# DESIGN, MEASUREMENT, AND ANALYSIS OF PERFORMANCE EXPERIMENTS ON SELECTED SORTING ALGORITHMS

*Santiago M. Alviar*[1]

*Abstract*

Experimental designs have not been widely used in the study of computing performance although they have been successfully applied in biological and physical sciences experiments. This paper illustrates the use of two experimental models adopted for comparison and prediction of the performance of some selected sorting algorithms. The results showed the potential of designed experiments as valuable tools in conducting computing performance evaluation.

## *1. Introduction*

In general, the study of computing performance of algorithms is done using mathematical analysis. When conducted in a broad class of algorithms solving a particular problem, it is referred to as the study of computational complexity. When it involves the study of time and space requirements of a particular algorithm for a particular implementation it is called analysis of algorithms. The purpose of the analysis is to compare the effect of changes adopted with the end in view of an optimum performance. When wider options for alternative algorithms are available the analysis is used also to provide criteria for selection.

[1]Associate Professor, Institute of Mathematical Sciences and Physics, University of the Philippine at Los Baños; currently International Development Program Fellow, University of Wollongong, New South Wales, Australia.

Statistical methods, on the other hand, when properly used can be faster and more easily applied with less stringent skill requirement on the part of the analyst. Once the methods are refined it can be applicable to a wider class of algorithms which when implemented may need very little or no modifications at all.

This study introduces an alternative means of conducting computer performance evaluation. In particular, the study present an example of analysis of variance and regression analysis models. It introduces basic ideas underlying the design, measurement, and analysis of experiments in the context of computing science studies.

## 2. *Design and Measurement*

Before the application of an experimental design one must know the assumptions underlying the analysis. For analysis of variance, there are four assumptions which must be satisfied by the data so that the result will be statistically valid. One condition is that the observations or the measures must be independent of one another or in more technical terms, the adjacent runs must not be correlated. It also requires that the effect of the imposed condition of the experiment called treatment must be additive rather than a proportionate increase. Equally important is that the variability of measurements from treatment to treatment must be the same. Lastly, the distribution of measurements must follow the normal probability distribution which means that the data balances evenly about its mean and is well behaved. All of these assumptions are necessary for analysis of variance, and any violation will exact some loss in the credibility of the results of the analysis.

To verify whether the observations from the experiment meet the assumptions, a preliminary experiment was conducted with a test program involving the implementation of the selected sorting algorithms which are linear insertion sort, heapsort, and quicksort.

The results of the test are shown in Figure 1. This showed that the means of execution time for the linear insertion sort and quicksort are directly related with the variance or the spread of the values. Also they are characteristically skewed or very much exponentially distributed.

14150-14154 4
14155-14159 88888888888899999
14160-14164 00000011111111112222222223333333
14165-14169 5555667777777888889
14170-14174 0222223344
14175-14179 57
14180-14184 0011144
14195-14199 78
14200-14204 013334
14205-14209 7
14210-14214 12
14215-14219 6

Linear Insertion Sort

1222-1223 23333
1224-1225 44444444444455555555
1226-1227 77777777777777777777777777777
1228-1229 999999999999999999999
1230-1231 0000000000001111
1232-1233 2222
1234-1235 4444
1236-1237 6

Heapsort

730-733 2
734-737 4445566666666667777
738-741 888888888889999999999999900000000000000000000000
742-745 22222222222333334444444444
746-749 6667
750-753 01
754-757
758-761 0
762-765
766-769
770-773 1

Quicksort

**Figure 1. Distribution of execution time (in hundredths of seconds) of linear insertion sort, heapsort, and quicksort, Wirth's Implementation**

For the same set of data, analysis of autocorrelation was run and the results showed no dependency among contiguous measurements. Clearly, among the assumptions underlying analysis of variance this is the only one satisfied. Also, it pointed out that to guarantee independence of measurements all test runs must be made with the machine dedicated. This also eliminated the "other users" effects.

When the variance is related to the mean, Anderson and Bancroft (1957) recommends a logarithmic transformation of all values prior to analysis. The transformation is intended to accomplish three things: (i) to stabilize the variance, (ii) to render the effects additive, and (iii) to approximate normality of the data.

