANDOMIZE TIMER
For i=1 TO n
xx=0
FOR j=1 TO m
t=RND: IF t<p THEN xx=xx+1
NEXT j
a(i)=xx
NEXT i
```

Once the n Binomial (m, p) sorting elements were generated this way they were then sorted using the code in section II (line no. 10 onwards identical)

Table-3.4

Insertion sort time in sec for n Binomial (m, p) distribution input :-

	p=0.2			p=0.5			p=0.8		
n	m= 10	m=100	m=1000	m=10	m=100	m=1000	m= 10	m=100	m=1000
1000	.549	.672	.694	.582	.641	.695	.548	.678	.716
2000	2.216	2.656	2.774	2.359	2.711	2.823	2.233	2.641	2.784
3000	4.944	5.967	6.376	5.327	5.966	6.279	5.001	6.046	6.426

The following points can be summarized from the above table:

(a) For given n & p , insertion sort time increases noticeably as the parameter m increases. Note that Chakraborty *et. al*. in ref. [11] showed that for given n & p quick sort time decreases as the parameter m increases! This means for higher m insertion sort is not preferable. We already know that it is not preferable for higher n though it is faster than bubble sort and replacement sort, a result proved by Prof. Knuth in ref [4].

To investigate into the matter further, a 3-cube factorial experiment was conducted taking m at three levels (10, 100, 1000), n at three levels(1000, 2000, 3000) and p at three levels(0.2, 0.5, 0.8) using MINITAB statistical package. While analyzing the data the three levels of m, n, p were coded as 0, 1, 2 and results are presented in the table below:

Table-3.5

Analysis of Variance of 3-Cube Factorial Experiment :-

Sources	d.f	S.S	M.S	F	Table(F)	Remark
m	2	5.919	2.960	913.35	3.178	Highly Significant
n	2	367.951	183.976	5.7E+04	3.178	Highly Significant
p	2	0.050	0.025	7.69	3.178	Significant
m*n	4	3.035	0.759	234.19	2.556	Highly Significant
m*p	4	0.142	0.035	10.94	2.556	Significant
n*p	4	0.032	0.008	2.44	2.556	Not Significant
m*n*p	8	0.118	0.015	4.56	2.114	Significant
error	54	0.175	0.003			
Total	80	35556.3				

Table 3.5 reveals following facts:-

Insertion sort is highly affected by the two factors m & n but the factor p has moderate effect on it. When we consider the interaction effect, interestingly we find that interactions m*n is highly significant, m*p moderately significant whereas n*p comes out to be non significant. At this point we may argue that the sorting time increases as the numbers to be sorted increases, but this increase is markedly affected by the changes in the levels of m which is clear from the high value of F(=234.19) as compared with table value(F=2.556) at 5% level of significance; see also table 3.4. However, the change in sorting time due to change in numbers to be sorted is little bit affected with changes in the levels of p which is clear from table 3.4(compare complexity at different levels of p, keeping m & n fixed), these changes comes out to be non significant as a result of F-test in table 3.5(F=2.44 against tabulated F=2.556). Interestingly, the two parameters of the binomial distribution are found to interact among themselves. The change in the levels of p produces significant effect on sorting time as the number of levels of m are also changed.

