1986

Strategies used in computer program comprehension and debugging.

Christopher B. Young
University of Massachusetts Amherst

STRATEGIES USED IN COMPUTER PROGRAM COMPREHENSION AND DEBUGGING

A Master's Thesis Presented

By

CHRISTOPHER B. YOUNG

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE

September, 1986

Psychology

STRATEGIES USED IN COMPUTER PROGRAM
COMPREHENSION AND DEBUGGING

A Thesis Presented
By

CHRISTOPHER B. YOUNG

Approved as to style and content by:

James I. Chumbley, Chairperson

Arnold D. Well, Member

Jerome L. Myers, Member

Beverly Woolf, Member

Robert S. Feldman, GPD
Psychology

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Jim Chumbley, whose thought and influence can be found throughout this work. Without his guidance and patience, this thesis would have been a nightmare. I would also like to thank Jerry Myers and Arnie Well for their help with the difficult conceptual as well as statistical problems connected with this work. Finally, I'd like to thank Beverly Woolf of the Computer and Information Science Department for giving us all a good look at this project through the eyes of a computer scientist.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

C H A P T E R I

INTRODUCTION

Computer programming permeates more industries and research laboratories than ever before. With the advent of faster and more efficient machines, the burning question in computer science is no longer efficiency of implementation - the issue is how to make programming easier for the human part of the man-machine synthesis.

One step in that direction is the movement toward more "structured" programming. This style of programming emphasizes the need to break large programs into many smaller, independent subprograms and to keep the flow of control logically linear. A structured program is a hierarchy of subprograms (modules), with the main program at the top, and successively lower-level routines at deeper levels in the hierarchy. Such a hierarchy appears in Figure 1 (see Appendix A). The main program (module #8) calls two modules at the next level (modules 4 and 7) and they each call two modules at the third level. When two modules on the same level are called by a higher level module, the leftmost module is called first (for example, 4 before 7 in Figure 1).

According to the introductory programming book by Schneider, Weingart, and Perlman (1978), the way to write a program with these characteristics is to use a technique called "stepwise refinement". This technique consists of planning the higher levels of an algorithm first, moving to the next level in the hierarchy only after specification of the current level, i.e., planning in a breadth-first fashion.

Presumably, stepwise refinement makes it easier to write a program because it allows one to defer lower-level details and reduces the number of things to be held in short-term memory at any one time. Similar reasoning suggests that it should be easier to comprehend programs when a breadth-first strategy is used. That is, it seems one should start by understanding the "gist" of the main program before proceeding to the more detailed analysis of the concepts at the next level.

This notion receives some support from research reported by Kintsch (1974). Kintsch demonstrated that it is not always necessary to decompose higher-level semantic concepts into their most primitive semantic constituents. This suggests that it should be possible (and perhaps more economical, in terms of processing requirements) to comprehend the modules at a given level without looking at their subprograms (i.e., lower-level semantic

constituents). Breadth-first comprehension of a program would proceed as illustrated in the top panel of Figure 1.

However, at least one theorist characterizes program comprehension as a goal-directed, depth-first process. Brooks (1983) proposed that programs are comprehended by generating and testing hypotheses about the function of its various parts, in a top-down but depth-first manner. The initial hypothesis about the program's function is successively refined by construction of subsidiary hypotheses, until a level is reached where the hypotheses can be compared against the code. A depth-first traversal of a program is represented in the middle panel of Figure 1. Unfortunately, Brooks did not provide any empirical support for his proposal.

Linger, Mills, and Witt (1979) have offered a third viewpoint on program comprehension. According to Linger et al., program comprehension may proceed "bottom-up", especially when the program is poorly documented. Following this strategy, lower-level modules are comprehended first, in order to provide a basis for understanding the higher-level modules. Although the notion of a "bottom-up" traversal of a structure like that in Figure 1 is simple enough, the operational definition of a single path is not clear. There appear to be several

such paths that could be called "bottom up".

The primary purpose of this research is to ascertain the strategies skilled programmers actually use in the comprehension of Pascal programs. If there are enough subjects using different strategies, the differing strategies can be compared for efficiency on some measure of comprehension. In the studies described below, the subjects' strategies are assessed by recording the order in which subjects studied the program modules and the time spent studying each module. From this record, the strategy a subject is using during comprehension can be determined since different strategies are associated with somewhat different paths through the program.

Given identification of the strategies skilled programmers use, program comprehension must be measured. There are a variety of methodologies for measuring program comprehension. Shneiderman (1980) recommends verbatim recall of the program source code. There are a number of problems with this method of assessment. First, verbatim recall seems to have little ecological validity - how many programmers recall code verbatim on the job? Secondly, verbatim recall of programs large enough to be useful is essentially impossible. Finally, ability to do verbatim recall does not necessarily imply that comprehension has

been attained. Other, more ecologically valid measures include program modification and debugging.

Program modification has the drawback of being potentially very time-consuming for the subject, especially if a modification sufficiently extensive to force comprehension of the entire program is used. Also, performance on such a task would be difficult to score since it is difficult to set an appropriate criterion for scoring this task. For example, how should an "algorithmically correct" but syntactically incorrect (poorly coded) modification be scored? It is difficult to set an appropriate criterion for scoring this task. One could conceivably use several different scoring schemes and obtain different results depending on which method was used for scoring.

Debugging, on the other hand, is not so time consuming for the subject if only one bug is used. However, there is evidence that debugging can be accomplished without full comprehension of the program. Sheppard and Love (1979) found that subjects used one of two methods to debug the programs they were given. They either used shallow search strategies based on erroneous output that had been provided, or they set about understanding the program from "beginning to end". It is

not clear from the report whether "beginning to end" means subjects scanned the listing from top to bottom, or that they followed the execution order of the program. Since the language used by Sheppard and Love (1979) was FORTRAN, it is quite possible that these two paths were confounded.

There is evidence from Gould and Drongowski (1974) that the type of bug can influence the strategy employed when debugging someone else's program. Gould and Drongowski used three types of bugs - "array bugs" (causing an array to exceed its dimensioned value), "iteration bugs" (causing an inappropriate number of iterations through a loop) and "assignment bugs" (created by changing a variable in an equation or assignment statement - for example, changing Z=X*Y TO Z=X*Z). It was found that assignment bugs were more difficult to detect (i.e., took longer to locate and/or were located less frequently) and required a "deeper" debugging strategy (involving a greater understanding of the program) than the other two types. It is conceivable that this result is due to a debugging strategy rather than the inherent difficulty of assignment bugs. If subjects looked for syntactic bugs first, or attempted to verify higher level logic and flow of control before getting into lower-level details, assignment bugs would have generally taken longer to locate.

The finding that assignment bugs were most difficult to locate was replicated by Atwood and Ramsey (1978). Atwood and Ramsey used Gould and Drongowski's programs, but conducted a more extensive theoretical analysis of the situation. They used the Kintsch and Van Dijk (1978) framework for text comprehension to develop a propositional representation of the programs. Their propositional hierarchy corresponded to the algorithmic structure of the programs, with the higher-level logic at the top of the hierarchy, and lower-level details at the bottom. They found that the deeper the bug was in their propositional representation of the program the harder it was to locate. Unfortunately, this finding is ambiguous because bugs deeper in the hierarchy also appeared later in the listing. This confounding occurred because the programs they used contained no subroutines, thus further obscuring the distinction between position in the listing and stage in the algorithm of the program.

A secondary purpose of Atwood and Ramsey's study was to ascertain whether a bug deeper in the hierarchy of modules was more difficult to locate than one higher in the hierarchy. According to Atwood and Ramsey, a bug deeper in the propositional hierarchy should be harder to locate, since the lower-level code in which it is embedded should

be less well understood (since it is less well remembered) than the higher-level code. Further, it may be that any difference in difficulty at various levels interacts with the strategy subjects use. If a bug in module 7 (Figure 1) is placed after the calls to modules 5 and 6, a depth-first subject should take longer to locate it than a breadth-first subject.

While I believe Atwood and Ramsey's representational scheme is inadequate, it should be noted that development of the analogue to the Kintsch and Van Dijk propositional textbase is difficult for computer programs. There seems to be no obvious way to represent the concept of a conditional branch as it has no direct analogue in regular text. Atwood and Ramsey chose to solve this problem by parsing their programs into propositions that were essentially the same as FORTRAN primitives. It seems necessary to utilize a more general and psychologically relevant group of primitive propositions if we want to say something about comprehension as a psychological process.

With few exceptions, e.g., Atwood and Ramsey (1978), most of the studies done on computer program comprehension can be broadly characterized in one of two ways. First, there is a large group of well-controlled, empirical and "applied" studies which are virtually theoretically barren.

Conversely, there is another, smaller group of papers which deal with comprehension of computer programs on a theoretical, speculative level but relate the theory to few, if any, empirical findings. Such a dichotomous approach to research is not as desirable as studies which contribute to both a general understanding of basic principles and to the solution of applied problems.

The line of research proposed here is different in that its goal is to identify the strategies used in comprehension of programs, to relate these findings to psychological theory about comprehension, and to (hopefully) improve these strategies. It is also quite possible that the results of this sort of research will bear on such issues as the design of programming languages. This research study represents an attempt at the first phase - identification of the strategies used by skilled programmers.

Three studies on the identification of strategies have been completed. In the first, subjects were asked to debug one of two Pascal programs, each with an assignment bug in one line of the program. Pascal was chosen because of its emphasis on the use of structured programming techniques discussed earlier. Assignment bugs were chosen because the relative "semantic" difficulty of this type of

bug should help to force comprehension because locating bugs of this type should require more global knowledge of the program than that required by array bugs or syntactic bugs. The programs used in this study had three levels (including the main program) of submodules as shown in Figure 1. This made it possible to place the bug at different levels in the programs and allowed at least a partial unconfounding of position in the listing and level in the hierarchy.

Subjects were asked to debug the programs without output. Although it could be argued that this is not the way people actually debug programs, there is evidence that output is not necessary (or even that helpful in some cases) for debugging. Gould and Drongowski (1974) found that subjects provided with output (either "correct" or that produced by a bugged version) did not debug significantly faster than those who had no such debugging aids. Further, based on the findings of Sheppard and Love (1979) discussed above, it was desirable to avoid encouraging shallow debugging strategies.

C H A P T E R II

EXPERIMENT I

Method

Materials. Two Pascal programs were written by the author for use in the study. The first is an inventory program, matching stock against a purchase order, while the second is a payroll program. The programs are 141 and 169 lines in length, respectively, and have identical hierarchical structures. The main program calls two modules at the second level, and each of these modules calls two modules at the third level. This hierarchical structure is illustrated in Figure 1. The unbugged programs appear in Appendix B. It should be noted that Pascal's declaration scheme requires that the global declarations appear first, followed by the body of the program, with the main program at the bottom. The module numbers in Figure 1 indicate one permissible order of declaration. Note that subprocedures must be declared before their calling procedure is declared. For example, in Figure 1, modules 2 and 3 must be declared before module 4, 4 before 8, and so on.

A bug was placed in one of three locations in each program: (1) in the second module called at the second level (module #7 in Figure 1), (2) in the first submodule of that module called at the third level (module #5 in Figure 1), and (3) in the second submodule (of the same second-level module) called at the third level (module #6 in Figure 1). This means that the bugs were only placed in the later half of the hierarchy. The first halves of both programs primarily involve read operations making it difficult, in most cases, to find appropriate assignment bugs in the first half of the hierarchy. Bugs placed in second level procedures were placed after the calls to the third level procedures in order to maximally differentiate between depth- and breadth-first strategies. A purely depth-first subject should take longer than a purely breadth-first subject to locate these bugs. The bugs and their line numbers appear in Table 1.

Table 1

Assignment Bugs (With Corresponding Line Numbers)

Position	Line Number	Code
	Inventory Program	
Level 2	133	COST:=COST+ORDERAMT+PRICE
Level 3 left	83	POUNDS:=(POUNDS+ORDERAMT)*WEIGHT
Level 3 right	105	UNITS:=TOTAL-ORDERAMT
	Payroll Program	
Level 2	157	PAY:=REGPAY+BPIECES*PAYRATE
Level 3 left	107	AVE:=COUNTA+COUNTB DIV 2
Level 3 right	131	BONUS:=BONUSCUTOFF*(BONUSAMT+DOUBLEBONUS)

Apparatus. In the past, the only way to trace subjects' paths through programs was by having them trace their own progress with a felt-tip pen. For this research, a special scrolling program was written to present the program to the subject in parts so that later the subject's path through the program could be traced. The program displayed only one module at a time and recorded how long the subject studied that module. The global declarations, comments, main program, and all subroutines were treated as modules by the scrolling program. The program allowed the

subject to jump to the global declarations, main program, or comments, to scroll up or down by one line (within a module) or module, to search for an arbitrary string, and to return to the previous location after a jump. A maximum of 21 lines were displayed at one time.

Subjects. 18 subjects from those University of Massachusetts students who had successfully completed at least the University's data structures course (the second programming course in Pascal) volunteered to participate. The goal was to obtain a skilled and relatively homogeneous (with respect to ability) sample. Unfortunately, the subjects were representative of a relatively large range of backgrounds. Some subjects had completed only two Pascal courses, others were also consultants at the University Computing Center, and 1 subject was engaged in writing a compiler for FORTH in Pascal. Subjects participated for one hour and were paid $10.00.

Design. The experimental design was a 2 X 3 complete factorial, between subjects design, with 3 subjects per cell. The two independent variables were Program (inventory or payroll) and position of bug in the program. The dependent variables were time to locate the bug and the proportion of time spent utilizing a particular strategy. Predictor variables for additional analyses were GPA in

computer science courses and quantitative SAT scores.

Procedure. Subjects were asked to debug a Pascal program of moderate length (about 150 lines). Instructions to subjects appear in Appendix E. Briefly, the subjects were told that they could assume that the program compiled correctly (meaning the bug was non-syntactic), and that the program contained 0-5 bugs (although in reality there was only one bug). This vagueness was recommended by Shneiderman (1980) to encourage more thorough comprehension during debugging. The subjects were not allowed to use written notes during the debugging session. After being instructed, they were given approximately 15 minutes of practice time using the scrolling program on a Televideo 912 terminal.

Following the practice time, the experimenter left the subject room and the Pascal program was displayed under subject command by the scrolling program. The only constraint put on the subjects was that they start in the global declarations after reading the comments. This should have avoided any bias in favor of any particular strategy. The scrolling program recorded the subject's progress through the Pascal program, timing the study time at each point in the program. The experimenter could monitor the subject's moves via a second screen in the

adjacent room. After correctly identifying the bug or studying the program for 45 minutes, the subject was informed that the session was over. Those subjects who incorrectly identified the bug were informed that the code was correct and that they should continue to search for the bug. At the end of the session, subjects filled out a questionnaire asking them about their programming background and use of strategies.

Results and Discussion

A variety of summary techniques were used in an attempt to identify the strategies used by the subjects to comprehend the programs. The first attempt took the form of a "frame-by-frame" graphic representation of the subject's moves, which displayed each module a subject moved to and how long the module was studied (see Appendix C for a sample). Unfortunately, the sheer volume of data for each subject made detection of any visible strategy very difficult. These summaries of subject behavior at a micro-level are, however, available to confirm conclusions made from other measures.

The second attempt at summarization consisted of constructing a 9 X 9 transition matrix (see Appendix D) for each subject. A subject's matrix showed how many moves were

made to a given module from any other module, and the total time associated with these moves (i.e, the time spent studying that module after moving from the preceding module). Thus, it was possible to look at cells in the matrix that characterized the different strategies, and to determine the total number of moves and time associated with the move in question. For example, one could look at the cells (4,7),(2,3),etc. (see Figure 1) to determine the number of moves and amount of time spent using a breadth-first strategy.

This method was somewhat more successful since it is possible to see more global patterns of strategy use by subjects. Examination of the matrices suggested that instead of demonstrating breadth- or depth-first strategies, subjects appeared to move sequentially through the program, from top to bottom, and/or from bottom to top. Since Pascal programs are declared in the order previously described, this path seems to have little to do with a breadth-first overview or execution order (depth-first path).

However, this pattern could be a function of the set of commands made available to the subject by the display program used in this experiment. In order to proceed through the program in a breadth- or depth-first manner, a

subject would need to use the search facility to directly display the next module appropriate to the strategy. It was observed, and corroborated by subjects, that it was simply easier to use the forward or backward scroll facility (especially since it jumped by modules in order to keep only one module on the screen at any time). This sort of reasoning suggests that the sample may have included subjects using depth- or breadth-first strategies but their matrices may have been cluttered with sequential "transitional" moves.

To test this possibility, a program was written to ignore moves that were less than or equal to a specified cutoff time. For example, if a sequence of moves from module 7 (Figure 1) to module 5 via module 6 had a study time associated with module 6 less than or equal to the cutoff, the sequence was recoded as a move from module 7 to module 5. Since there was some overlap among moves that are characteristic of the three strategies, all the cells in the matrix were split up into 9 mutually exclusive "strategic" categories. These consisted of: the seven sets of cells that were consistent with at least one of breadth-first, depth-first, and sequential strategies; the set of cells consisting of moves to and from the comments and the global declarations; and, all cells not in any of those 8

categories. One of the categories (breadth-first intersection depth-first intersection sequential) was empty (no cell was part of all three paths), leaving 8 categories for analysis.

A 2-way, within-subjects ANOVA was run with the cutoff times (0-10 sec., by 1 sec. steps) and the strategic categories as the 2 factors. The dependent variable was the proportion of the total time spent in each category at each of the cutoffs. It was decided that proportion of time should be used rather than proportion of moves because: (1) the two should be highly correlated, meaning that any information embodied in the moves data should also be present in the time data, and (2) it is likely that the time data is a more sensitive measure, since some subjects had very few moves in the matrix.

The main effect for cutoff is of no interest, since the average proportion over categories at each cutoff must equal 1.00/8. There was a significant main effect for category, and a significant interaction of category with cutoff. In addition to the proportions at each cutoff, a baseline proportion and the corresponding 95% confidence interval were calculated. The baseline proportion was calculated by assuming that a random distribution of moves would result in roughly equal total times in all the cells.