### *3. Experimental Factors*

A factor is an experimental variable which the analyst wants to evaluate at different levels (Winer (1971) ). A treatment, on the other hand, is a combination of the levels of the factors chosen for the experiment. Three factors were chosen for the experiment presented in in this paper. The first factor is the sorting algorithm and the levels are the three selected methods: linear insertion sort, heapsort, and quicksort. The uniqueness of their techniques and the manner by which they behave with respect to some properties of the keys earn their merits as favorite subject of analysis. The second factor of the experiment is the size or the number of records to be sorted. The levels chosen in the study are sizes 16, 64, 256, and 1024. The geometric increase of the levels is to effect linear spacing when transformed to logarithms to the base 2. Third factor is upsequence, a measure of degree of the ordering of keys in sorting algorithms.

Most of the mathematical analyses of sorting algorithms consider random, ordered (non-decreasing) and reverse-ordered keys as the bases and show the fastest, average and the worst performances. Practically, this suggests that a measure of the degree of order of the keys is needed so that it can be used as another factor of the experi-

ment. This will provide a more detailed and comprehensive picture of the behaviour of the performance of the algorithm as affected by this factor. A measure of degree of order of keys is the length of the longest upsequence (U) which was defined by Gries (1982), shown in an example below. An upsequence is any subsequence of an array with non-decreasing elements which are not necessarily contiguous to one another.

Example of the length of longest upsequence (U).

| *Upsequence* | *Element* | | | | | *Length (U)* |
|---|---|---|---|---|---|---|
| | *1* | *2* | *3* | *4* | *5* | |
| Original | 2 | 1 | 4 | 5 | 3 | |
| A | 2 | | 4 | 5 | | 3 (60%) |
| B | | 1 | 4 | 5 | | 3 (60%) |
| Minimum Value | | | | | | 0 ( 0%) |
| Maximum Value | | | | | | 5(100%) |

In the example, A and B are two upsequences whose lengths are both equal to 3. Hence, the longest upsequence is equal to 3.

The length of longest upsequence (U) has maximum value bounded by S, the array size, which is also a factor of the experiment. When used jointly, the dependencies complicate the interpretation of the results since it would be difficult to isolate their separate effects. In practice, whenever possible, this kind of dependency should be minimized and effort should be exerted to eliminate or reduce it.

The approach taken for this study was to take the individual percentages with respect to the maximum value. This allowed the measures to assume values freely over the range 0 to 100 irrespective of

the value of S. Hence, for the example given above, U = 60%. For the experiment the levels of upsequence were maintained at 0, 25, 50, 75, and 100%.

### *4. Experimental Models*

The factors of an experiment in general can be grouped into two types: (i) qualitative or nominal and (ii) quantitative or measurable type. The qualitative or nominal type are those factors with levels that are non-quantifiable or can only be named. Algorithm, programmer, and compiler are some of the examples. The quantitative or the measurable types are those factors with levels arising from measurements. The examples are size and upsequence.

The type of factors in the experiment predetermines the model and the associated analysis that can be used. When at least one of the factors is qualitative, then analysis of variance technique is used. When quantitative factors are added to this experimental model the levels are considered discrete points, and their continuity property is lost in the sense that no interpretation is possible for the intermediate points between two levels. When all factors are purely quantitative then a regression model is used and the analytic methods are linear regression and correlation analyses. There are other models which can be used but discussion of them is beyond the scope of this paper. The two models described above can be expressed more explicitly as follows:

#### *4.1. Analysis of Variance Model*

| | |
|---|---|
| T[i, j, k, l] | [the execution time for the ith algorithm, jth size, kth upsequence and lth replication] |
| = Mean | [the population mean] |
| + A[i] | [ith algorithm main effect] |
| +S[j] | [jth size main effect] |
| + A*S[i, j] | [the interaction effect for the ith algorithm and the jth size] |
| +U[k] | [the kth upsequence main effect] |

| | |
|---|---|
| + A*U[i, k] | [the interaction effect between the ith algorithm and the kth upsequence] |
| + S*U[j, k] | [the interaction effect for the jth size and the kth upsequence] |
| + A*S*U[i, j, k] | [the interaction effect for the ith algorithm, jth size, and kth upsequence] |
| + E[i, j, k, l] | [the random error effect for each of the test runs] |

In this model, we assume fixed effects except for the random error component. This means that these effects are constant parameters and do not have any probability distribution. The error components, however, are assumed to be a random sample from a normal population. The assumptions fit the experimental conditions since all the levels of the factors used are either selected or fixed at predetermined points.