SECTION IV

In the previous section, it was observed that all the three factors play significant role in describing he complexity of insertion sort. In this section we examine the order of complexity defined by individual factor

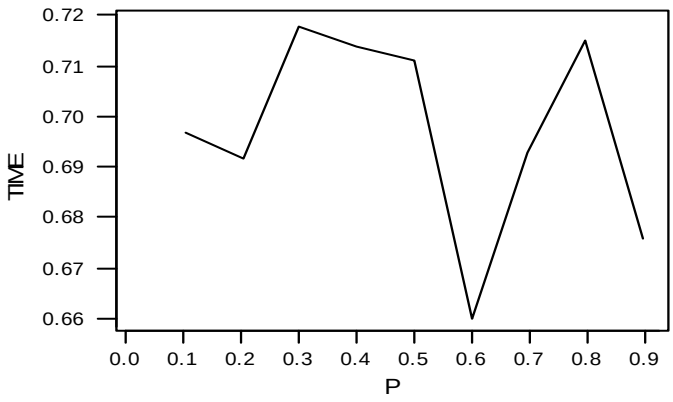
(i) changes in the levels of p keeping m at fixed level=1000 & n at fixed level=1000

Table-4.1

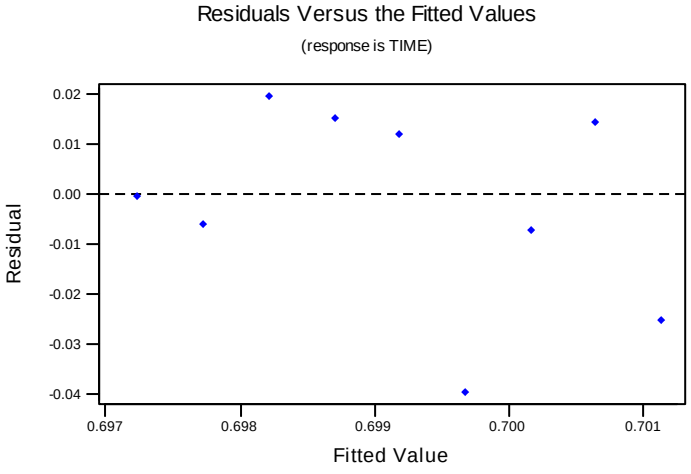
p	sorting time
0.1	.6966146
0.2	.6914063
0.3	.7174479
0.4	.7135417
0.5	.7109375
0.6	.6601563
0.7	.6927084
0.8	.7148438
0.9	.6757013

When time is plotted against the value of p and residuals against fitted value in MINITAB statistical package we get the following figure.

Fig 4.1



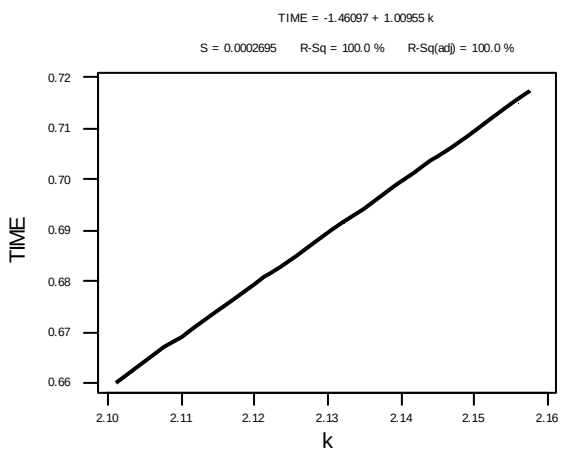
F1g-4.2



The plot of values as well as plot of residuals shows lack of linearity between the insertion short time and binomial parameter p. A plot of residuals against fitted values, as obtained in fig 4.2 is indicative of the fact that lack of linearity is due to non independence and non constant

variance of error terms. When we make the transformation √time + √(time+1) to stabilize the variance , we get a linear trend in figure 4.3.

Fig-4.3



k=√time + √(time+1)

(ii) Time of insertion sort was noted down corresponding to the changes in the levels of m while keeping n fixed at level 1000 and p fixed at level 0.5.