Thus, the baseline for a given category would equal the number of cells making up that category divided by the total number of cells in the matrix. The curves for the zero cutoff proportion and asymptotic proportion for each category with its baseline and confidence limits appear in Figure 2. Figures depicting the proportion at each cutoff for all categories appear in Appendix F.

The figures reveal a clear pattern. Only the curves for the categories containing at least some sequential cells (BS, DS, and S) are consistently above the upper confidence limit. There is a slight hint of some other strategy since the curve for the intersection of breadth-first, depth-first, and not sequential paths (BD) does reach a level above the 95% confidence interval after discarding study times less than or equal to 1 sec.

Examination of individual subject data indicated that no subject was above chance on the proportion of depth- or breadth-first moves (before applying the cutoffs), and virtually all subjects were above chance on the proportion of sequential moves. Even after the transitional moves were removed, only a few subjects showed proportions of depth-first moves above chance. Further, the individual sequential proportions were rarely low enough to be considered due to chance.

The debugging times are shown in Table 2. One of the bugs (the bug in module number 5 of the inventory program, i.e., the level 3, right module bug in Table 1) was not located by any of the 3 subjects in that cell. Although the bugs are all assignment bugs, particular (albeit similar) bug and level of the bug were confounded in this study. It is therefore impossible to know whether the particular bug or its placement was responsible for the difficulty in locating it. On close inspection, however, it seems that the unlocated bug requires a somewhat deeper knowledge of the program than do the rest, possibly accounting for its relative difficulty. A 1-way, between-subjects ANOVA was performed on the remaining 5 cells, using the times of only those subjects those who found the bug. There was no significant difference in the debugging times ($F = 1.41$, $SED = 464.03$).

Table 2

Mean Debugging Times (in sec) for Subjects Finding the Bug

	Program	
Position	Inventory	Payroll
Level 2	1007.65 (2)	1380.45 (2)
Level 3 left	1311.27 (3)	588.90 (2)
Level 3 right	--- (0)	1626.07 (3)

Note: Number of subjects who found the bug in parentheses.

It is clear that the subjects in this study utilized a strategy that could be referred to as "sequential" to perform the debugging task. The question then becomes: Why, would skilled programmers attack this task in a sequential fashion? There are at least 3 possibilities. First, there could be a legitimate reason for doing the task in this manner - perhaps Pascal's declaration scheme lends itself to this sort of strategy.

This seems unlikely, however. Looking at Appendix B, the reader can see that the global declarations appear at the top, followed by the body of the program, with the main program at the bottom of the listing. Further, lower-level subroutines must be declared before the calling routine (at a higher level) is declared. This seems to result in a

less than optimal order for someone comprehending sequentially, unless the subject makes mostly backward-sequential moves (labeled with a "B" in Figure 1), which none of the subjects did. Even if backward-sequential moves (other than short transitional moves) are indicative of other strategies, it is hard to imagine a way to declare these programs such that a backward path would yield a depth-first traversal of the hierarchy. There is a declaration order that would correspond to a breadth-first traversal - however, the proportion of "actual" breadth-first moves is so low that it seems unlikely that the backward moves are "disguised" breadth-first moves.

Second, despite instructions to "debug by getting a good understanding of the program first", subjects may have decided to do something different than they normally would have when comprehending someone else's program. Their goal may have been debugging first and comprehending second.

Third, there is the characteristic of the display program already mentioned. It was easier to scroll sequentially through the program to find a particular module than to use the search command to find it.

The second experiment was a variation of the first,

with subjects comprehending a correct version of a program and then debugging the program after introduction of one bug. The second experiment was necessary to be certain that subjects in the first study were indeed comprehending, and not doing something more like surface debugging, despite the evidence that a debugging task with an assignment bug should force comprehension. The results of Experiment I suggested that subjects went through the program "sequentially" (top to bottom, and vice-versa). This pattern does not correspond to either a breadth- or depth-first strategy, and perhaps indicates a debugging strategy rather than a comprehension strategy. Thus, the obvious manipulation was to separate the comprehension and debugging tasks.

C H A P T E R III

EXPERIMENT II

Method

Materials. In this experiment, only the inventory program was used with the first bug appearing in Table 1. This bug was chosen because of its moderate difficulty (cf., Table 2) and its representativeness of the bugs in Experiment I.

Subjects. 12 subjects were recruited from those University of Massachusetts undergraduates who have been admitted to the University's computer science major. This resulted in a more homogeneous (with respect to background) sample than in Experiment I. Subjects were paid $10.00 to complete the 1.5 hour experiment.

Design. Subjects' strategies were studied in two tasks, comprehension and debugging. The dependent variables were time to locate the bug and proportion of time spent utilizing a particular strategy during comprehension and during debugging.

Procedure. The procedure was the same as that for Experiment I except that subjects were told that they were to comprehend the program and that they would be "tested" on their comprehension of the program. They were not told what type of test would be administered. (The "test" was the debugging task, with the program they had just studied, except that it now contained the bug mentioned above). The reason for this instructional change was to reduce the likelihood of subjects using strategies aimed at debugging rather than comprehension during the comprehension phase. The scrolling program was used in both phases to display the program and record study times.

Results and Discussion

The mean comprehension and debugging times (in seconds) were 1567.69 and 655.78, respectively. The data output by the scrolling program were summarized using the same methods as those used in Experiment I, yielding records of the subjects' paths for both the comprehension and debugging phases of Experiment II. A 2-way, within-subjects ANOVA was run for both phases of the experiment on the proportion of total time spent in each category at each of the cutoffs. There was a significant main effect for category and a significant interaction of category with

cutoff for the both the comprehension and debugging data. We now turn to more detailed analysis of the data.

Comprehension.

The zero cutoff proportion and asymptotic proportion for each category with its baseline and confidence limits appear in Figure 3. Figures depicting the proportions at each cutoff for each category appear in Appendix F. The group data are similar to those in Experiment I, with one important difference. Although the proportion of time spent in sequential moves is still above the upper confidence limit, the proportion of time spent in "pure" depth-first moves (category D) exceeds the upper confidence limit at the 1 sec. cutoff point, and remains above the limit through the rest of the cutoff points. Since the proportion of time spent in breadth-first moves is even lower (relative to the lower limit of the confidence interval) than in Experiment I, it is clear that two comprehension strategies (sequential and depth-first) predominate in Experiment II.

Examination of individual subject data confirmed this observation. No subject had any breadth-first proportion

significantly above the baseline - the subjects seem to split between sequential and depth-first strategies. Unlike Experiment I, the proportion of time spent by some subjects in depth-first moves exceeded the upper confidence limit at the 0 sec. (all moves included) cutoff. It should be noted, however, that even the depth-first subjects exhibit a substantial sequential component.

Debugging.

The zero cutoff proportion and asymptotic proportion for each category with its baseline and confidence limits appear in Figure 4. Figures depicting the proportion at each cutoff for each category appear in Appendix F. As the reader can see from Appendix F, the breadth-first curve does exceed the upper limit of the confidence interval from the 4 sec. cutoff to the 6 sec. cutoff points, perhaps indicating some tendency toward breadth-first debugging. However, the curve for the sequential category exceeds the upper limit at most of the cutoff points, indicating that the primary debugging strategy was a sequential one.

Again, examination of individual subject proportions corroborates the group results. Although there are a few subjects that could be described as utilizing a breadth- or

depth-first strategy after removal of shorter transitional moves, most of these subjects also have a significant sequential component. The individual data, however, are much less clear-cut than the group data.

In order to assess the degree to which comprehension strategy determined debugging time, subjects were split into a depth-first group (n=6) and a sequential group (n=6) based on their individual proportions. There was no significant difference in debugging time due to strategy, $t(10) = -0.7896$, $P > .05$. Interpretation of this null result should be undertaken with care, however, due to the relatively low power of the test.

Additional analyses.

To be certain that the results were not influenced by the choice of dependent variable, the analyses of variance were repeated for both studies, using proportion of total moves as the dependent variable. The results of these analyses were essentially identical to those performed on the proportion of total time.

C H A P T E R IV

EXPERIMENT III

The primary purpose of Experiment III was to ascertain the degree to which the tendency of subjects to adopt a sequential strategy during comprehension and debugging was due to an artifact of the scrolling program. With a slight modification in the scrolling program, it was possible to keep subjects from: (1) scrolling into or out of a module accidentally, and (2) using the scroll facility for moves between two modules. This was accomplished by allowing subjects to specify which module in the hierarchy to jump to. This modification removed the "bias" toward a sequential strategy (while not introducing any bias against it) which I believe was operating in the previous two studies.

A second purpose of this study was to examine the effect of level of training on comprehension and debugging strategies. In an attempt to explore this question, two groups of subjects with different levels of training in computer science were recruited.

Method

Materials and apparatus. The materials were the same as those used in Experiment II. The scrolling program was modified so that subjects could not scroll into or out of a module. Instead, the hierarchy of modules was displayed next to the command menu. The display looked like Figure 1, except that there are no lines or arrows like those in Figure 1. Each module was numbered in the order it was declared, with the global declarations being numbered 1, and the main program being numbered 8. The comments were displayed by a separate command, as in the previous studies. The program now included an option to jump directly to a particular module number. This modification gave all three strategies equal footing in terms of their implementation using the scrolling program.

Subjects. 20 subjects were recruited from those University of Massachusetts undergraduates who had been admitted to the University's computer science major. These subjects were all seniors with at least 80 degree credits, and had taken 5-11 computer science courses. Another group (novices) of 9 subjects was recruited from the University's data structures course, and had taken 2-3 computer science courses.

Design and Procedure. The design and procedure

differed only slightly from those of Experiment II. During the practice time with the scroller, it was explained to subjects that the numbers on the hierarchy corresponded to the order that the modules would appear in a listing. Subjects were also told that the main program (module 8 in Figure 1) calls the two modules at the next level, that the left module is called first, and that this process is repeated at the next level with these modules and their subprocedures. Additionally, subjects were given a list with the numbers and their corresponding procedure names (see Table 3).

Table 3

Module Numbers and Corresponding Module Names Used in Experiment III

Module number	Module name
1	Global Declarations
2	Getorder
3	Getstock
4	Readorderandstock
5	Tellshipping
6	Tellproduction
7	Checkorder
8	Main Program

Results and Discussion

Surprisingly, traning had little effect on the total comprehension or the total debugging time. The mean

comprehension times for the seniors and novices were 1499.68 seconds and 1619.69 seconds, respectively. The mean debugging times (also in seconds) were 476.95 and 686.33, respectively. Although the means were in the expected direction (seniors less than novices) the t-values did not reach significance - $t(27) = .46$, $p > .05$ and $t(27) = 1.20$, $p > .05$, respectively. This result is almost certainly due to the high variability in the debugging times, a problem which was not helped by having only nine novice subjects. Variability among programmers is well known, and suggests that any study designed to investigate effects due to experience would be well advised to run a large number of subjects.

The data output by the scrolling program were "reduced" to proportions of time and moves consistent with the 8 strategic categories for both comprehension and debugging. Due to the modification of the scrolling program, it was no longer necessary to apply the cutoff times, since the rationale for their application was the filtering of transitional sequential moves. As in Experiment II, analyses performed on number of moves did not change the <u>pattern</u> of results in any appreciable way. In every case, analysis on proportion of total moves has the effect of increasing both depth-first and sequential proportions somewhat (relative to proportion of time) while

decreasing the proportion consistent with their intersection. Consequently, except where so stated, analyses reported below were run only on proportion of total time.

Comprehension.

The mean proportion of time spent in each strategic category, the associated baseline proportion, and its individual confidence interval for the seniors and novices appear in Figures 5 and 6, respectively. The seniors and novices look similar, but one critical difference exists. Whereas both groups show proportions of time significantly above that expected by chance (the baseline) for Categories BD, BS, and DS, only the seniors exhibit a significant proportion of time consistent with a "pure" depth-first strategy (Category D). The novices exhibit a proportion of time consistent with a depth-first strategy that falls just below the upper confidence limit. It should be noted that the proportion of moves consistent with a depth-first strategy and the proportion of moves consistent with a sequential strategy do exceed their respective upper confidence limits. Also worth noting is the fact that the proportion of time consistent with a "pure" breadth-first strategy (Category B) falls significantly below the lower

confidence limit for both training groups.

Examination of individual subject data also indicates differences between the two training groups. Whereas 19 of the 20 seniors had an above-chance proportion of time consistent with a depth-first strategy, 5 of the 9 novice subjects exhibited proportions within chance levels. Additionally, 3 of these 5 novice subjects also exhibit proportions of time consistent with a sequential strategy which are above the upper confidence limit. From these results, it appears that the seniors are using a depth-first strategy more consistently than the novices, who seem to be split between the depth-first and sequential strategies.

The reader may be somewhat confused by the high observed proportions in the first two categories (breadth-first intersection depth-first and breadth-first intersection sequential) given that the proportion consistent with a "pure" breadth-first strategy is so low. This question suggested yet another method of summarizing the data. If the proportions consistent with these two categories really do represent predominately one of the two strategies, it would be desirable to determine which of the two strategies makes predictions closer to the observed proportions.

Table 4 shows "predicted" and observed proportions for six of the eight categories (excluding comments/global declarations and the uninterpretable category). The first three lines in the table are the proportions predicted by each of the three strategies for each of the six categories. These expected proportions are calculated by dividing the number of moves (or "cells", if the reader prefers) contained in the category by the total number of moves consistent with one of the "pure" strategies. For instance, the proportion predicted for the breadth-first intersection depth-first category by a "pure" breadth-first strategy would be 1/6, since there is one move in the category (8 -> 4) and six moves consistent with a breadth-first strategy (see Figure 1). The rationale for this calculation is as follows. If the subjects are using a pure strategy, it is assumed that they spend time making only moves that are consistent with that strategy, and spend 1/(number of moves) amount of time in making each of the moves. Thus, the proportion of time in any category should equal the number of moves in the category divided by the number of moves consistent with the strategy.

The fourth row of predicted proportions (labeled "speeded depth-first") requires some explanation. An examination of Figure 1, will indicate that the depth-first

path has been quite narrowly defined, and may not apply with smaller programs such as the present one. During the experiment, it appeared as though subjects were using a more or less depth-first strategy during comprehension, but it was a modified version of that depicted in Figure 1. Figure 7 depicts an intuitively reasonable "speeded" version of the depth-first strategy. Module number 4 (Figure 7) is a relatively simple module, (see Table 3 and Appendix B) and subjects have the hierarchy to aid them in remembering the structure of the program. This implies that there should be little need to return to module 4 to retrieve the name of module 3, or to refresh one's memory about what module 4's function is before returning to the main program. Similar reasoning applies for the right half of the hierarchy, except that there is a small amount of code after the calls to modules 5 and 6 in module 7, hence the inclusion of the move from module 6 to module 7.

Table 4

Expected and Normalized Observed Proportions in the Comprehension Task For Six of the Eight Strategic Categories

Strategy		Category BD	BS	B	DS	D	S
Breadth-first		.17	.33	.50	.00	.00	.00
Depth-first		.08	.00	.00	.50	.42	.00
Sequential		.00	.14	.00	.43	.00	.43
Speeded Depth-first		.14	.29	.00	.29	.29	.00
		Observed					
Seniors	S1	.17	.25	.08	.21	.29	.00
	S2	.08	.23	.00	.17	.37	.15
	S3	.11	.29	.06	.19	.29	.06
	S4	.11	.07	.02	.37	.39	.04
	S5	.13	.25	.11	.16	.33	.02
	S6	.09	.20	.00	.25	.42	.04
	S7	.10	.14	.11	.22	.40	.03
	S8	.12	.23	.00	.12	.49	.05
	S9	.13	.23	.02	.19	.33	.10
	S10	.13	.30	.02	.08	.45	.02
	S11	.12	.28	.00	.29	.29	.01
	S12	.07	.09	.00	.47	.21	.16
	S13	.12	.21	.00	.18	.26	.23
	S14	.05	.11	.06	.37	.39	.02
	S15	.05	.17	.01	.41	.26	.10
	S16	.05	.24	.00	.36	.09	.26
	S17	.13	.39	.00	.18	.26	.03
	S18	.24	.19	.00	.13	.31	.13
	S19	.07	.11	.07	.20	.24	.31
	S20	.09	.08	.00	.49	.26	.08
	Mean	.11	.20	.03	.25	.32	.09
Novices	S1	.11	.09	.00	.47	.18	.16
	S2	.08	.12	.02	.25	.34	.19
	S3	.14	.16	.00	.29	.37	.04
	S4	.07	.12	.03	.17	.47	.14
	S5	.13	.22	.00	.16	.13	.35
	S6	.00	.37	.00	.33	.00	.29
	S7	.11	.14	.00	.37	.28	.11
	S8	.00	.21	.00	.39	.10	.30
	S9	.07	.09	.00	.66	.09	.09
	Mean	.08	.17	.01	.34	.22	.18

Calculation of speeded depth-first predictions is somewhat different, since it was not one of the original "pure" strategies. It is necessary to divide the number of moves in the intersection of speeded depth-first and the category by 7. Only seven of the eight speeded depth-first moves are contained in the six categories of interest while the eighth is contained in the Category "Other".

The rest of the entries in Table 4 are the observed proportions for both training groups. Some of the rows do not sum to exactly 1.00 due to rounding error. These entries were "normalized" to adjust for the removal of the proportions due to moves to and from the comments/global declarations and uninterpretable moves. This was accomplished by dividing the proportions for the six remaining categories by (1.0 - the sum of the proportions in the two excluded categories) for each subject and then calculating a mean proportion. This is appropriate, since the predictions mentioned above are based on the ideal assumption that all of the time spent comprehending the program can be interpreted on the basis of one of the three strategies alone.

Table 4 makes possible a preliminary classification of individual subjects into "strategy groups" based on

their pattern of proportions. For example, most of the seniors appear to fit the speeded-depth predictions better than the predictions from the other three strategies. However, subjects 4 and 20 (seniors) appear to fit the "pure" depth-first predictions more closely, while subject 16 fits the sequential predictions.

The breadthfirst strategy does not have much support in the data in table 4. Looking at the group mean observed proportions for the two groups, it is obvious that the alternative strategies (to breadth-first) make more accurate predictions for the BD and BS categories than does a breadth-first strategy. The mean proportion for the intersection of breadth-first and depth-first for both groups is more in line with the prediction from "pure" depth-first than from "pure" breadth-first. Likewise, the mean proportion for the intersection of breadth-first and sequential for both groups is closer to that predicted by "pure" sequential.