### *4.2. Quadratic Regression Model*

| | |
|---|---|
| T[i] | [the execution time for the ith test run] |
| = Bo | [the intercept] |
| + Bs*S[i] | [the linear effect of S] |
| + Bu*U[i] | [the linear effect of U] |
| + Bss*S[i]*S[i] | [the quadratic effect of S] |
| + Buu*U[i]*U[i] | [the quadratic effect of U] |
| + Bsu*S[i]*U[i] | [the interaction effect between S and U] |
| + E[i] | [the random error effect] |

This model is recommended when a comprehensive exploration of the response to controlling variables is desired. This is almost always accompanied by a graphical representation of the response surface. In industry and agriculture the technique is used in optimization experiments. As a whole, this is referred to as Response Surface Analysis (RSA) which was developed extensibly by Box and Hunter

(1957). The same set of assumptions holds for this model except that the levels required are all quantitative.

## 5. *Methodology*

The experiments were conducted using the facilities of the Department of Computing Science, University of Wollongong. The test runs were made at the Computing Laboratory of the Department which is equipped with two Perkin Elmer (PE 3230) computers running under the UNIX operating system. As a result of the preliminary experiment the test runs were made in a dedicated machine. All statistical analyses were done using a customized package developed specifically for the experiments. The programs for the implementation of the three sorting algorithms given in Wirth (1976) were developed.

The final experiment was implemented by first generating the data for each of the array sizes 16, 64, 256, and 1024 records at the upsequence levels 0%, 25%, 50%, 75% and 100%. All the treatment combinations of factors algorithm (A), size (S), and upsequence (U) were then executed three times in random order. The results of the test runs are presented in Tables 1, 2, and 3. Note that in these tables, size (S) has been converted to base 2 logarithm while upsequence (U) is in percent.

An analysis of variance was run for the three factors using the specified model above. Prior to running the analysis, the data were first transformed into base 2 logarithms for reasons explained in section 2. Regression and correlation analyses were applied on the same set of data for the quadratic model. To get more stable values of regression coefficients the means of the three replications of transformed data were used.

## 6. *Results*

The results of the analysis of variance of the combined data of Tables 1, 2, and 3 are shown in Tables 4 and 5. In the analysis the last level of factor size was eliminated due to a problem on stack overflow for the linear insertion sort implementation.

Table 1. Execution time for linear insertion sort (in hundredths of seconds) by size and upsequence, Wirth's Implementation.

| *Size (S)* | *Upsequence (U)* | *Replication* | | |
|---|---|---|---|---|
| | | *1* | *2* | *3* |
| 4 | 00 | 26 | 25 | 23 |
| | 25 | 23 | 23 | 23 |
| | 50 | 23 | 23 | 21 |
| | 75 | 20 | 18 | 20 |
| | 100 | 19 | 19 | 21 |
| 6 | 00 | 173 | 169 | 172 |
| | 25 | 125 | 125 | 125 |
| | 50 | 113 | 111 | 113 |
| | 75 | 88 | 85 | 85 |
| | 100 | 60 | 63 | 63 |
| 8 | 00 | 2011 | 2010 | 2006 |
| | 25 | 1137 | 1142 | 1137 |
| | 50 | 1007 | 982 | 973 |
| | 75 | 690 | 690 | 681 |
| | 100 | 240 | 243 | 240 |
| 9 | 00 | 7543 | 7539 | 7506 |
| | 25 | 3732 | 3729 | 3734 |
| | 50 | 3412 | 3340 | 3317 |
| | 75 | 2281 | 2292 | 2274 |
| | 100 | 475 | 485 | 471 |

Table 2. Execution time for heapsort algorithm (in hundreths of seconds) by size and upsequence, Wirth's Implementation.