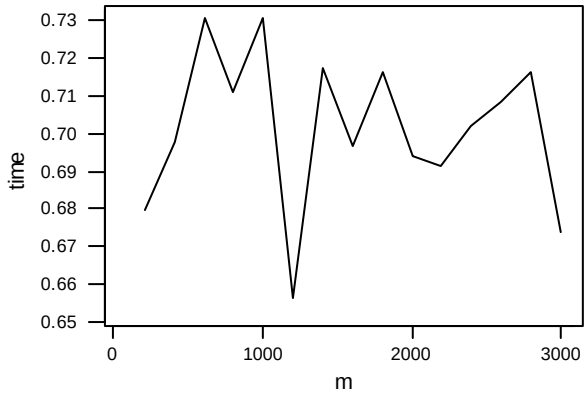
Table-4.2

n=1000, p=0.5

m	insertion sort time
200	.6796876
400	.6979167
600	.730467
800	.7109325
1000	730468
1200	.65625
1400	.71745
1600	.696615
1800	.7161458
2000	.6940104
2200	.691406
2400	.701823
2600	.70833
2800	.716146
3000	.674010

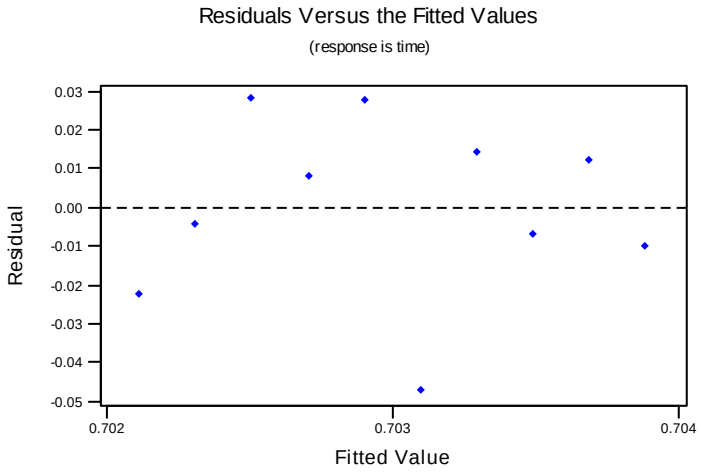
An m-time plot is shown as below :-

Fig-4.4



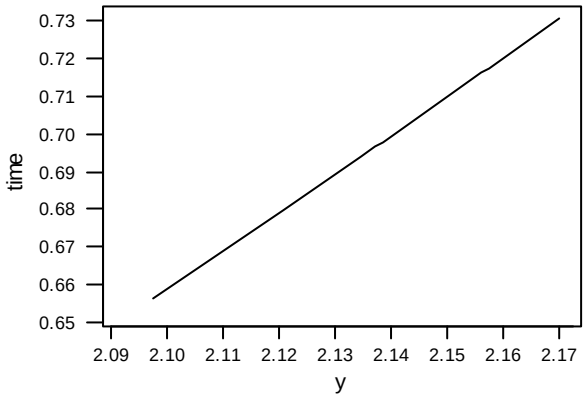
Residual plot is given as below:

Fig-4.5



Neither of the above graphs shows any systematic pattern. Residual plot is indicative of the fact that error terms have non constant variance. As before the transformation √time + √time+1 used to stabilize the variance, shows a linear trend in figure 4.6.

Fig-4.6



SECTION V

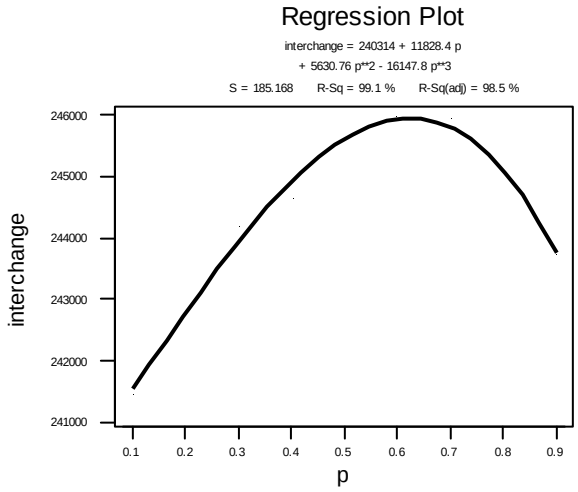
Table 5.1

Simulation result with Binomial distribution input
n=1000, m=1000

P	i=mean no of interchanges	s.d
0.1	241463.4	5464.605
0.2	242845.1	4876.874
0.3	244171.7	3495.884
0.4	244624.0	3956.592
0.5	245572.6	5784.517
0.6	245976.0	3916.423
0.7	245943.9	4588.48
0.8	245081.7	4230.438
0.9	243726.2	4101.091

It is interesting to note that i increases with increasing values of p, attaining a maximum at p= .6, decreasing gradually thereafter with further increase in p. A fitted line plot in MINITAB, shows that behavior of i with changing p can well be explained by a cubic fit. However in case of replacement sort, a quadratic fit was found to be excellent to explain the behavior of i with changing p(see Chakraborty *et. al*. in ref. [5]).