Further, the expected proportions (on the basis of a depth-first strategy) for the "pure" depth-first and sequential categories are more in line with the seniors' group means than the novices' group means. Also, the group mean proportion for the novice group in the category consistent with the intersection of the depth-first and

sequential strategies seems closer to the value expected (on the basis of a sequential strategy) than does the mean for the seniors.

Interestingly, the speeded depth-first predictions seem to mirror the seniors' group means even better than the "pure" depth-first predictions, particularly for the intersection of depth-first and sequential and "pure" depth-first. The only troublesome category (for speeded depth-first) seems to be the intersection of breadth-first and sequential, where a "pure" sequential strategy makes the best prediction. However, when one considers the overall "fit", it looks like the seniors were following a path consistent with a speeded depth-first strategy. These predictions do not seem to be as close to the observed means for the novices. The mean observed proportions for the novices in the sequential and depth-first intersection sequential categories are quite a bit higher than the speeded depth-first predictions, but not nearly to the same degree for the seniors. Also, the novices' means for the breadth-first intersection depth-first and breadth-first intersection sequential are somewhat lower relative to the predictions than the seniors' means.

Debugging.

The mean proportion of time spent in each strategic category, the associated baseline proportion, and its individual confidence interval for the seniors and novices appear in Figures 8 and 9, respectively. Of the three strategic categories of interest (breadth-first, depth-first, and sequential) only the proportion of time consistent with a depth-first strategy reaches significance, and then, only for the seniors. The proportion of time consistent with a depth-first strategy for the novices falls just below the upper confidence limit. Again, it should be noted that the proportion of moves consistent with a depth-first strategy for the novices is significantly above chance.

Examination of individual subject data reveals less consistency than that found in the comprehension results. Of the twenty seniors, nine exhibit proportions of time consistent with a depth-first strategy which are not above chance levels. Two of these nine subjects have proportions of time consistent with a breadth-first strategy which are above the chance limit, while one has a sequential proportion which is above chance. Four of the nine novices

exhibit proportions of time consistent with a depth-first strategy which are not significantly above chance. Of these four subjects, one showed a breadth-first proportion above chance, and one showed a sequential proportion above chance. Although the debugging behavior is not as easy to categorize as the comprehension behavior, the depth-first strategy seems to predominate.

Examination of the data in Table 5 suggests that this may be an oversimplification. For the seniors, the observed proportions in the first two categories (breadth-first intersection depth-first and breadth-first intersection sequential) are most consistent with the proportion predicted by a breadth-first strategy. This is also the case for the novices group in the first category (breadth-first intersection depth-first). These results would suggest that there may be a substantial breadth-first component in subjects' debugging behavior. However, the speeded depth-first predictions come reasonably close to both groups' means, suggesting that something more like a depth-first strategy is being employed.

Quite probably, subjects are making some use of strategies specific to debugging (e.g., searching for all occurences of a given variable), even in this "impoverished" debugging task. These sorts of strategies

would probably fall primarily into the comments/global declarations and uninterpretable categories. Consequently, conclusions about debugging behavior based on these debugging data should be undertaken with some caution.

Table 5

Expected and Normalized Observed Proportions in the Debugging Task For Six of the Eight Strategic Categories

Strategy		Category BD	BS	B	DS	D	S
Breadth-first		.17	.33	.50	.00	.00	.00
Depth-first		.08	.00	.00	.50	.42	.00
Sequential		.00	.14	.00	.43	.00	.43
Speeded Depth-first		.14	.29	.00	.29	.29	.00
		Observed					
Seniors	S1	.36	.00	.64	.00	.00	.00
	S2	.17	.37	.00	.03	.44	.00
	S3	.10	.07	.00	.59	.25	.00
	S4	.86	.00	.00	.00	.14	.00
	S5	.00	.41	.00	.07	.35	.17
	S6	.08	.01	.05	.76	.10	.00
	S7	.12	.00	.00	.68	.20	.00
	S8	.50	.29	.00	.00	.21	.00
	S9	.30	.30	.00	.00	.41	.00
	S10	.12	.61	.00	.00	.27	.00
	S11	.38	.44	.00	.00	.18	.00
	S12	.17	.49	.00	.00	.34	.00
	S13	.33	.44	.00	.00	.22	.00
	S14	.42	.40	.00	.00	.16	.02
	S15	.07	.53	.00	.21	.13	.06
	S16	.04	.35	.04	.31	.10	.16
	S17	.10	.31	.12	.20	.27	.00
	S18	.16	.00	.00	.00	.24	.60
	S19	.09	.29	.12	.26	.16	.09
	S20	.20	.26	.00	.19	.35	.00
	Mean	.23	.28	.05	.16	.23	.05
Novices	S1	.20	.00	.00	.53	.21	.06
	S2	.09	.02	.00	.52	.38	.00
	S3	.28	.46	.00	.00	.26	.00
	S4	.09	.21	.00	.23	.47	.00
	S5	.64	.14	.00	.00	.21	.00
	S6	.00	.12	.01	.71	.07	.10
	S7	.09	.01	.19	.46	.25	.00
	S8	.13	.26	.00	.34	.02	.25
	S9	.00	.19	.00	.58	.02	.21
	Mean	.17	.16	.02	.37	.21	.07

Training and Task Differences.

In an effort to determine whether there was an effect of training level, task (comprehension vs. debugging), or an interaction of either of these factors with strategic category, a 2 (training) X 2 (task) X 3 (category) ANOVA was run on the proportion of total time consistent with three of the eight strategic categories. The categories included in the analysis were depth-first (D), sequential (S), and the intersection of depth-first and sequential (DS). There was a significant main effect of category, $F(2,162) = 17.60$, $MSE = 0.01161$, and a significant interaction of training level with category, $F(2,162) = 4.57$, $MSE = 0.01161$. The mean proportions of time consistent with each of the three categories for both training groups appear in Figures 10 (comprehension) and 11 (debugging). The mean proportions for both tasks appear in Figures 12 (seniors) and 13 (novices).

In the comprehension task, the proportion of time spent in the three categories was dependent on training level. Although the proportion of time for the category DS does not appear to be very different across training groups, the proportions for categories D and S are more divergent (Figure 10). Specifically, the seniors spent more time making moves consistent with a depth-first

strategy (D) than the novices, while spending less time making moves consistent with the sequential (S) strategy. In the debugging task, the pattern is quite different (Figure 11). While there was a large difference in the proportion of time spent in category DS across training level, there was little difference in the proportion of time spent in categories D and S across training level.

Selected repeated measures t-tests were performed to determine (within training level and task) exactly which proportions produced the differences. Because of the inflated Type I error probability associated with this approach, it was decided that an alpha level of .01 would be used to evaluate the significance of the obtained t-values. Reference to Figure 10 and 11 should help to make the choice of comparisons clear. It should also be noted that the three categories have associated baselines which are unequal (see Figure 5), hence one might expect differences among the proportions whether there is an effect present or not. To keep this in perspective, though, I should point out that the baselines differ only slightly, and not enough to produce the results reported below.

For the seniors in the comprehension task, there was a significant difference between the depth-first and

sequential proportions, $t(19) = 5.97$, $p < .001$. There was also a significant difference between the proportion consistent with the intersection of depth-first and sequential strategies and the sequential proportion, $t(19) = 4.887$, $p < .001$, but no difference between the proportion consistent with the intersection and the depth-first proportion, $t(19) = -1.1106$, $p > .05$. For the novices in the comprehension task, there was no significant difference between the proportion consistent with the intersection and either the depth-first or the sequential proportion, $t(8) = 1.52$, $p > .05$, and $t(8) = 2.12$, $p > .05$, respectively. Again, it appears that the seniors follow a primarily depth-first path through the program, while the novices use a mixture of depth-first and sequential strategies.

Between-groups contrasts for the comprehension task revealed a significant difference in the proportion of depth-first time, $F(1,27) = 4.728$, $p < .05$. Both the intersection and sequential between-group differences were not significant, $F(1,27) = 2.00$, $p > .05$, and $F(1,27) = 3.97$, $p > .05$, respectively. While the seniors spend more time engaged in a depth-first strategy than the novices, the two groups do not differ with regard to the proportion of time engaged in a sequential strategy. However, the difference in sequential proportions is suggestive, and

perhaps with more subjects, would reach significance.

For the seniors in the debugging task, there was a significant difference between the depth-first and sequential proportions, $t(19) = 4.24$, $p < .001$, but no significant difference between the intersection and sequential proportions. For the novices, there was no significant difference between the depth-first and sequential proportions, or between the intersection of depth-first and sequential and "pure" depth-first. There was, however, a significant difference between the intersection of depth-first and sequential and "pure" sequential, $t(8) = 3.38$, $p < .01$. So, while the seniors spent significantly more time following a depth-first path than a sequential path during debugging, the novices did not. There was little difference between the two groups for the depth-first and sequential categories (Figure 11), but there was a significant difference for the intersection of depth-first and sequential, $F(1,27) = 7.39$, $p < .01$.

Strategy and debugging time.

In an attempt to assess the relationship between a subject's distribution among the three strategic categories of interest (depth-first, sequential, and their

intersection) and their debugging time, two separate stepwise multiple regression analyses were performed, with debugging time (DEBUG) as the criterion variable. In the first analysis, the predictor variables were the proportion of time consistent with each of the three categories for the comprehension task. The second analysis used proportion of time consistent with the three categories (D, S, DS) for the debugging task as predictor variables. Table 6 presents the correlation matrix, means, and standard deviations for the criterion and predictor variables.

The first solution (predicting debugging time from comprehension strategy) accounted for only 16 percent of the variance in the debugging time, $R2 = .16$, $F(1,27) = 5.03$, $p < 0.05$. One variable (proportion of time spent in a sequential comprehension strategy) was entered into the equation. The second solution (predicting debugging time from debugging strategy) also only accounted for 16 percent of the variance in debugging time, $R2 = .16$, $F(1,27) = 5.00$, $p < .05$. Again, only one variable (proportion of time spent in a debugging strategy consistent with the intersection of the depth-first and sequential strategies) was entered into the equation.

Table 6

Correlations, Means, and Standard Deviations for the Criterion and Predictor Variables

		Predictor Variables					
		Comprehension			Debugging		
	DEBUG	DS	D	S	DS	D	S
DEBUG	1.00						
DS (Comp.)	.13	1.00					
D (Comp.)	-.25	-.38	1.00				
S (Comp.)	.40	.23	-.63	1.00			
DS (Debugging)	.40	.26	-.25	.19	1.00		
D (Debugging)	.17	-.19	.42	-.14	.01	1.00	
S (Debugging)	.11	.16	-.44	.37	.09	-.35	1.00
Mean	541.90	.177	.171	.077	.172	.128	.037
SD	439.31	.093	.072	.072	.211	.072	.074

Note - See text for explanation of variables. Debugging time is in seconds.

Besides the fact that those subjects spending more time engaged in a sequential comprehension strategy tend to take more time in the debugging task, there is evidence that subjects follow the same path through the program in the comprehension and debugging tasks. The correlation between the proportion of time engaged in a depth-first comprehension strategy and the proportion of time engaged in a depth-first debugging strategy was significant, r = time engaged in a sequential comprehension strategy and the proportion of time engaged in a sequential debugging

strategy was also significant, $r = .37$, $p < .05$.

Questionnaire data.

A stepwise multiple regression analysis was performed on the questionnaire data from all subjects from Experiments II and III (N = 41). This was considered legitimate, since there was no difference in debugging time across the two experiments. Experiment I questionnaire data were not included, as it is hypothesized that the debugging time is "contaminated" by comprehension processes. The predictor variables were number of programming languages known (NLANG), number of programming courses taken (NCOUR), GPA, and SAT. Table 7 presents the correlation matrix, means, and standard deviations for the criterion and predictor variables. The solution was not a particularly good one, multiple R2 = .29, $F(1,34) = 11.24$, $p < 0.01$ with only one variable, SAT, exceeding the criterion for inclusion in the equation.

Table 7

Correlations, Means, and Standard Deviations for Criterion and Predictor Variables

		Predictor Variables			
	DEBUG	NLANG	NCOUR	GPA	SAT
DEBUG	1.000				
NLANG	-.002	1.000			
NCOUR	-.206	.600	1.000		
GPA	-.230	.064	-.024	1.000	
SAT	-.518	.033	.058	.269	1.000
Mean	564.2	4.139	6.972	3.115	619.167
SD	477.71	1.376	3.783	.495	96.433

Note - See text for explanation of variables. Debugging time is in seconds.

C H A P T E R V

GENERAL DISCUSSION

It appears that subjects in the first study were doing something different than those in the second study. Since the only difference between the studies was the separation of the comprehension and debugging phases in the second study, the most reasonable conclusion is that subjects in the first study were doing something much more like debugging than comprehending. The fact that the group data from the debugging part of Experiment II are similar to the group data from Experiment I (see Figure 14) lends support to this conclusion. If the large proportion of sequential moves was only due to the set of commands the subject had to work with, one would expect, in the comprehension phase, to see only the sequential curve consistently above the upper confidence limit. The fact that the results of the comprehension phase of Experiment II differ from the results of Experiment I suggests that experiments using only a debugging task to measure comprehension may be measuring something else.

However, the overall tendency toward moves consistent

with a sequential strategy is probably related to the set of scrolling commands. Almost invariably, as a subject's proportion of time spent in breadth- or depth-first moves increases with increasing cutoff levels, there is a corresponding decrease in the proportion spent in sequential moves. This indicates that at least part of the sequential component is made up of short, transitional moves which are actually a "by-product" of another strategy. Further, it is difficult to say how many of the longer, "actual" sequential moves may have been encouraged by the characteristics of the scrolling program.

This line of reasoning is supported by the results of Experiment III. The proportion of time spent making moves consistent with a sequential strategy failed to reach significance at both training levels. Further, the proportion of moves consistent with a depth-first strategy did reach significance for the seniors, and almost reached significance for the novices.

Clearly, the display change from Experiment II to Experiment III made a difference in the comprehension task. Looking at Figure 15, it can be seen that the novices from Experiment III had a proportion of time in the depth-first category that was quite a bit higher than the corresponding asymptotic proportion from Experiment II. Since the

subjects in Experiment II are, for the most part, more experienced than the novices in Experiment III, this result argues against ability/experience differences as the primary explanation for the increase in depth-first behavior. That is not to say that experience makes no difference in these studies. As shown above, the two training groups in Experiment III differ with regard to comprehension strategy. It appears that, given the opportunity, most subjects choose to proceed depth-first through the program, and that more experienced subjects do so to a greater degree.

As indicated earlier, there is some difficulty in defining exactly what a bottom-up path through the hierarchy should be. It seems that whatever path one chooses, it is probably highly confounded with a sequential path (Figure 1). The relative lack of evidence for the use of the sequential strategy in Experiment III suggests that, even though the Pascal program is poorly documented, subjects did not use a bottom-up strategy.

Although there seem to be some differences in strategy usage among subjects, the proportions of time in categories consistent with depth-first intersection sequential, depth-first, and sequential were not especially good predictors of debugging time. The fact that the

proportion of time spent in the sequential category during comprehension is positively correlated with debugging time suggests that this strategy may not be a very good one. However, since this evidence is only correlational, there are other interpretations. For example, it could be the case that lower ability subjects tend to use a sequential comprehension strategy and also tend to be less efficient when debugging.

The relationship between proportion of time spent in the intersection of depth-first and sequential in the debugging task and debugging time is not so clear, but still seems interpretable. The moves in this category are (see Figure 1) 3 -> 4, 4 -> 3, 6 -> 7, 7 -> 6, 8 -> 7, and 7 -> 8. Since the main program (module #8) consists of two procedure calls, subjects aren't going to be making many moves or spending much time making moves to and from module #8. This leaves four moves to be considered. Since the bug is placed in module #7 after the call to module #6, (recall the discussion of bug placement in the Method section of Experiment I) a subject spending lots of time moving to and/or from this module is probably looking in the wrong place. Similarly, a subject spending a lot of time looking in module numbers 3 and 4 is definitely looking in the wrong place. So, more time spent making these four moves means a longer debugging time.

Although there was a reduction in the sequential component in Experiment III relative to Experiment II, it is difficult to ascertain whether the reduction is due to: (1) the new scrolling mode allowing the "hidden" strategies to emerge, or (2) to the fact that subjects in Experiment III had an informational advantage over those in the previous experiments. Subjects in Experiments I and II had to abstract the hierarchy of modules, whereas subjects in Experiment III were given the hierarchy. Perhaps once a subject has this hierarchy (either from abstraction or from having it given to him/her), it becomes easier to proceed through the program in a depth-first or breadth-first manner. If this "abstraction" was occurring in Experiments I and II, perhaps it accounts for the large sequential component in the comprehension phase of Experiment II and whatever part of the results of Experiment I was due to comprehension processes.

One possible continuation of the proposed study would be to force subjects to take a particular path through the program for one full pass, with their study time controlled by the display program. This method provides a better test than the first three studies, since subjects spend the same amount of time comprehending the program regardless of which strategy is used. Alternatively, subjects could be

allowed to proceed through the program at their own pace, and study times for each module could be measured. In either case, the strategy yielding the best comprehension should also result in shorter debugging times.

Unfortunately, there would be at least one problem with interpretation of the results of such studies. Inferior performance on the debugging task by subjects forced to use a particular strategy could indicate (1) that the strategy is not the best one for comprehension, or (2) that subjects have difficulty comprehending when forced to follow a path other than their usual path (i.e., depth-first, for most subjects) through a program. Even if subjects were self-paced, this interpretational difficulty would still exist.

Assuming an "optimal" strategy is found, one could ask what characteristics of the code make it easier or harder to use that strategy efficiently. For example, would mnemonic variable and procedure names and extensive in-line comments make it easier to go through a program breadth-first? If subjects have a better idea of the function of lower-level procedures without actually visiting them, it should be easier to comprehend the current level.