| *Size (S)* | *Upsequence (U)* | *Replication* | | |
|---|---|---|---|---|
| | | *1* | *2* | *3* |
| 4 | 00 | 26 | 24 | 24 |
| | 25 | 24 | 26 | 26 |
| | 50 | 25 | 25 | 25 |
| | 75 | 25 | 24 | 26 |
| | 100 | 28 | 25 | 26 |
| 6 | 00 | 99 | 93 | 94 |
| | 25 | 99 | 96 | 98 |
| | 50 | 99 | 96 | 98 |
| | 75 | 102 | 98 | 99 |
| | 100 | 102 | 103 | 99 |
| 8 | 00 | 415 | 419 | 417 |
| | 25 | 432 | 432 | 440 |
| | 50 | 439 | 432 | 435 |
| | 75 | 440 | 440 | 438 |
| | 100 | 426 | 415 | 418 |
| 10 | 00 | 1901 | 1859 | 1862 |
| | 25 | 1942 | 1933 | 1934 |
| | 50 | 1989 | 1947 | 1947 |
| | 75 | 1968 | 1954 | 1947 |
| | 100 | 2031 | 1995 | 1979 |

Table 3. Execution time for quicksort algorithm (in hundredths of seconds), by size and upsequence, Wirth's Implementation.

| *Size (S)* | *Upsequence (U)* | *Replication* | | |
|---|---|---|---|---|
| | | *1* | *2* | *3* |
| 4 | 00 | 18 | 18 | 20 |
| | 25 | 21 | 21 | 21 |
| | 50 | 20 | 23 | 21 |
| | 75 | 19 | 23 | 19 |
| | 100 | 19 | 19 | 19 |
| 6 | 00 | 71 | 70 | 73 |
| | 25 | 81 | 83 | 85 |
| | 50 | 83 | 85 | 81 |
| | 75 | 76 | 81 | 81 |
| | 100 | 71 | 71 | 71 |
| 8 | 00 | 290 | 285 | 284 |
| | 25 | 343 | 339 | 342 |
| | 50 | 338 | 336 | 336 |
| | 75 | 336 | 332 | 332 |
| | 100 | 275 | 281 | 278 |
| 10 | 00 | 1186 | 1185 | 1177 |
| | 25 | 1440 | 1442 | 1434 |
| | 50 | 1484 | 1489 | 1510 |
| | 75 | 1411 | 1411 | 1415 |
| | 100 | 1137 | 1142 | 1142 |

Table 4. Mean of execution time for the interaction of algorithm, upsequence and size.[1]

| *Algorithm (A)* | *Upsequence (U)* | *Size* | | | *(U) within (A)* | *(A)* |
|---|---|---|---|---|---|---|
| | | *4* | *6* | *8* | | |
| Linear Insertion Sort | 0 | 4.62 | 7.42 | 10.97 | 7.67 | |
| | 25 | 4.52 | 6.96 | 10.15 | 7.21 | |
| | 50 | 4.48 | 6.81 | 9.95 | 7.08 | |
| | 75 | 4.27 | 6.43 | 9.42 | 6.71 | |
| | 100 | 4.30 | 5.95 | 7.91 | 6.05 | |
| (S) within (A) | | 4.44 | 6.72 | 9.68 | | 6.94 |
| Heapsort | 0 | 4.62 | 6.57 | 8.70 | 6.63 | |
| | 25 | 4.66 | 6.61 | 8.76 | 6.68 | |
| | 50 | 4.64 | 6.64 | 8.76 | 6.68 | |
| | 75 | 4.64 | 6.64 | 8.78 | 6.69 | |
| | 100 | 4.72 | 6.66 | 8.71 | 6.70 | |
| (S) within (A) | | 4.66 | 6.62 | 8.74 | | 6.67 |
| Quicksort | 0 | 4.22 | 6.16 | 8.16 | 6.18 | |
| | 25 | 4.39 | 6.37 | 8.42 | 6.39 | |
| | 50 | 4.41 | 6.37 | 8.40 | 6.39 | |
| | 75 | 4.34 | 6.31 | 8.38 | 6.34 | |
| | 100 | 4.25 | 6.15 | 8.12 | 6.17 | |
| (S) within (A) | | 4.32 | 6.27 | 8.29 | | 6.30 |
| (S) | | 4.47 | 6.54 | 8.90 | | 6.64 |

[1]Execution time and size were transformed to base 2 logarithm.