FIG-5.1



In the next study , we observed i versus m for fixed n and p

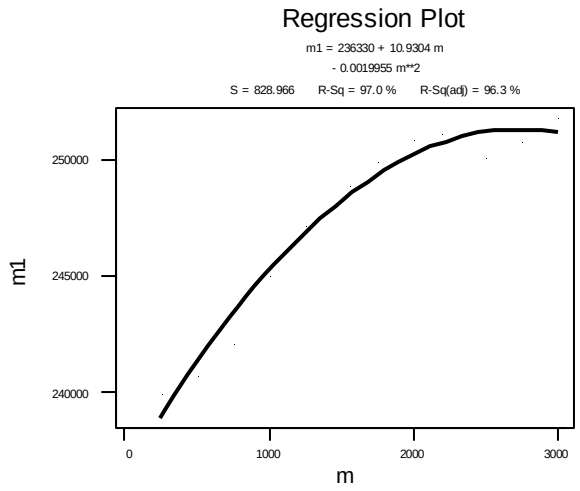
Table 5.2

n=1000, p=0.2

m	i= mean number of interchanges	s.d
250	239997	2831.479
500	240764.2	3305.354
750	242064.2	3340.105
1000	245033.4	5014.582
1250	247168.3	5430.781
1550	248881.7	5958.971
1750	249861.2	6053.645
2000	250796.3	7363.908
2250	251071	4458.402
2550	250045.5	5083.432
2750	250747	6543.963
3000	2251747.8	6253.664

A fitted line plot of interchange versus m in MINITAB shows a good approximation to a quadratic fit with a value of R-square=.97. The plot is given in the figure below.

Fig. 5.2



SECTION VI

Conclusion and suggestions for future work:

3-cube factorial experiment conducted on insertion sort reveals that for certain algorithms such as sorting, the parameters of the input distribution singularly as well as interactively are important factors, besides the size of input, for evaluating time complexity more precisely. However in our case though the two parameters (m & p) of Binomial distribution have significant effect on the sorting time, due to non constant variance of error terms, which do arise in the model due to varying the specific input elements and their relative position in the array for a particular distribution (Mahmoud,2000), some specific pattern between the sorting time and the two parameters can not be specified. But at the same time, when we make the transformation √time + √time+1 to stabilize the variance, sorting time can well be explained as a cubic function of p and quadratic function of m . In section V, it is further revealed that number of interchanges is a function of both the input size and the parameters of the distribution. Combining the results, a new improved model for predicting average time complexity of insertion sort is suggested as follows:

$$T_{avg} = a + b\, n(n-1)/2 + c\, i(n, p, m) + \varepsilon \qquad \text{.....(***)}$$

where n(n-1)/2 represents the number of comparisons which is evident from the code given below:-

```
for i=2 to n do begin
    j=i;
       while a[j],a[j-1] do begin
             swap a[j],a[j-1];
              j:=j-1;
        end
end
```

otherwise also we find that an element at the ith position is compared with each of the previous i-1 elements. So the total number of comparisons in an array with n elements
=1+2=3+ 4+……..+(n-1)==n(n-1)/2 , € is the error term.

Further it can be noted that dependence of the number of interchanges on the parameters is more prominent for discrete distributions rather than continuous distributions. It is due to the fact that probability of tie is zero in a continuous case ,presence of ties and their relative position in the array is crucial for discrete cases .The effect of ties can also be examined as below:-
During every comparison, the probability of a tie[Binomial(m,p) case]

$$= P[a(i)=a(j), \text{ i and j unequal}]$$
$$=\sum_{r=0}^{r=m} P[a(i)=r, a(j)=r, \text{ i and j unequal}]$$
$$=\sum_{r=0}^{r=m} P[a(i)=r]\, P[a(j)=r] \text{ due to independence}$$
$$=\sum_{r=0}^{r=m} [\, {}^{m}C_{r}\; p^{r}(1-p)^{m-r}\,]^{2}$$

which is evidently an expression depending on the parameters (m and p). Given that there are n(n-1)/2 comparisons, one only has to multiply the expression derived above by n(n-1)/2 to get the expected number of ties. If two tied elements are far away, common sense has it that there would be more moves required before they could be brought to adjacent positions. Given further that the average distance between tied elements increases linearly with the array size n, as proved earlier, we immediately have the final model (***).