The great variety of programming tasks, styles, etc. underscores the need for some sort of theory of program comprehension which will make useful predictions for situations not yet empirically tested. If we attempted to investigate every possible commenting, indenting, and mnemonic style, combined with different sized programs and different algorithms, we would find ourselves in the laboratory for quite some time. Turner (1984) has taken a step in the direction toward such a theory by demonstrating that at least some of the principles that apply in text comprehension also apply in program comprehension. Using the Kintsch and van Dijk (1978) model of text comprehension, Turner has demonstrated one such principle, the "levels effect", in a task involving recall of Pascal programs. Program statements whose corresponding propositions are lower in the propositional hierarchy were not recalled as well as those higher in the hierarchy. Such findings suggest that theory and research on prose comprehension may offer some explanatory power when applied to program comprehension.

I think that these studies represent a successful attempt to uncover the strategies that skilled programmers use when comprehending a computer program. This should make it possible to determine whether these strategies are indeed the best ones, and also what sorts of variations in

the code make particular strategies more or less effective. Now that we have some idea what experts do, we can ask whether what they do is consistent with predictions from psychological theory about what should be optimal.

BIBLIOGRAPHY

Atwood, M. E. and Ramsey, R. H. (1978). Cognitive structures in the comprehension and memory of computer programs: An investigation of computer program debugging. ARI Technical Report, 1978.

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. International Journal of Man-Machine Studies, 18, 543-554.

Gould, J. D. and Drongowski, P. (1974). An exploratory study of computer program debugging. Human Factors, 16, 258-277.

Kintsch, W. (1974). The Representation of Meaning in Memory. Hillsdale, N.J.: Lawrence Erlbaum Associates.

Kintsch, W. and van Dijk, T. A. (1978). Toward a model of text comprehension and production. Psychological Review, 85, 363-394.

Linger, R. C., Mills, H. D., and Witt, B. I. (1979). Structured Programming: Theory and Practice. Reading, Mass: Addison-Wesley.

Schneider, M. G., Weingart, S. W., and Perlman, D. M. (1978). An Introduction to Programming and Problem Solving With Pascal. New York: John Wiley and Sons, Inc.

Sheppard, S. B., and Love, T. (1979). Modern coding practices and programmer performance. Computer, 12, 41-49.

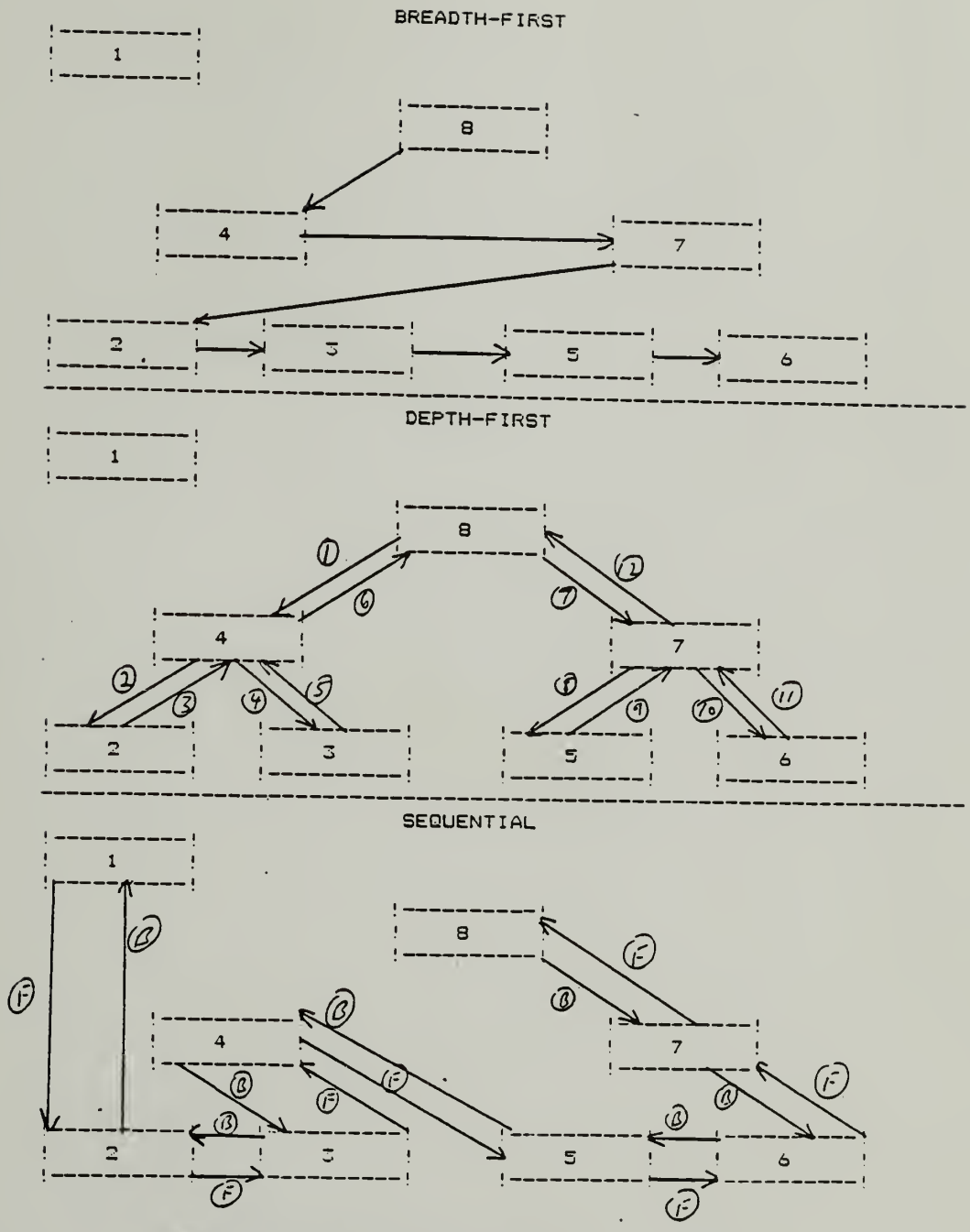
Shneiderman, B. (1980). Software Psychology. Cambridge, MA: Winthrop.

Turner, A. (1984). Recall and coding as measures of program comprehension. Unpublished Doctoral Dissertation, University of Colorado.

APPENDIX A

Figure 1

Hierarchy of modules for all experiments (numbers indicate declaration order) illustrating three possible comprehension strategies.



F = forward sequential, B = backward sequential

Figure 2

Proportion of total study time spent in the eight categories in Experiment I.

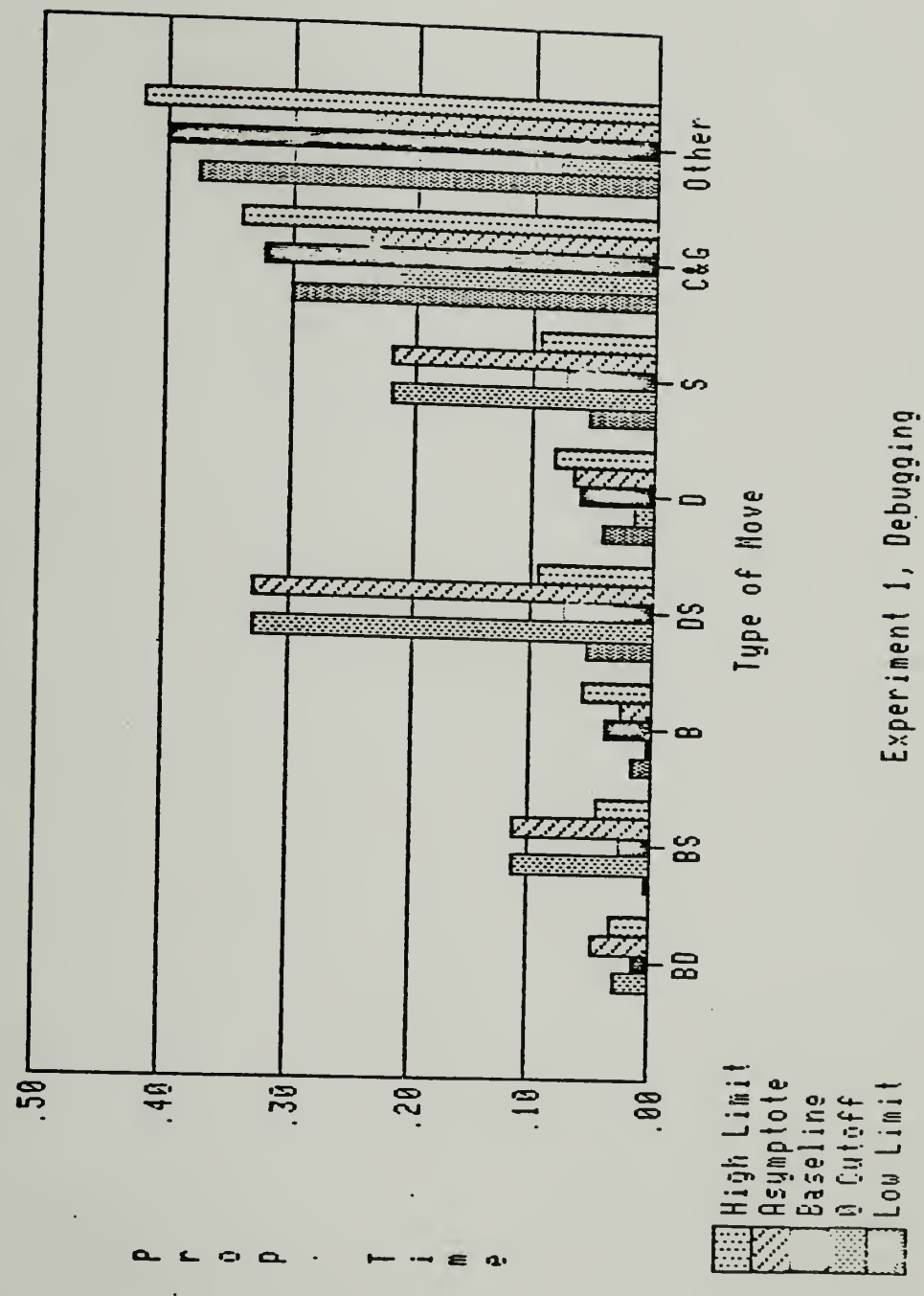
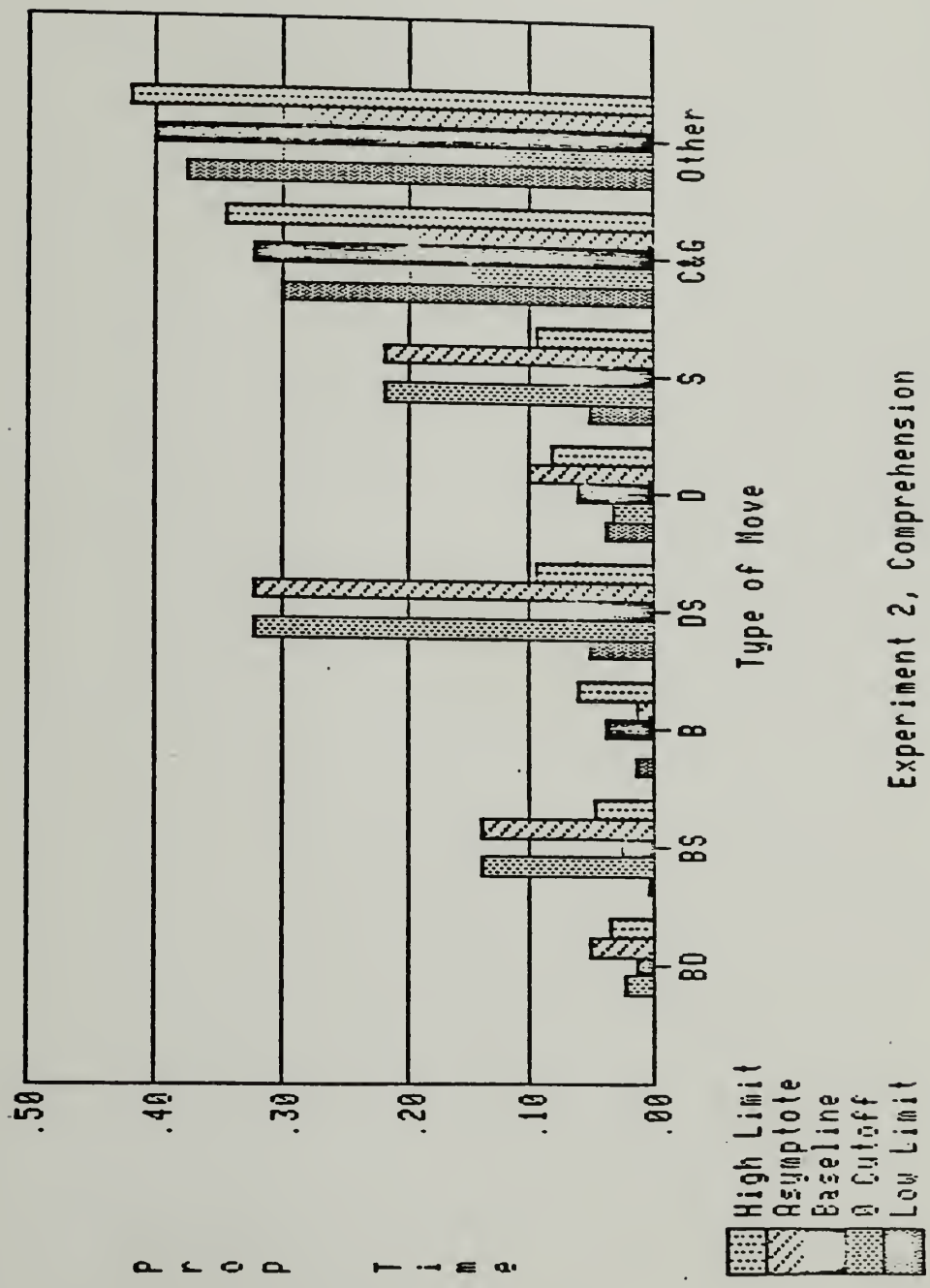

Prop. Time
.50
.40
.30
.20
.10
.00
BD
BS
B
DS
D
S
C&G
Other
Type of Move
High Limit
Asymptote
Baseline
Cutoff
Low Limit
Experiment 1, Debugging

Figure 3

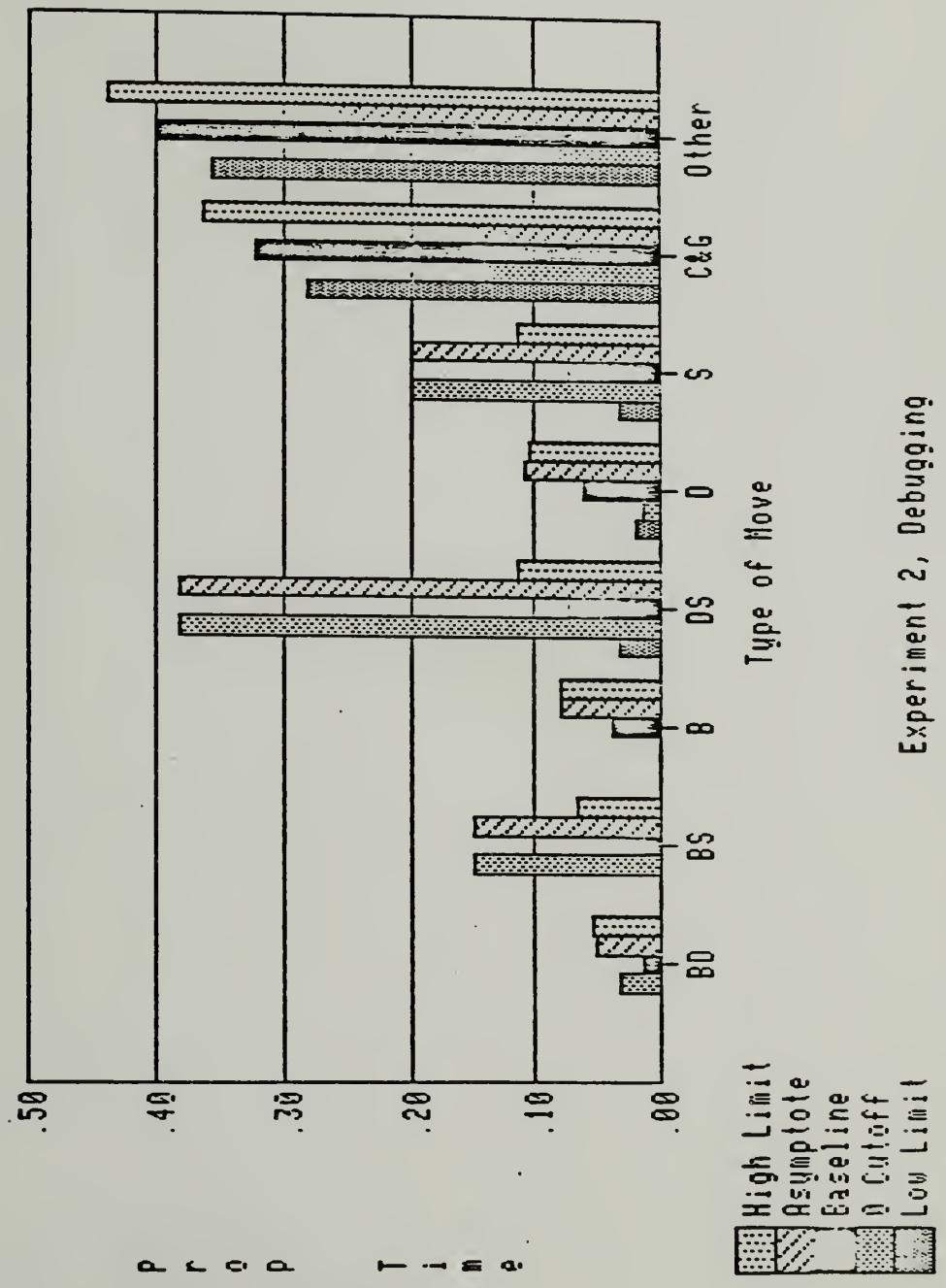
Proportion of total study time spent in the eight categories for the comprehension task in Experiment II.