In Table 4, the means of the algorithms showed quicksort as having the best time of 6.30 followed by heapsort with 6.67. The poorest was linear insertion sort with 6.94. Since the three-factor interaction is significant, it is necessary to account for the size and upsequence levels in order to compare the three algorithms. A case in point is the comparison of linear insertion sort at upsequence 100% and for sizes 4, 6, and 8 with heapsort and quicksort. Clearly, the earlier mean comparison is contradicted since linear insertion sort

showed better timing or it was within random error limit of what heapsort and quicksort did. Also, it is clear from Table 4 the overall superiority of quicksort over heapsort and linear insertion sort except in the case of ordered or nearly ordered keys where the latter performed equally or better.

Table 5. The analysis of variance for execution time.[1]

| *SV* | *DF* | *SS* | *MS* | *F* |
|---|---|---|---|---|
| Algorithm (A) | 2 | 9.5609 | 4.7805 | 1986.15** |
| Size (S) | 2 | 443.0603 | 221.5302 | 92037.63** |
| A x S | 4 | 7.9810 | 1.9953 | 828.96** |
| Upsequence (U) | 4 | 4.6049 | 1.1512 | 478.29** |
| A x U | 8 | 9.0899 | 1.1362 | 472.07** |
| S x U | 8 | 2.4051 | 0.3006 | 124.90** |
| A x S x U | 16 | 3.8579 | 0.2411 | 100.18 |
| Error | 90 | 0.2166 | 0.0024 | – |
| TOTAL | 134 | 480.7768 | (CV = 0.74%) | |

[1] Execution time and size were transformed to base 2 logarithm.
**Level of significance less than 1%.

An easy way of interpreting interaction results is to use Figure 2. It is interesting to note in this graph how fast the performance of linear insertion sort is degraded as size is increased and upsequence decreased. On the other hand, the heapsort curve is almost a straight line parallel to upsequence axis indicating the small effect of upsequence. Regarding quicksort, Wirth's implementation considers the middle key as the partitioning point which favors good performance for nearly or completely ordered keys in either sense of ordering. In this kind of study, using analysis of variance alone may not provide all the information that the analyst may want. For instance, even if the factors size and upsequence are quantitative, since their levels are regarded as discrete points, drawing inferences on intermediate points is not possible. Therefore, another method can be considered for that purpose. This time the interest is to find out if the