Important remark: The mean distance between a tied pair in an array of size n randomly filled by n sorting elements from a particular distribution is a linear function of n independent of the parameters of the input distribution. To prove it let D=ABS(i-j) whenever a(i)=a(j) with i and j unequal; i, j=1, 2…n. Obviously D is a discrete random variable with probability mass function P(D=d) = (n-d)/[n(n-1)/2] where d=1, 2, 3….n-1. Note that tied pair at positions (i, j) and (j, i) are taken as same. For this distribution E(D)=(n+1)/3 proving the claim. Also Var(D)=(n+1)(n-2)/18. We omit the details.

Table 6

d	favourable positions	probability
1	(1, 2) (2, 3)….(n-1, n)	(n-1)/[n(n-1)/2]
2	(1,3) (2, 4)…..(n-2, n)	(n-2)/[n(n-1)/2]
……	……	……
n -1	(1, n)	[n-(n-1)]/[n(n-1)/2]

We have used a 3-cube factorial experiment to study the dependence of the factors on average time complexity of a useful program like sorting. However, we can also apply the operations research methods to minimize the average run time of a program, giving thereby a new dimension in this field. This is reserved as a rewarding future work.

As a final comment, insertion sort is not used for higher n. We therefore propose to implement this methodology (using factorial experiments as opposed to surface plots) in quicksort and other advanced algorithms. This is just an illustrative example. The reader must stress the methodology rather than the illustration. Also, since we are directly working on times, we shall regard execution time as a "weighted sum" of computing operations, the weights being the corresponding times. Of course this time has a system dependent part, but we are looking for identifiable patterns that would be the same in every system (see also ref. [10]). Thus we are trying to say things that are "system invariant" rather than "system independent"(the latter means predicting from the desk). Our approach might be "untraditional" but realistic. And we keep n sufficiently large to get rid of idiosyncrasies of computer clocks.
[concluded]

References:

1. J.Sacks, W. Weltch, T. Mitchel and H. Wynn, Design and Analysis of Computer Experiments, Statistical Science, vol.4(4), 1989
2. H. Mahmoud, Sorting: a Distribution theory, John Wiley and Sons, 2000
3. D. E. Knuth, The Art of Computer programming, vol. 1, Fundamental Algorithms, Addison-Wesley (Pearson Education Reprint), 3rd ed, 2001
4. D. E. Knuth, The Art of Computer Programming, vol.3, Sorting and Searching, Addison Wesely (Pearson Eduction reprint) 2nd ed., 2000
5. S Chakraborty, M. Bose, K. Sushant: On Why Parameters of Input Distributions Need to be Taken Into Account For a More Precise Evaluation of Complexity For Certain Algorithms, InterStat Oct2005#3(www.interstat.statjournals.net)
6. E. J. Weyuker, Evaluating Software Complexity Measures, IEEE Transactions on Software Engineering, 14(9):1357-1365, Sept.1988
7. A. Aho, J. Hopcroft, J. Ullman, Data Structures and Algorithms, Pearson Education, 2000
8. N. Draper and H. Smith, Applied Regression Analysis, John Wiley & Sons, 1966
9. A. Aho, J. Hopcroft, J. Ullman, Data Structures and Algorithms, Pearson Education reprint, 2000
10. S. Chakraborty and P. P. Choudhury, A Statistical Analysis of an Algorithmic Complexity, Applied Math. Lett., 13 (5), 2000
11. S. Chakraborty *et. al.*, On How Statistics Can Provide a Reliable and valid Measure for an Algorithm's Complexity, InterStat Dec2004#2