Experiment 2, Comprehension

Figure 4

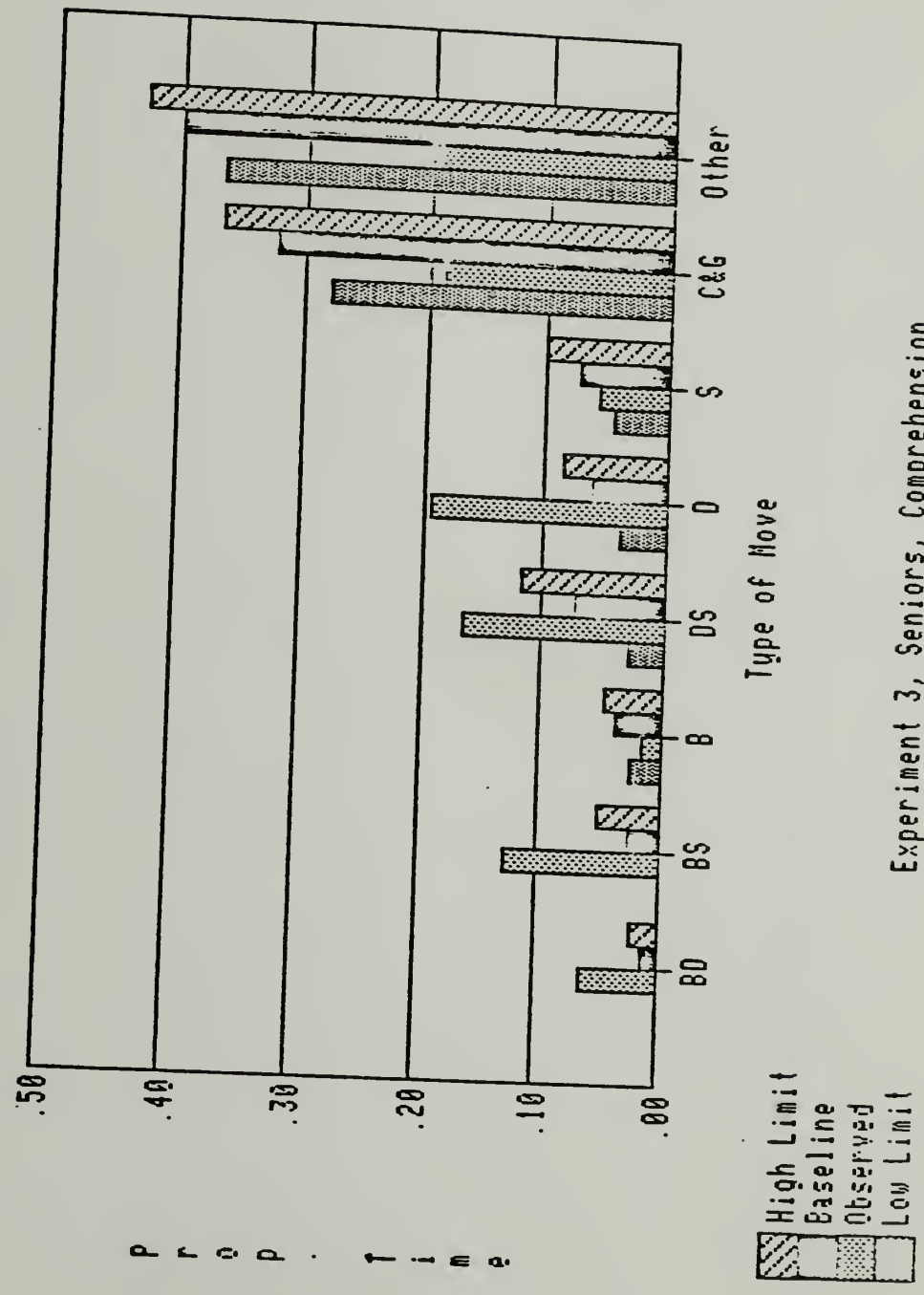
Proportion of total study time spent in the eight categories for the debugging task in Experiment II.



Experiment 2, Debugging

Figure 5

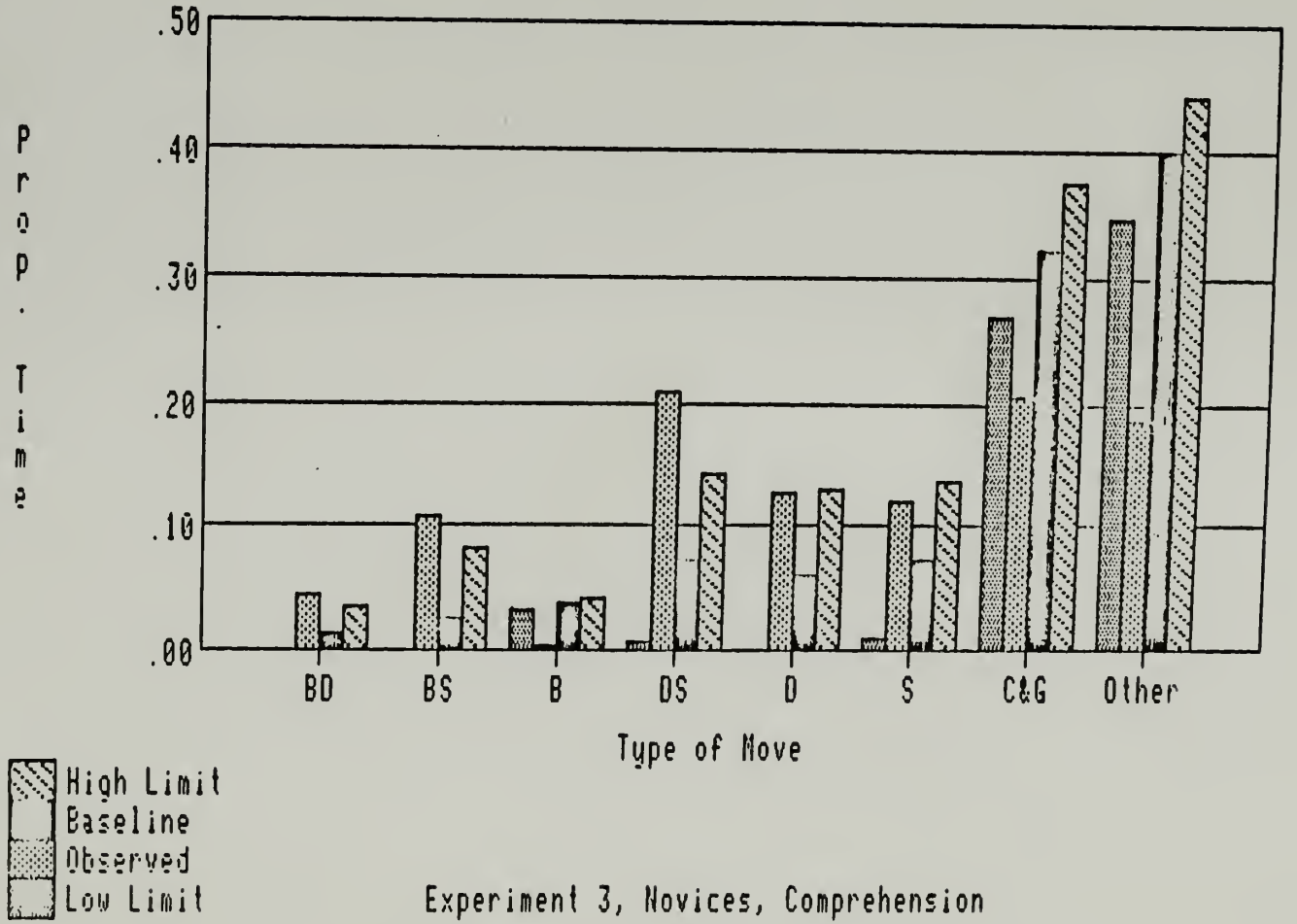
Proportion of total study time spent in the eight categories for the seniors in the comprehension task in Experiment III.



Experiment 3, Seniors, Comprehension

Figure 6

Proportion of total study time spent in the eight categories for the novices in the comprehension task in Experiment III.



Experiment 3, Novices, Comprehension

Figure 7

Hierarchy of modules depicting a speeded depth-first traversal.

SPEEDED DEPTH-FIRST

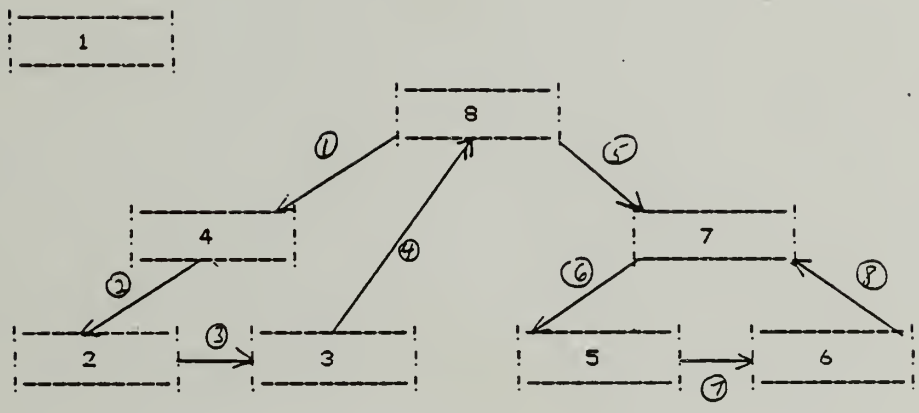


Figure 8

Proportion of total study time spent in the eight categories for the seniors in the debugging task in Experiment III.

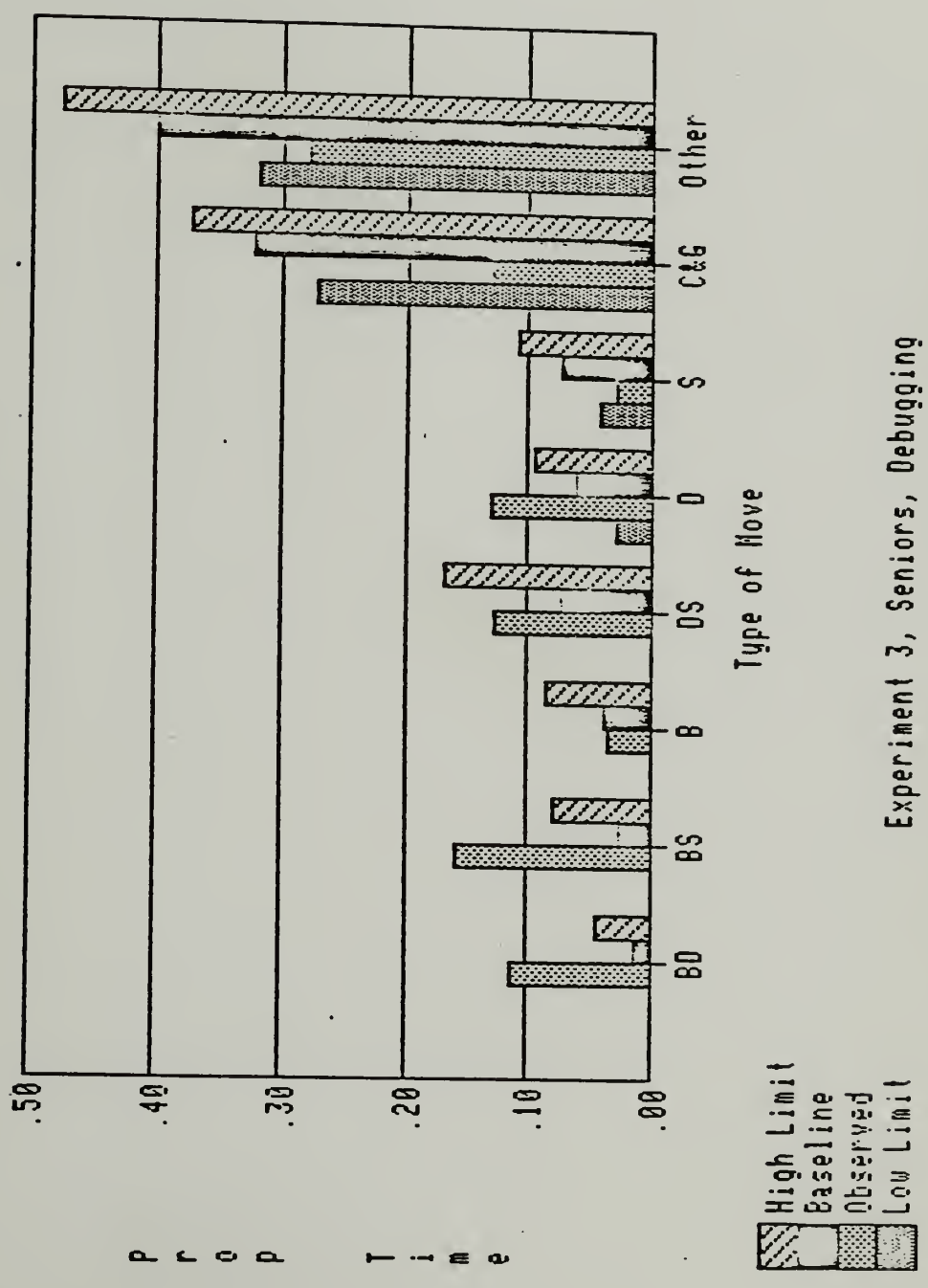
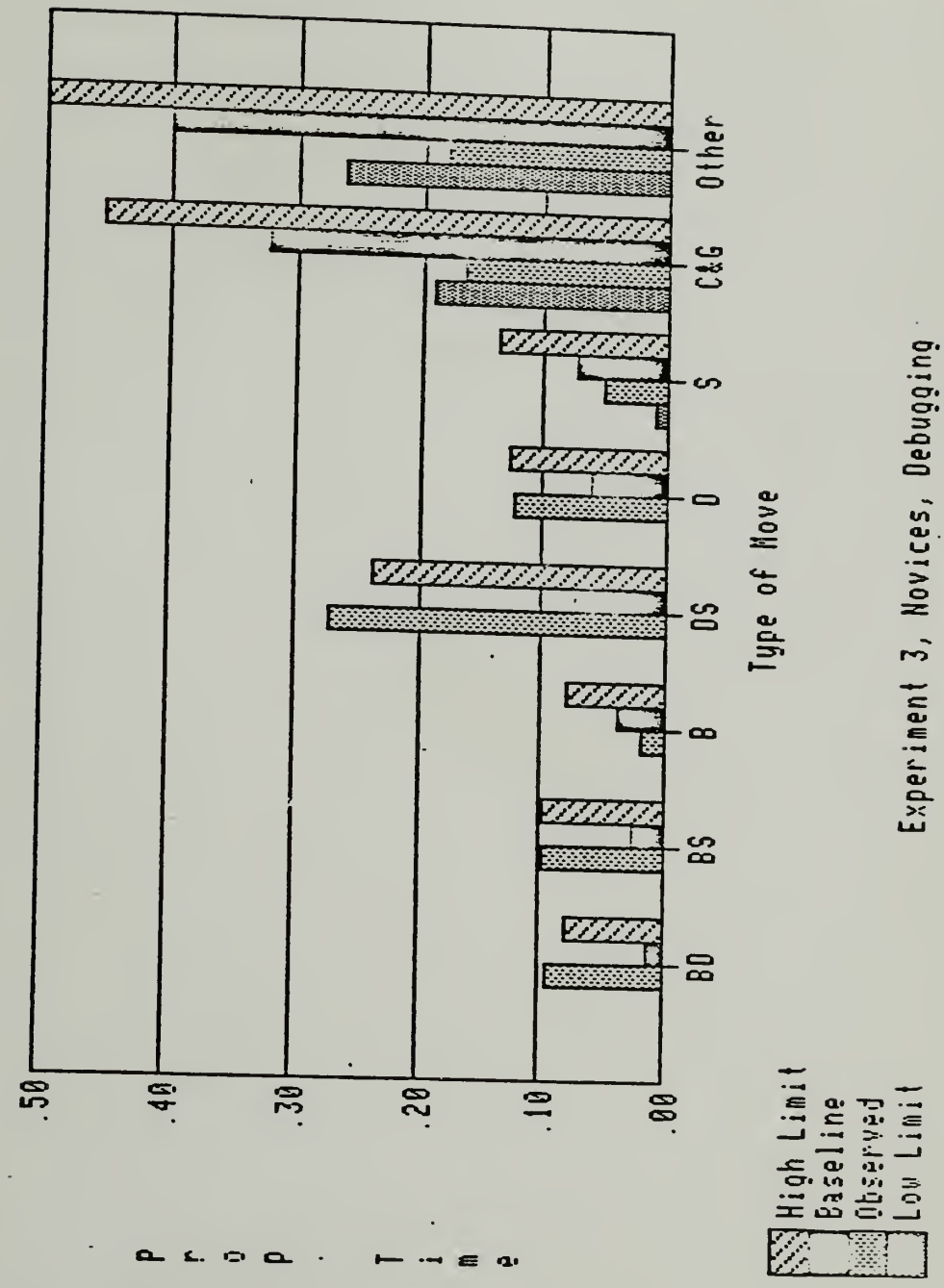

Prop Time
.50
.40
.30
.20
.10
.00
BD
BS
B
DS
D
S
C&G
Other
Type of Move
High Limit
Baseline
Observed
Low Limit
Experiment 3, Seniors, Debugging

Figure 9

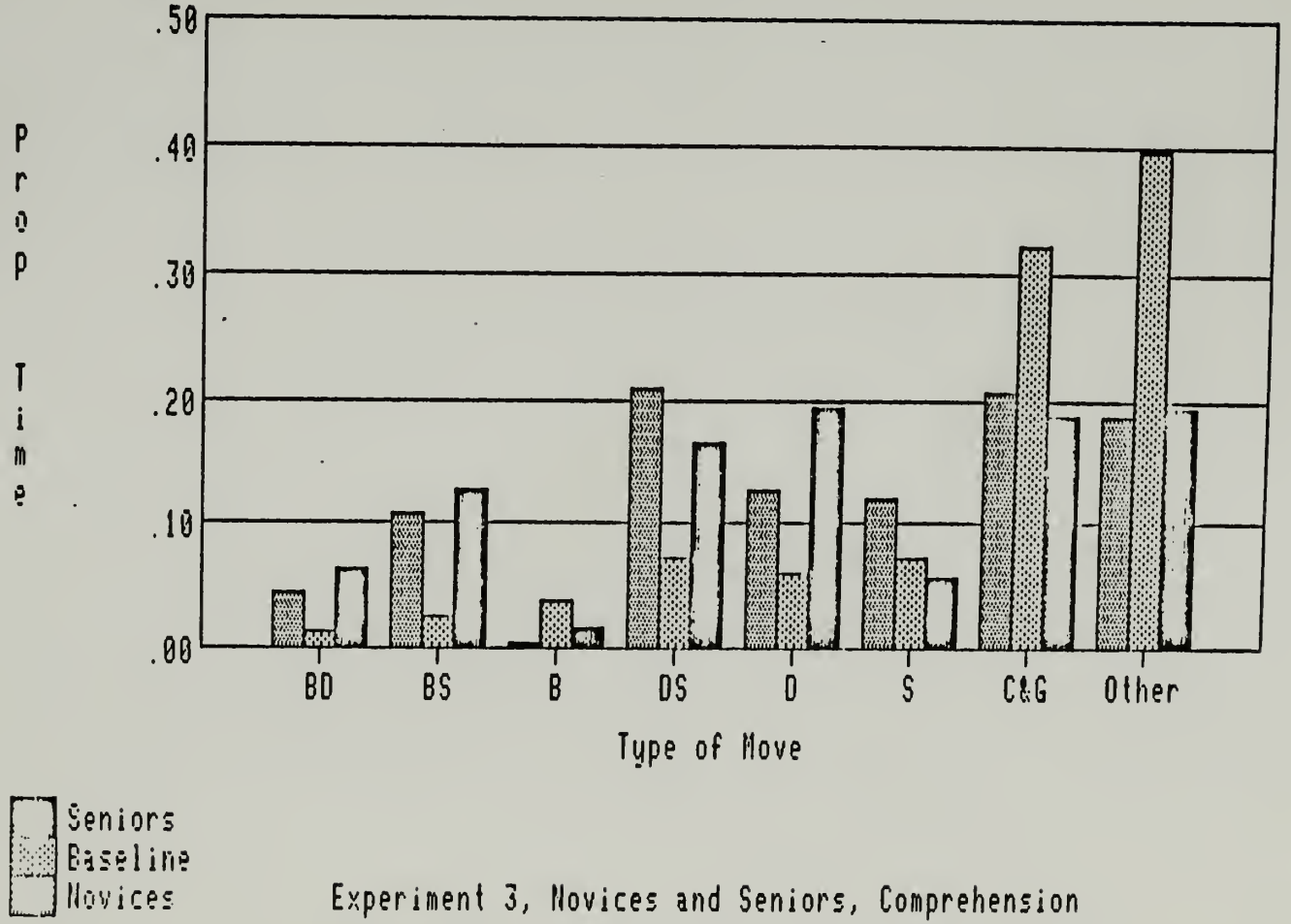
Proportion of total study time spent in the eight categories for the novices in the debugging task in Experiment III.