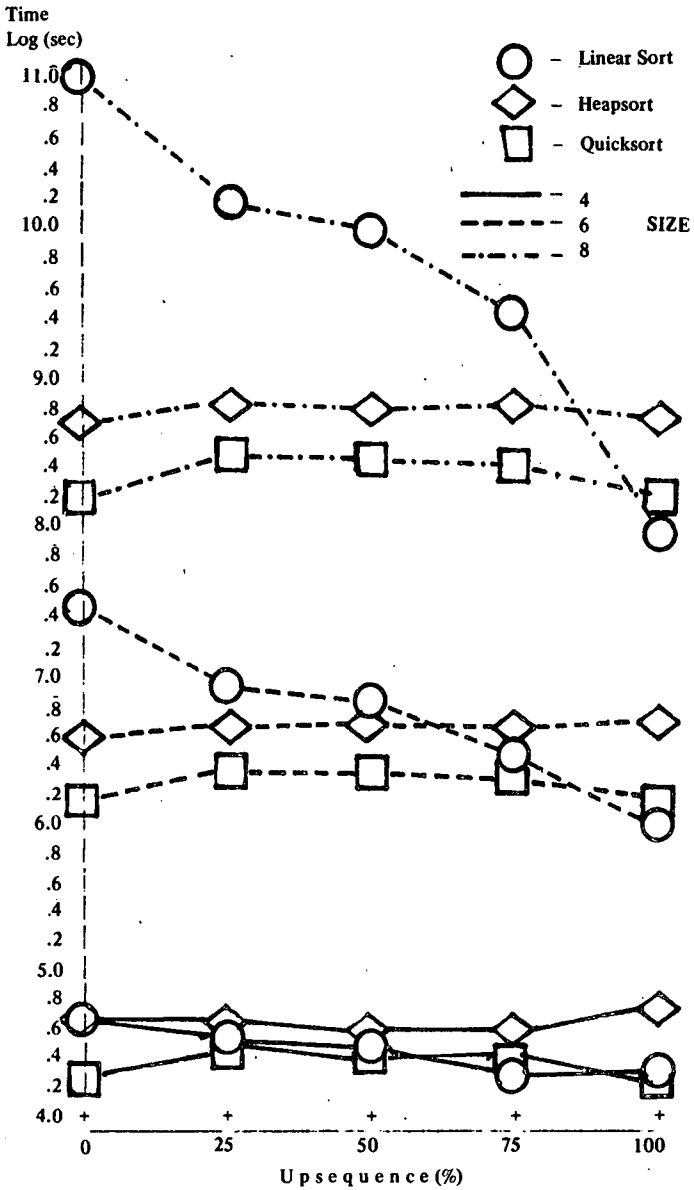


Figure 2. Execution time of linear insertion sort, heapsort, and quicksort as affected by size and upsequence.

performance of each of the algorithm can be expressed as a function of upsequence and size by fitting a regression model.

The quadratic regression model fitted to the data of Table 1 showed that the performance of linear insertion sort is highly dependent upon the degree of order of the keys or upsequence aside from the array size. As shown in Table 6, it is worth noting that the regression terms of the model contribute significantly to the prediction. This is shown in Figure 3.

Table 6. Response surface analysis of execution time on size and upsequence of linear insertion sort algorithm, Wirth's Implementation.

| *SV* | *DF* | *SS* | *MS* | *F* |
|---|---|---|---|---|
| Regression | 5 | 154.175 | 30.835 | 313.09** |
| Error | 14 | 1.379 | 0.098 | |
| TOTAL | 19 | 155.554 | | |

R-square = 0.9911

Test for Ho: Beta = 0.

| *Regression Term* | *Regression Coefficient* | *t-computed* |
|---|---|---|
| Intercept | 0.325198 | |
| Size (S) | 0.720113 | 2.022835* |
| Upsequence (U) | 0.035813 | 3.624452** |
| S x S | 0.074574 | 2.760132* |
| U x U | –0.000137 | –2.035272* |
| S x U | –0.006230 | –6.027755** |

*Level of significance less than ten percent
**Level of significance less than one percent

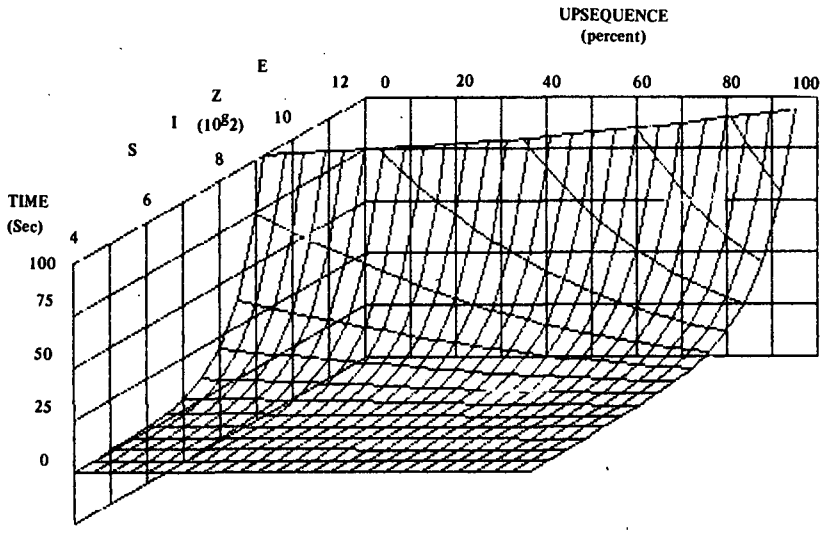


**Figure 3. Response surface of execution time of linear insertion sort on size and upsequence**

The run for heapsort revealed an entirely different behaviour from that of the linear insertion sort. Again, the analysis in Table 7 showed that heapsort performance is not significantly affected by upsequence nor the degree of order of the keys. The test of significance of the regression coefficients showed that only the size, both linear and quadratic terms, contribute to prediction. See also Figure 4.