Experiment 3, Novices, Debugging

Figure 10

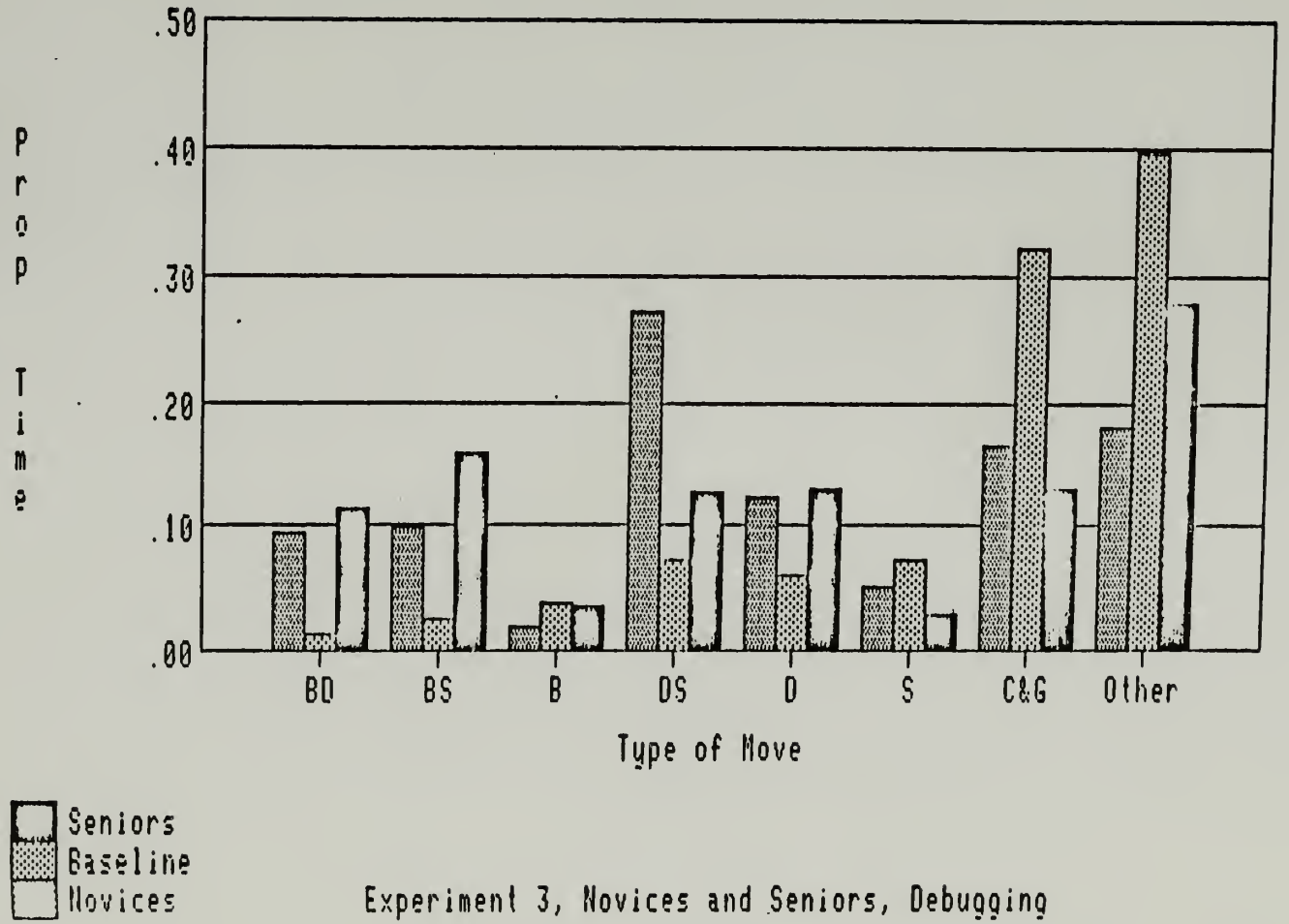
Proportion of total study time spent in the eight categories for the novices and seniors in the comprehension task in Experiment III.



Experiment 3, Novices and Seniors, Comprehension

Figure 11

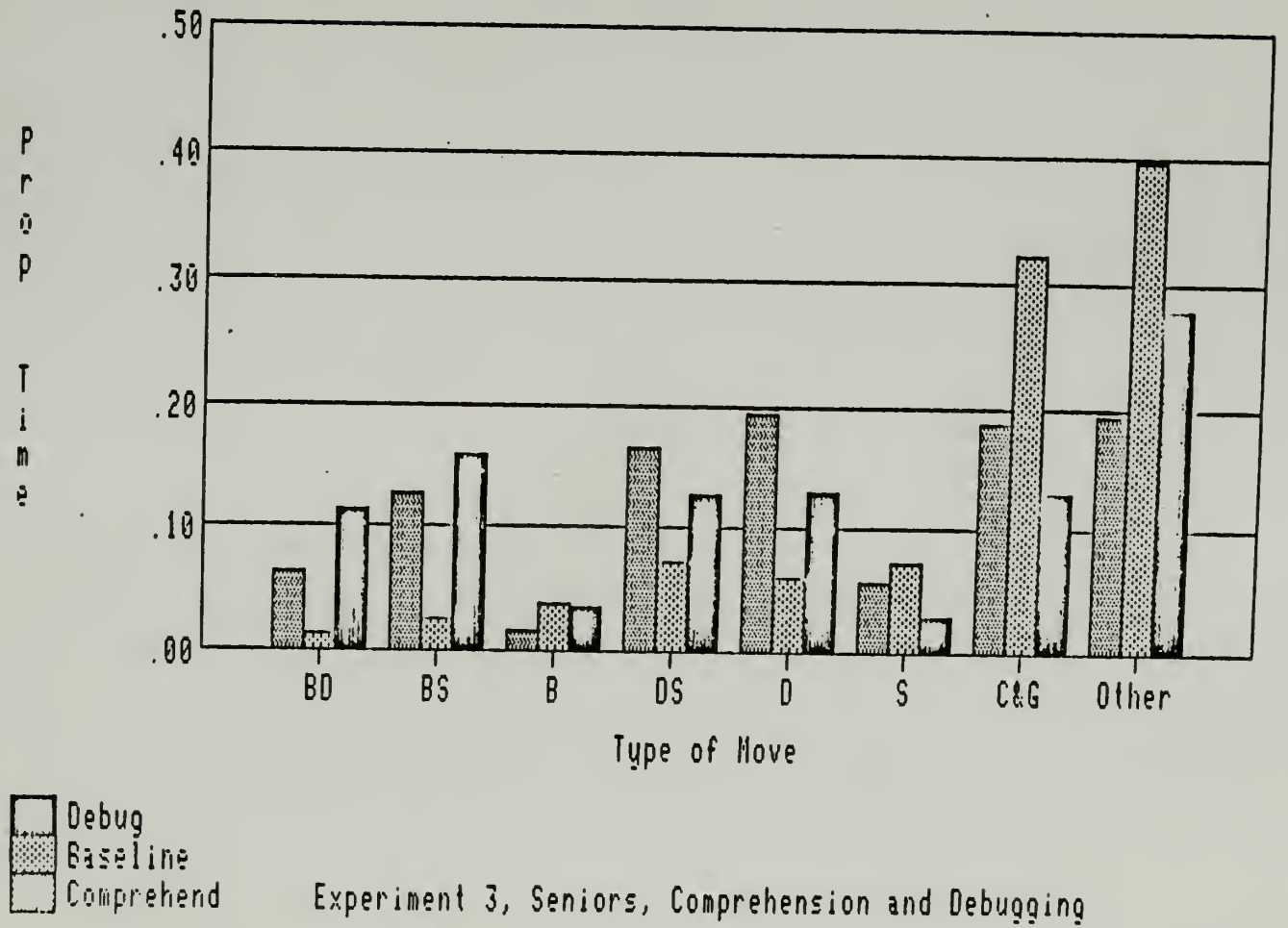
Proportion of total study time spent in the eight categories for the novices and seniors in the debugging task in Experiment III.



Experiment 3, Novices and Seniors, Debugging

Figure 12

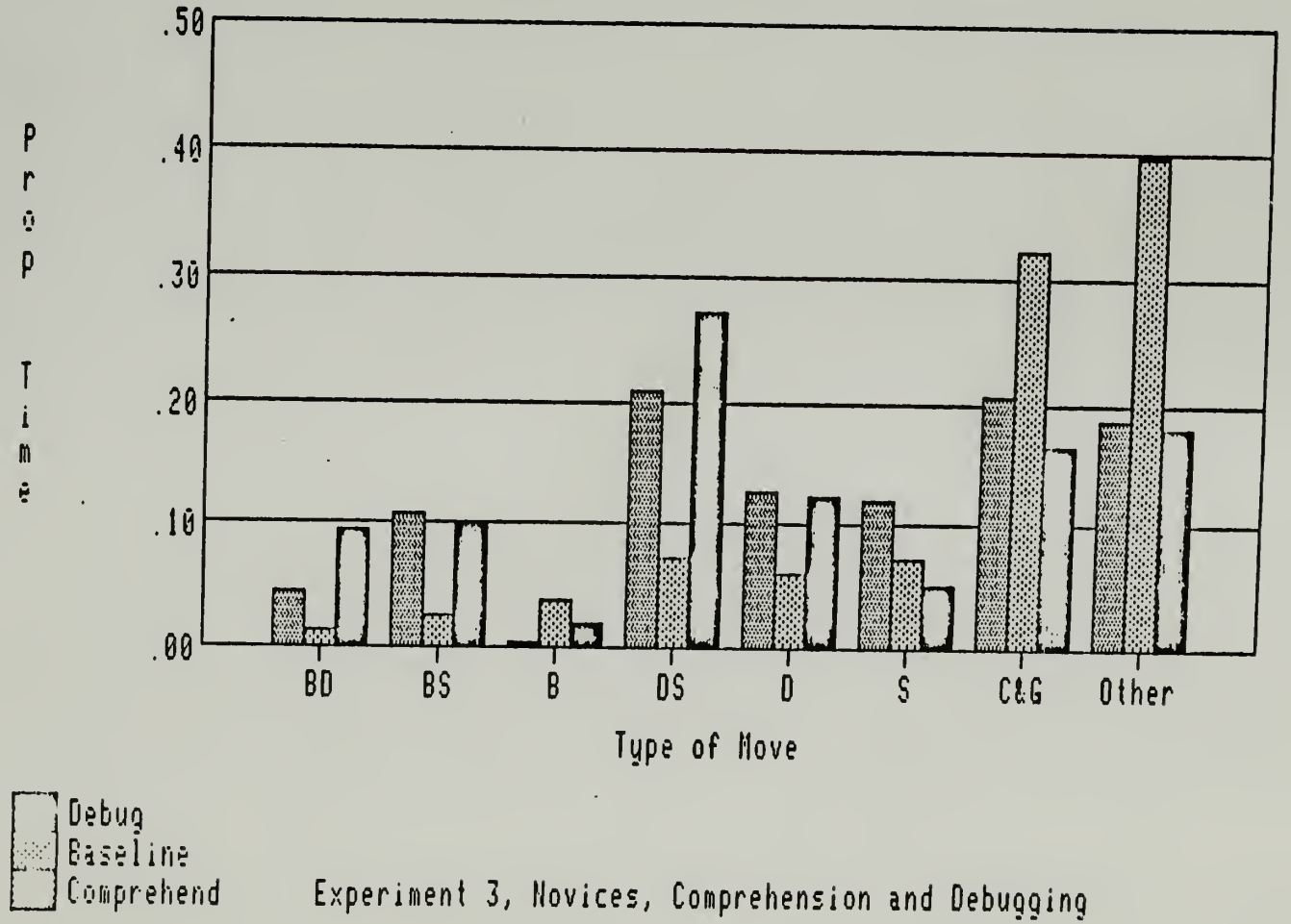
Proportion of total study time spent in the eight categories for the seniors in both tasks in Experiment III.



Experiment 3, Seniors, Comprehension and Debugging

Figure 13

Proportion of total study time spent in the eight categories for the novices in both tasks in Experiment III.



Experiment 3, Novices, Comprehension and Debugging

Figure 14

Proportion of total study time spent in the eight categories in the debugging task for all Experiments.

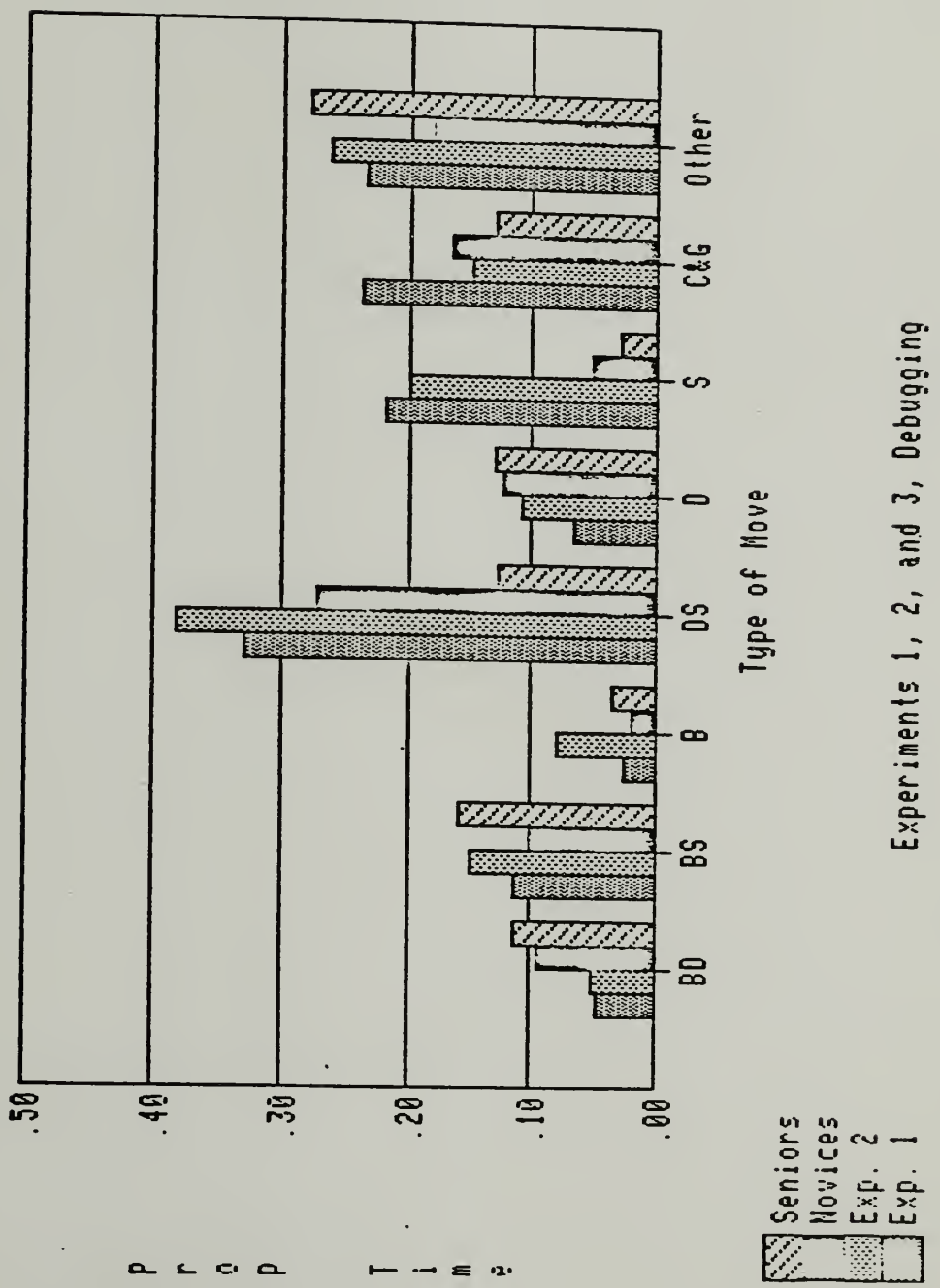
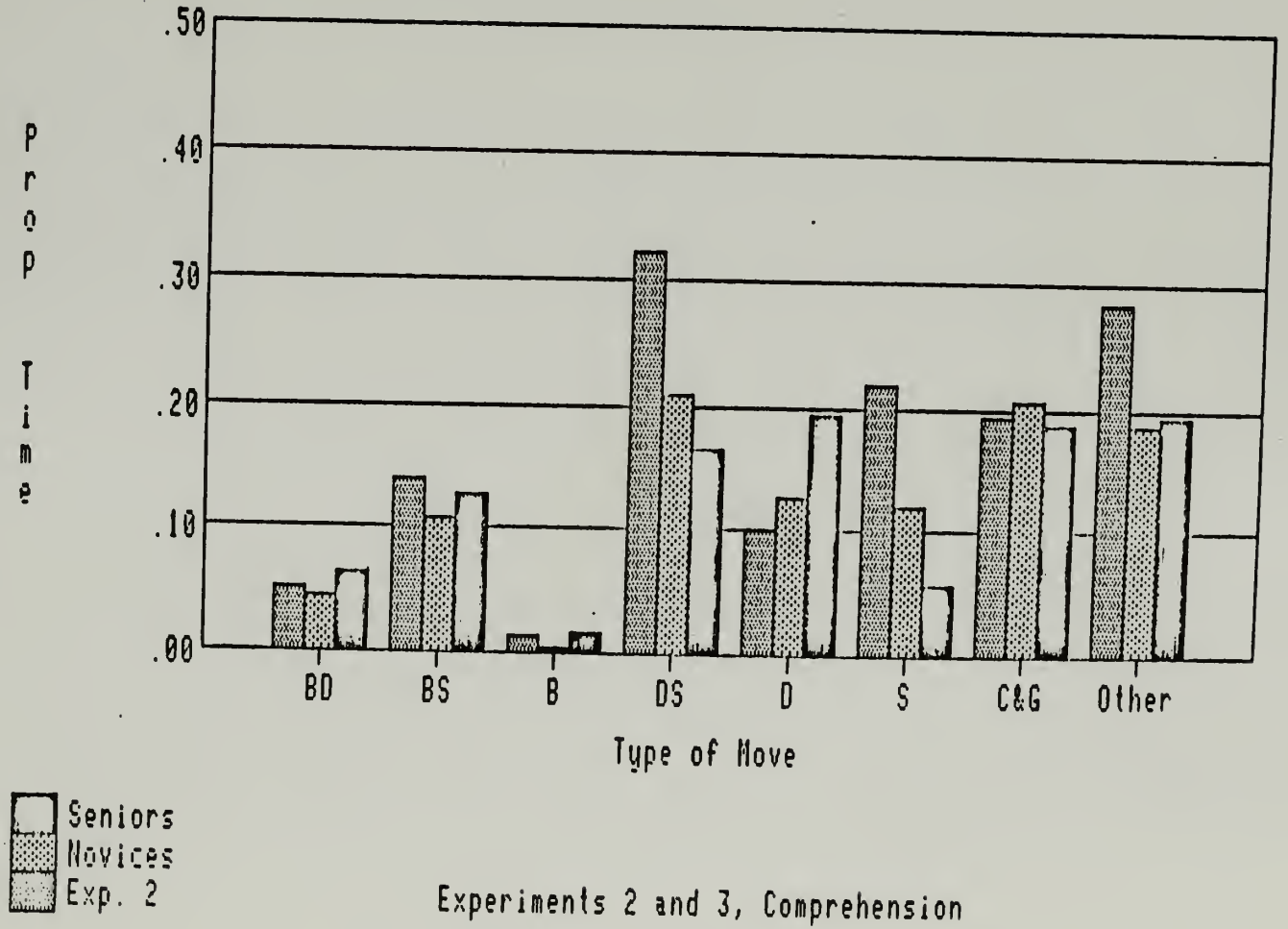

Prop Time
.50
.40
.30
.20
.10
.00
BD
BS
B
DS
D
S
C&G
Other
Type of Move
Seniors
Novices
Exp. 2
Exp. 1
Experiments 1, 2, and 3, Debugging

Figure 15

Proportion of total study time spent in the eight categories in the comprehension task for Experiments I and II.



Experiments 2 and 3, Comprehension

APPENDIX B

```
TURBO PASCAL Program Lister, Copyright 1983 Borland International   Page    1
Listing of: B:INVCLEAN.PAS
```

```
 1  (* -
 2   PROGRAM INVENTORY(INPUT,STOCKIN,ORDERIN,OUTPUT);

 4    CONST
 5        NUMPARTS = 4;
 6        NAMELENGTH = 9;

 8    TYPE
 9        PARTREC = RECORD
10                    PARTNAME : PACKED ARRAY[1..NAMELENGTH] OF CHAR;
11                    PARTNUM  : INTEGER;
12                    TOTAL    : INTEGER;
13                    ORDERAMT : INTEGER;
14                    WEIGHT   : REAL;
15                    PRICE    : REAL;
16                    OK       : BOOLEAN
17                  END;
18        PARTS = ARRAY[1..NUMPARTS] OF PARTREC;

20    VAR
21        ORDERNUMBER                 : INTEGER;
22        STOCK                       : PARTS;
23        STOCKIN, ORDERIN            : TEXT;
24        QUIT                        : BOOLEAN;
25  (* ------------------------------------------------------------------------ *)
26     PROCEDURE GETORDER(VAR STOCK : PARTS; VAR ORDERIN : TEXT);

28      VAR
29         INDEX       : INTEGER;

31        BEGIN
32           RESET(ORDERIN);
33           FOR INDEX := 1 TO NUMPARTS DO
34              WITH STOCK[INDEX] DO
35                 READLN(ORDERIN,ORDERAMT)
36        END; (* GETORDER *)
37  (* ------------------------------------------------------------------------ *)
38     PROCEDURE GETSTOCK(VAR STOCK : PARTS; VAR STOCKIN : TEXT);

40      VAR
41         INDEX, POINTER                 : INTEGER;

43        BEGIN
44           RESET(STOCKIN);
45           FOR INDEX := 1 TO NUMPARTS DO
46              WITH STOCK[INDEX] DO
47                 BEGIN
48                    FOR POINTER := 1 TO NAMELENGTH DO
49                       READ(STOCKIN,PARTNAME[POINTER]);
50                    READ(STOCKIN,PARTNUM);
```

```
 51                      READ(STOCKIN,TOTAL);
 52                      READ(STOCKIN,PRICE);
 53                      READLN(STOCKIN,WEIGHT)
 54                    END (* WITH...DO *)
 55          END; (* GETSTOCK *)
 56    (* ------------------------------------------------------------------------ *)
 57      PROCEDURE READORDERANDSTOCK(VAR ORDERNUMBER : INTEGER;VAR QUIT : BOOLEAN);

 59          BEGIN
 60             READLN;
 61             WRITELN('PLEASE ENTER THE ORDER NUMBER,');
 62             WRITELN('9999 TO TERMINATE THE PROGRAM.');
 63             QUIT := FALSE;
 64             READLN(ORDERNUMBER);
 65             IF ORDERNUMBER = 9999
 66             THEN QUIT := TRUE
 67             ELSE BEGIN
 68                     GETORDER(STOCK,ORDERIN);
 69                     GETSTOCK(STOCK,STOCKIN)
 70                  END (* ELSE *)
 71          END; (* READORDERANDSTOCK *)
 72    (* ------------------------------------------------------------------------ *)
 73      PROCEDURE TELLSHIPPING(COMPLETEORDER : PARTS; ORDERNUM : INTEGER);

 75       VAR
 76          INDEX        : INTEGER;
 77          POUNDS       : REAL;

 79          BEGIN
 80             POUNDS := 0;
 81             FOR INDEX := 1 TO NUMPARTS DO
 82                WITH COMPLETEORDER[INDEX] DO
 83                   POUNDS := POUNDS + (ORDERAMT * WEIGHT);
 84             WRITELN;
 85             WRITELN('ORDER ',ORDERNUM:4,' IS FILLED AND READY TO BE SHIPPED');
 86             WRITELN('WITH A TOTAL WEIGHT OF ',POUNDS:7:2,' LBS.');
 87             WRITELN
 88          END; (* TELLSHIPPING *)
 89    (* ------------------------------------------------------------------------ *)
 90      PROCEDURE TELLPRODUCTION(PARTIALSTOCK : PARTS; ORDERNUM : INTEGER);

 92       VAR
 93          INDEX, UNITS      : INTEGER;

 95          BEGIN
 96             WRITELN('ORDER ',ORDERNUM:4,' IS NOT FILLED AND NEEDS THE');
 97             WRITELN('FOLLOWING PARTS:');
 98             WRITELN;
 99             WRITELN('PART NUMBER  NUMBER OF UNITS');
100             WRITELN('-----------  ---------------');
101             FOR INDEX := 1 TO NUMPARTS DO
```

```
102                 WITH PARTIALSTOCK[INDEX] DO
103                     IF NOT OK
104                     THEN BEGIN
105                             UNITS := ORDERAMT - TOTAL;
106                             WRITELN(PARTNUM:4,'          ',UNITS:4)
107                          END
108         END; (* TELLPRODUCTION *)
109     (* ------------------------------------------------------------------------ *)
110         PROCEDURE CHECKORDER(VAR STOCK : PARTS);

112         VAR
113             INDEX               : INTEGER;
114             ORDERFILLED         : BOOLEAN;
115             COST                : REAL;

117           BEGIN
118             ORDERFILLED := TRUE;
119             FOR INDEX := 1 TO NUMPARTS DO
120                 WITH STOCK[INDEX] DO
121                       IF TOTAL >= ORDERAMT
122                       THEN OK := TRUE
123                       ELSE BEGIN
124                               OK := FALSE;
125                               ORDERFILLED := FALSE
126                            END;
127             IF ORDERFILLED
128             THEN TELLSHIPPING(STOCK,ORDERNUMBER)
129             ELSE TELLPRODUCTION(STOCK,ORDERNUMBER);
130             COST := 0;
131             FOR INDEX := 1 TO NUMPARTS DO
132                 WITH STOCK[INDEX] DO
133                     COST := COST + ORDERAMT * PRICE;
134             WRITELN('THE TOTAL COST OF THIS ORDER IS $',COST:3:2,'.')
135         END; (* CHECKORDER *)
136     (* ------------------------------------------------------------------------ *)
137         BEGIN (* MAIN *)
138           READORDERANDSTOCK(ORDERNUMBER,QUIT);
139           IF NOT QUIT
140           THEN CHECKORDER(STOCK)
141         END.
```

TURBO PASCAL Program Lister, Copyright 1983 Borland International Page 1
Listing of: B:PAYNOBUG.PAS

```
 1    PROGRAM PAYROLL(INPUT,WORKERINFIL,OUTPUT);

 3     CONST
 4          NUMWORKERS = 4;
 5          IDLENGTH = 7;

 7     TYPE
 8          IDTYPE = PACKED ARRAY[1..IDLENGTH] OF CHAR;
 9          WORKER = RECORD
10                     IDNO          : IDTYPE;
11                     PCSPERIOD1    : INTEGER;
12                     PCSPERIOD2    : INTEGER;
13                     HOURS         : INTEGER;
14                     PAY           : REAL
15                   END; (* WORKER *)
16          WORKERS = ARRAY[1..NUMWORKERS] OF WORKER;

18    VAR
19          WORKFORCE                : WORKERS;
20          WORKERINFIL              : TEXT;
21          QUIT                     : BOOLEAN;

23  (* ------------------------------------------------------------------------ *)

25    PROCEDURE READCOUNTS(VAR WORKFORCE : WORKERS; VAR WORKERINFIL : TEXT);

27     VAR
28        INDEX, POINTER   : INTEGER;

30       BEGIN
31          RESET(WORKERINFIL);
32          FOR INDEX := 1 TO NUMWORKERS DO
33             WITH WORKFORCE[INDEX] DO
34                BEGIN
35                   FOR POINTER := 1 TO IDLENGTH DO
36                      READ(WORKERINFIL,IDNO[POINTER]);
37                   READ(WORKERINFIL,PCSPERIOD1);
38                   READ(WORKERINFIL,PCSPERIOD2);
39                   READLN(WORKERINFIL)
40                END (* WITH... DO *)
41       END; (* READCOUNTS *)

43  (* ------------------------------------------------------------------------ *)

45    PROCEDURE STOREHOURS(VAR WORKFORCE : WORKERS; IDNUM   IDTYPE,
46                         NUMHOURS : INTEGER);

48     VAR
49        INDEX             : INTEGER;
50        FOUND             : BOOLEAN;
```

```
52          BEGIN
53             FOUND := FALSE;
54             INDEX := 1;
55             WHILE (NOT FOUND) AND (INDEX <= NUMWORKERS) DO
56                BEGIN
57                   IF WORKFORCE[INDEX].IDNO = IDNUM
58                   THEN BEGIN
59                           FOUND := TRUE;
60                           WORKFORCE[INDEX].HOURS := NUMHOURS
61                        END;
62                   INDEX := INDEX + 1
63                END; (* WHILE...DO *)
64             IF NOT FOUND
65             THEN WRITELN(IDNUM,' NOT FOUND.  PLEASE CHECK AND/OR RE-ENTER.')
66       END; (* STOREHOURS *)

68    (* ------------------------------------------------------------------------ *)

70       PROCEDURE READANDSTORE(VAR WORKFORCE : WORKERS; VAR QUIT : BOOLEAN);

72        VAR
73           NOOFHOURS, POINTER     : INTEGER;
74           IDNUMBER               : IDTYPE;

76         BEGIN
77            WRITELN('READING COUNTS FROM DISK...');
78            READCOUNTS(WORKFORCE,WORKERINFIL);
79            WRITELN('ENTER A WORKER ID. FOLLOWED BY HOURS WORKED,');
80            WRITELN('ON THE SAME LINE, SEPARATED BY COMMAS.');
81            WRITELN('ENTER "9999999" AS THE ID. TO STOP ENTRY.');
82            FOR POINTER := 1 TO IDLENGTH DO
83               READ(IDNUMBER[POINTER]);
84            READ(NOOFHOURS);
85            QUIT := TRUE;
86            WHILE IDNUMBER <> '9999999' DO
87               BEGIN
88                  QUIT := FALSE;
89                  STOREHOURS(WORKFORCE,IDNUMBER,NOOFHOURS);
90                  FOR POINTER := 1 TO IDLENGTH DO
91                     READ(IDNUMBER[POINTER]);
92                  READ(NOOFHOURS)
93               END (* WHILE...DO *)
94        END; (* READANDSTORE *)

96    (* ------------------------------------------------------------------------ *)

98       FUNCTION BONUSPCS(COUNT1, COUNT2 : INTEGER) : INTEGER;

100       CONST
101            QUOTA = 300;
```

TURBO PASCAL Program Lister, Copyright 1983 Borland International Page 3
Listing of: B:PAYNOBUG.PAS

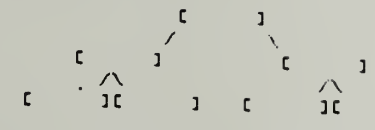
```
103          VAR
104             AVERAGE, PIECES     : INTEGER;

106           BEGIN
107              AVERAGE := (COUNT1 + COUNT2) DIV 2;
108              PIECES := AVERAGE - QUOTA;
109              IF PIECES > 0
110              THEN BONUSPCS := PIECES
111              ELSE BONUSPCS := 0
112           END; (* BONUSPCS *)

114   (* ------------------------------------------------------------------------ *)

116      FUNCTION BONUS(NUMBONUSPCS : INTEGER) : REAL;

118       CONST
119            BONUSCUTOFF = 30;
120            BONUSAMT = 1.00;

122         VAR
123            DOUBLEBONUS     : REAL;


126         BEGIN
127            IF NUMBONUSPCS <= BONUSCUTOFF
128            THEN BONUS := NUMBONUSPCS * BONUSAMT
129            ELSE BEGIN
130                    DOUBLEBONUS := (NUMBONUSPCS - BONUSCUTOFF) * (2 * BONUSAMT),
131                    BONUS := (BONUSCUTOFF * BONUSAMT) + DOUBLEBONUS
132                 END
133         END; (* BONUS *)

135   (* ------------------------------------------------------------------------ *)

137      PROCEDURE CALCULATEPAY(VAR WORKFORCE : WORKERS);

139       CONST
140            PAYRATE = 5.00;

142       VAR
143          INDEX, BPIECES          INTEGER;
144          REGPAY, BONUSPAY        REAL;


147         BEGIN
148            WRITELN('EMPLOYEE NO.       PAY');
149            WRITELN('------------    -------');
150            FOR INDEX := 1 TO NUMWORKERS DO
151               WITH WORKFORCE[INDEX] DO
152                  BEGIN
```

```
153                     WRITE(IDNO,'         ');
154                     REGPAY := HOURS * PAYRATE;
155                     BPIECES := BONUSPCS(PCSPERIOD1,PCSPERIOD2);
156                     BONUSPAY := BONUS(BPIECES);
157                     PAY := REGPAY + BONUSPAY;
158                     WRITELN(PAY:6:2)
159                   END (* WITH...DO *)
160         END;

163   (* ------------------------------------------------------------------------------ *)

165     BEGIN (* MAIN *)
166       READANDSTORE(WORKFORCE,QUIT);
167       IF NOT QUIT
168       THEN CALCULATEPAY(WORKFORCE)
169     END.
```