Quicksort differs markedly from linear insertion sort in the sense that it is generally faster and behaves differently with respect to upsequence. It also enjoys appreciable margin over that of heapsort in performance, practically at all points. Also, using Wirth's implementation, it performed favorably on cases of ordered or nearly ordered keys unlike heapsort. This is seen from the regression coefficients in Table 8. Note that both predictors have significant linear and quadratic terms. The significance of the quadratic terms is a manifestation of better performance when the keys are ordered or nearly ordered. This, however, should be considered specifically true only for Wirth's implementation or for his partitioning strategy. (Figure 5).

Table 7. Response surface analysis of execution time on size and upsequence for heapsort algorithm, Wirth's Implementation.

| *SV* | *DF* | *SS* | *MS* | *F* |
|---|---|---|---|---|
| Regression | 5 | 109.583 | 21.9174 | 25747.32** |
| Error | 14 | 0.012 | 0.0008 | |
| TOTAL | 19 | 109.595 | | |

R-square = 0.9999

Test for Ho: Beta = 0.

| *Regression Term* | *Regression Coefficient* | *t-computed* |
|---|---|---|
| Intercept | 0.972952 | |
| Size (S) | 0.855111 | 36.56** |
| Upsequence (U) | 0.001262 | 1.45 |
| S x S | 0.013715 | 8.41** |
| U x U | −0.000006 | −0.90 |
| S x U | −0.000013 | −0.15 |

**Level of significance less than one percent.

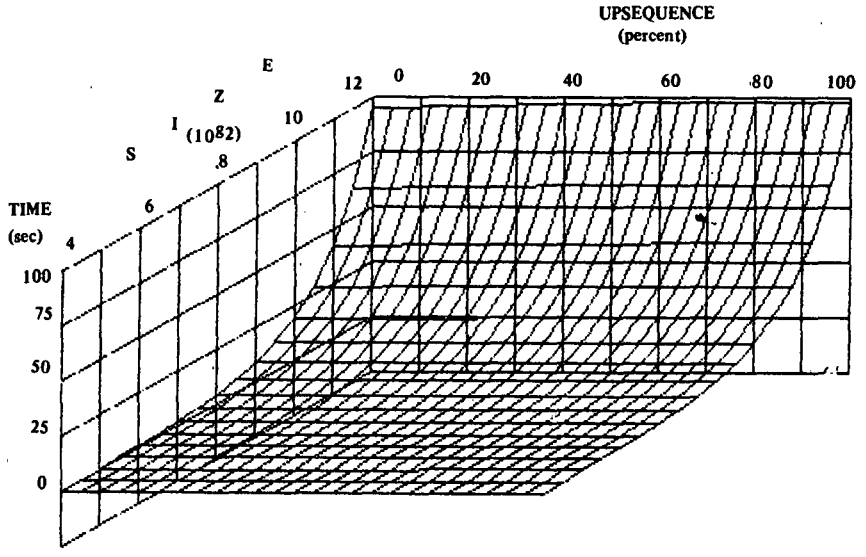


**Figure 4. Response surface of execution time of heapsort on size and upsequence**

## 7. *Summary and Conclusion*

All the results of performance evaluation for the three selected algorithms, linear insertion sort, heapsort, and quicksort agree with the results of mathematical analyses. This shows that statistical methods can be used to strengthen the outcome of the analyses using the other methods or can be relied upon in the absence of any other means of comparison. Note that the coefficient of determination for all the regression analyses gave values higher than 0.99 and significant model F-tests at level of significance less than one percent.

For other computing algorithms, the area of concern is to fit in the right statistical tool or develop the tool together with the development of measurement strategies. The latter is as relevant as the development of tools since the field has not yet been fully explored and studied. As envisioned, this may range from the generation of simple synthetic data as illustrated in this study to scriptwriting of operation scenarios for the experiment.

Table 8. Response surface analysis of executive time of quicksort algorithm on size and upsequence, Wirth's implementation.

| *SV* | *DF* | *SS* | *MS* | *F* |
|---|---|---|---|---|
| Regression | 5 | 101.997 | 20.3994 | 10329.21** |
| Error | 14 | 0.028 | 0.0020 | |
| TOTAL | 19 | 102.024 | | |

R-square = 0.9997

Test for Ho: Beta = 0.

| *Regression Term* | *Regression Coefficient* | *t-computed* |
|---|---|---|
| Intercept | 0.457761 | |
| Size (s) | 0.901041 | 25.29** |
| Upsequence (U) | 0.011109 | 8.38** |
| S x S | 0.007995 | 3.22** |
| U x U | −0.000108 | −11.38** |
| S x U | −0.000088 | −0.70 |

**Level of significance less than one percent

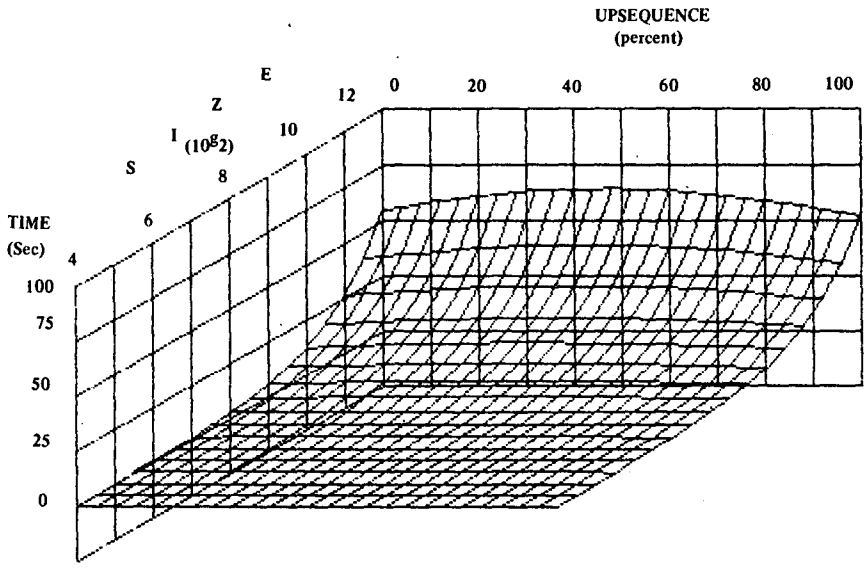


**Figure 5. Response surface of execution time of quicksort on size and upsequence**

Only two methods were illustrated in this study but it should be understood that there are other statistical methods which may be as powerful or even more powerful whose potentials have yet to be tapped.

## *8. Acknowledgement*

The author is indebted to Professor J. Reinfelds and Dr. N. B. Gray for their invaluable advice and assistance. He wishes to thank Dr. H.P. Artis for his personal concern, Ms. S. Linde for her critical reading and editing of the manuscript, and Ms. L. Maxwell for the preparation of the final paper.

## REFERENCES

Anderson, R. L. and Bancroft, T. A. (1957). *Statistical Theory in Research.* McGraw-Hill Book Co., Inc. New York.

Box, G. E. P. and Hunter, J. S. (1971). Multi-factor experimental designs for exploring response surfaces. *Annals of Mathematical Statistics.* 28, 195-241.

Gries, D. (1980). *The Science of Programming.* Springer-Verlag, New York, Heidelberg and Berlin.

Winer, B. J. (1971). *Statistical Principles in Experimental Design.* McGraw-Hill, New York.

Wirth, N. (1976). *Algorithms + Data Structures = Programs.* Prentice-Hall, Inc., New Jersey.

## REFERENCES

Anderson, R. L. and Bancroft, T. A. (1957). *Statistical Theory in Research.* McGraw-Hill Book Co., Inc. New York.

Box, G. E. P. and Hunter, J. S. (1971). Multi-factor experimental designs for exploring response surfaces. *Annals of Mathematical Statistics.* 28, 195-241.

Gries, D. (1980). *The Science of Programming.* Springer-Verlag, New York, Heidelberg and Berlin.

Winer, B. J. (1971). *Statistical Principles in Experimental Design.* McGraw-Hill, New York.

Wirth, N. (1976). *Algorithms + Data Structures = Programs.* Prentice-Hall, Inc., New Jersey.