APPENDIX C

FILE: C MOVE1102.DAT

[74.0] COMMENTS [ ] GLOBAL

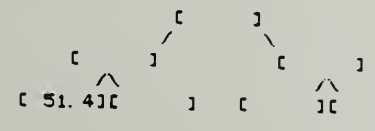


SEQUENTIAL POSITION = 0

[ ] COMMENTS [45.1] GLOBAL

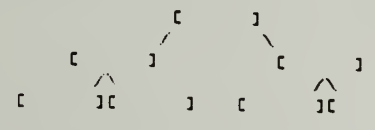
SEQUENTIAL POSITION = 1

[ ] COMMENTS [ ] GLOBAL



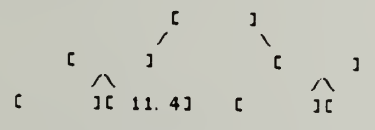
SEQUENTIAL POSITION = 2

[38.0] COMMENTS [ ] GLOBAL



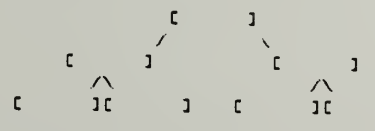
SEQUENTIAL POSITION = 0

[ ] COMMENTS [ ] GLOBAL

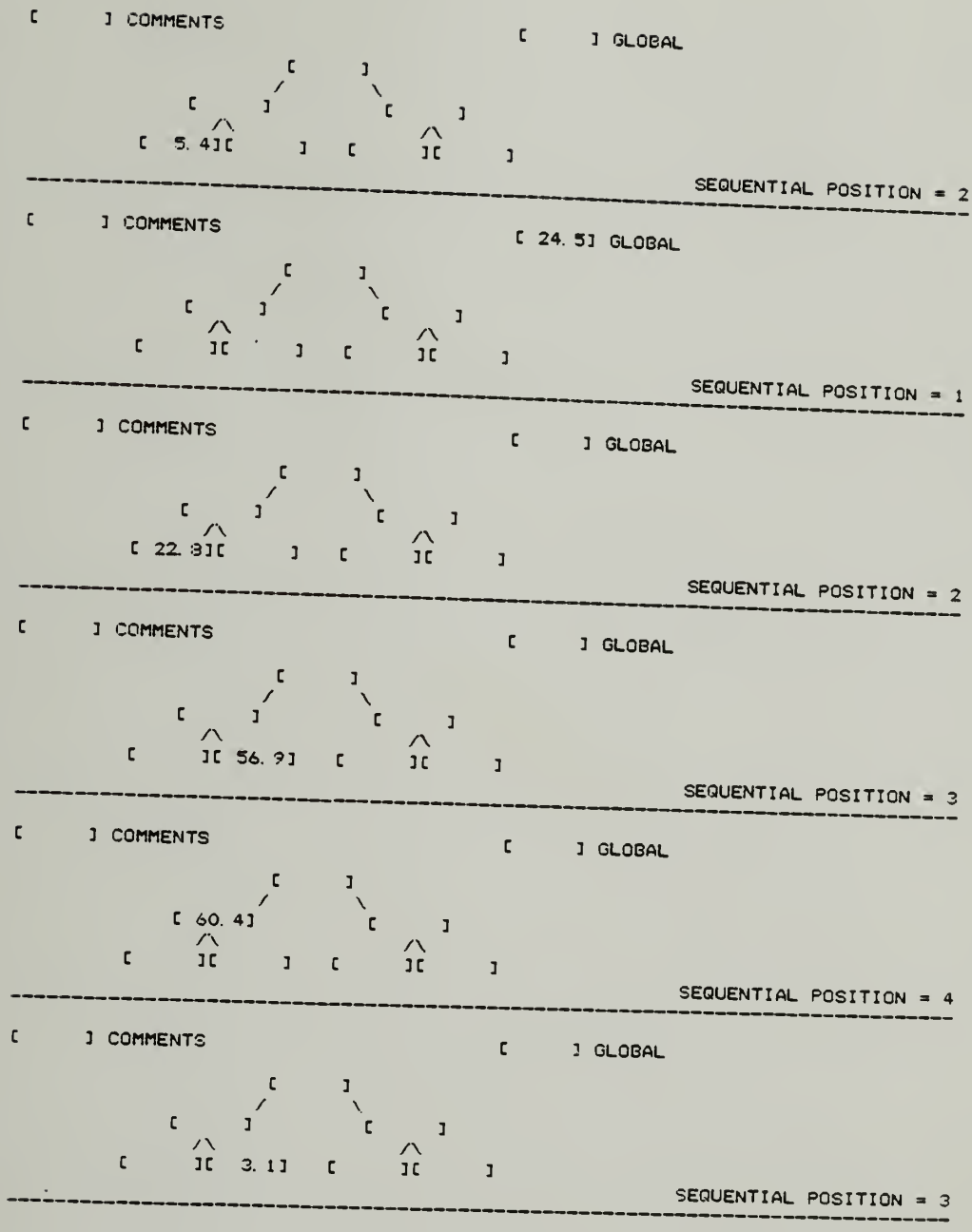


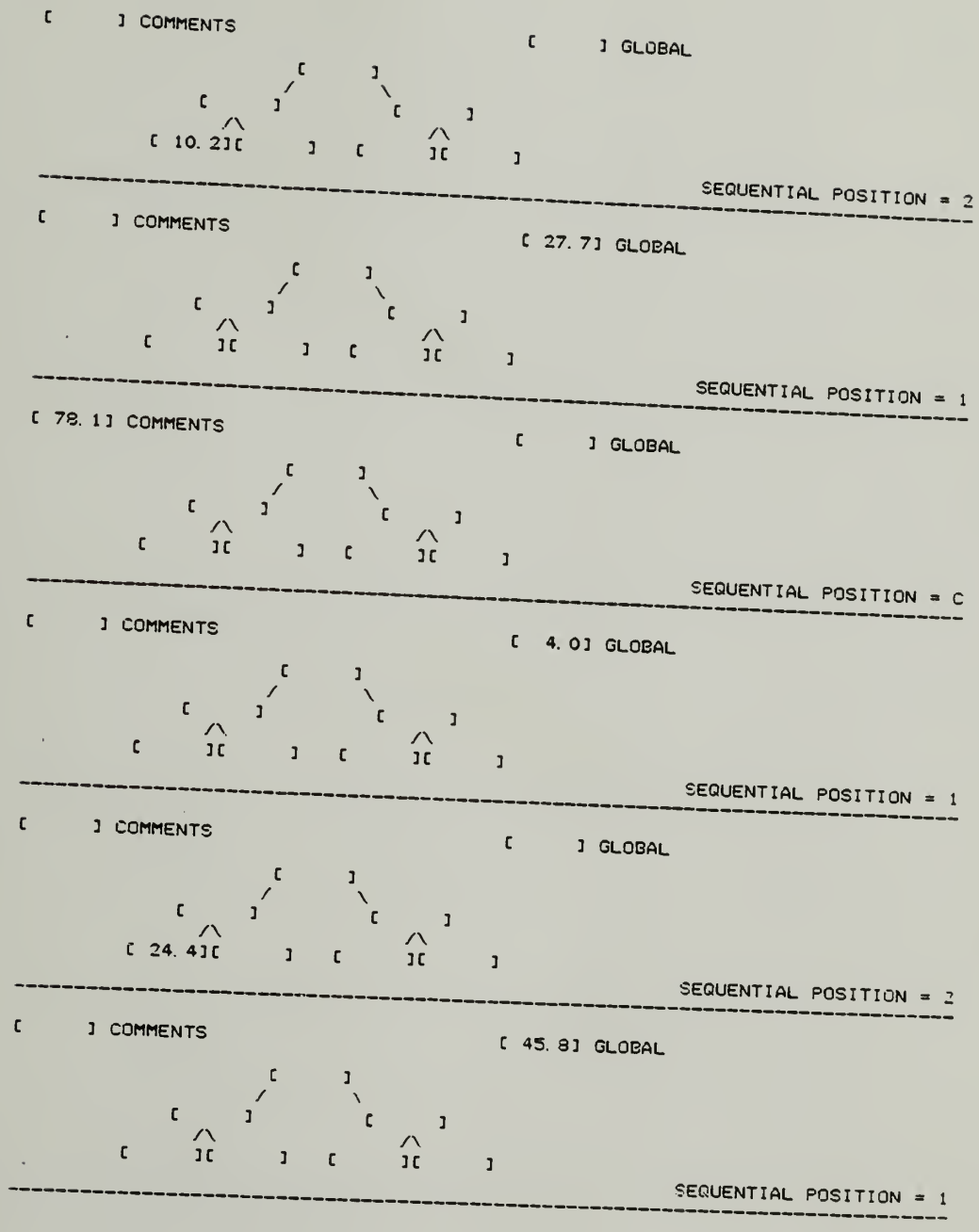
SEQUENTIAL POSITION = 3

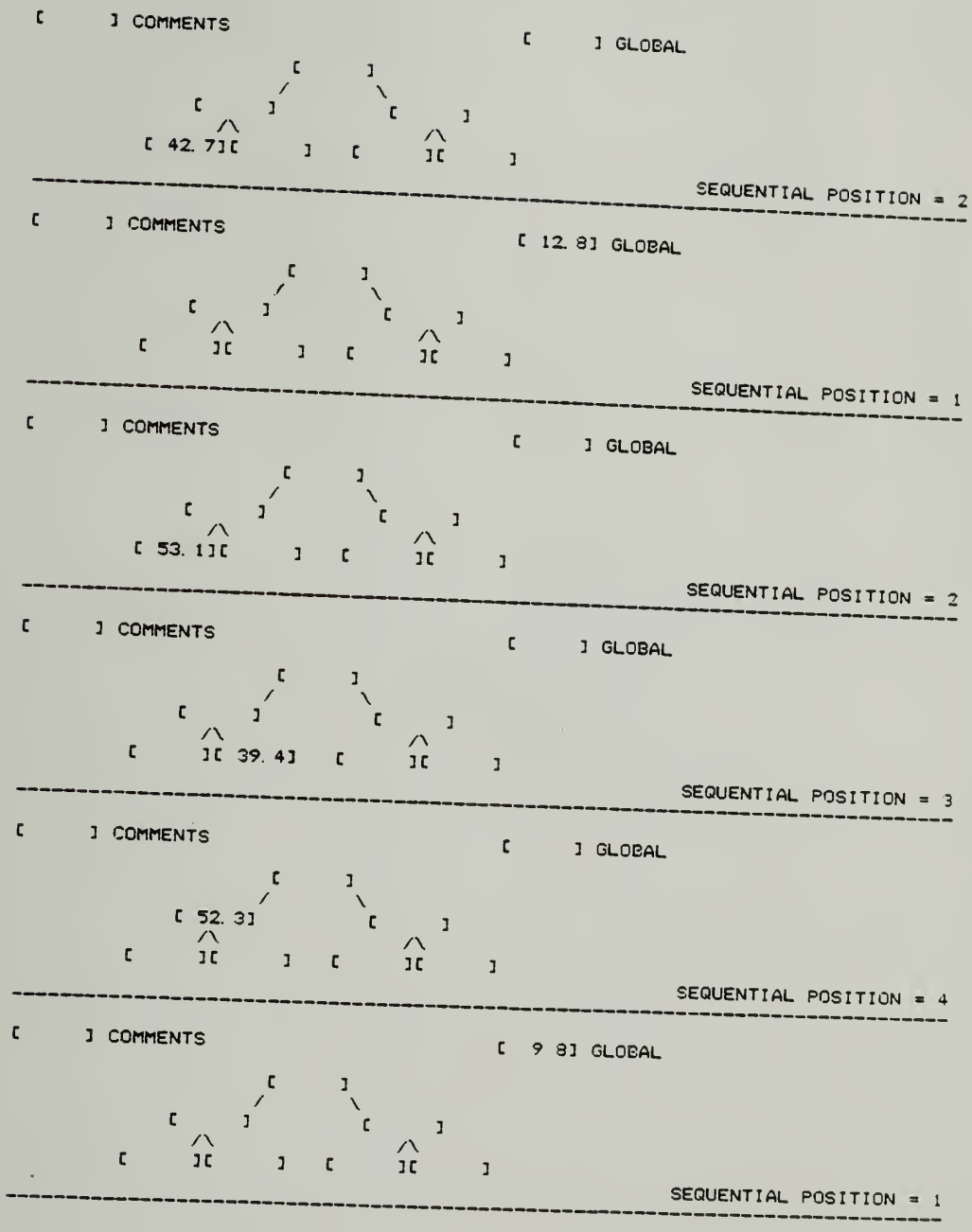
[ ] COMMENTS [26.2] GLOBAL

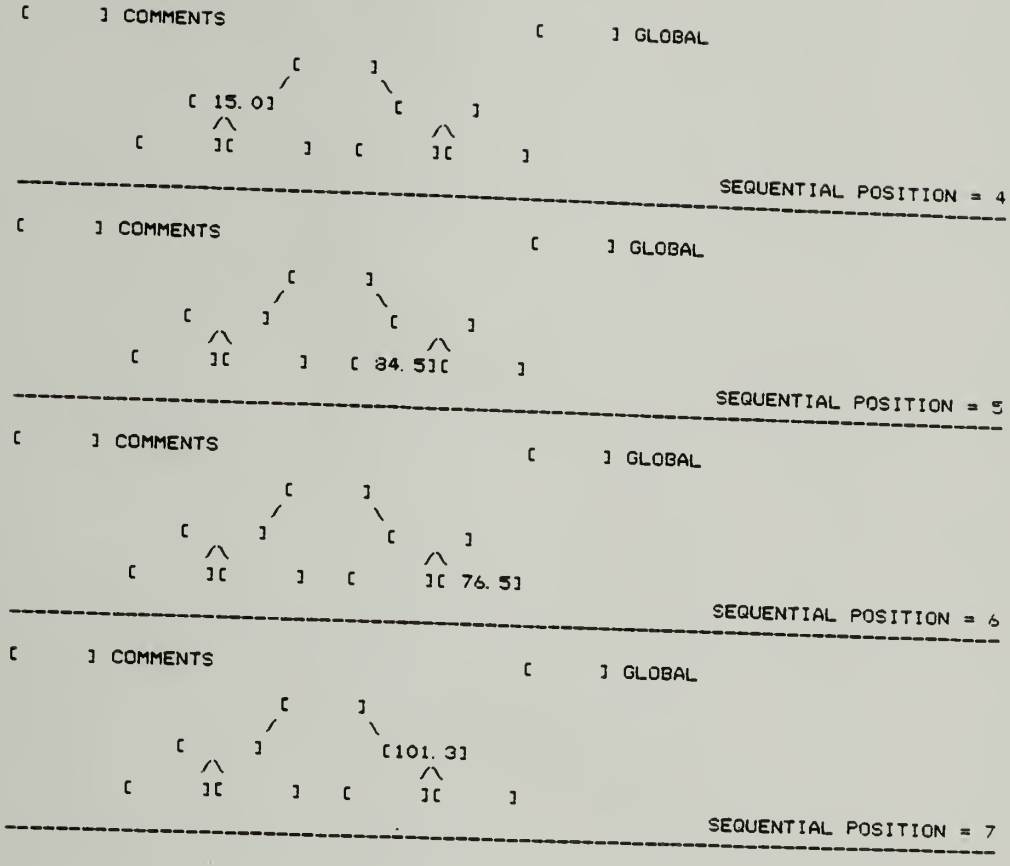


SEQUENTIAL POSITION = 1


[ ] COMMENTS
[ ] GLOBAL
[5. 4][ ] [ ][ ]
SEQUENTIAL POSITION = 2
[ ] COMMENTS
[24. 5] GLOBAL
[ ][ ] [ ][ ]
SEQUENTIAL POSITION = 1
[ ] COMMENTS
[ ] GLOBAL
[22. 8][ ] [ ][ ]
SEQUENTIAL POSITION = 2
[ ] COMMENTS
[ ] GLOBAL
[ ][56. 9] [ ][ ]
SEQUENTIAL POSITION = 3
[ ] COMMENTS
[ ] GLOBAL
[60. 4] [ ][ ]
SEQUENTIAL POSITION = 4
[ ] COMMENTS
[ ] GLOBAL
[ ][3. 1] [ ][ ]
SEQUENTIAL POSITION = 3


[ ] COMMENTS
[ ] GLOBAL
[10. 2][ ] [ ][ ]
SEQUENTIAL POSITION = 2
[ ] COMMENTS
[27. 7] GLOBAL
[ ][ ] [ ][ ]
SEQUENTIAL POSITION = 1
[78. 1] COMMENTS
[ ] GLOBAL
[ ][ ] [ ][ ]
SEQUENTIAL POSITION = C
[ ] COMMENTS
[4. 0] GLOBAL
[ ][ ] [ ][ ]
SEQUENTIAL POSITION = 1
[ ] COMMENTS
[ ] GLOBAL
[24. 4][ ] [ ][ ]
SEQUENTIAL POSITION = 2
[ ] COMMENTS
[45. 8] GLOBAL
[ ][ ] [ ][ ]
SEQUENTIAL POSITION = 1


[ ] COMMENTS
[ ] GLOBAL
[42. 7][ ] [ ][ ]
SEQUENTIAL POSITION = 2
[ ] COMMENTS
[12. 8] GLOBAL
[ ][ ] [ ][ ]
SEQUENTIAL POSITION = 1
[ ] COMMENTS
[ ] GLOBAL
[53. 1][ ] [ ][ ]
SEQUENTIAL POSITION = 2
[ ] COMMENTS
[ ] GLOBAL
[ ][39. 4] [ ][ ]
SEQUENTIAL POSITION = 3
[ ] COMMENTS
[ ] GLOBAL
[52. 3] [ ]
[ ][ ] [ ][ ]
SEQUENTIAL POSITION = 4
[ ] COMMENTS
[9 8] GLOBAL
[ ][ ] [ ][ ]
SEQUENTIAL POSITION = 1


[ ] COMMENTS
[ ] GLOBAL
[15.0]
SEQUENTIAL POSITION = 4
[ ] COMMENTS
[ ] GLOBAL
[84.5]
SEQUENTIAL POSITION = 5
[ ] COMMENTS
[ ] GLOBAL
[76.5]
SEQUENTIAL POSITION = 6
[ ] COMMENTS
[ ] GLOBAL
[101.3]
SEQUENTIAL POSITION = 7

APPENDIX D

FILE: B:MOVE1102.DAT

	C	G1	3LL	3LR	2L.	3RL	3RR	2R.	1.
C		67/245.5 2/ 491 290.62		33/114.0 1/ 114 0.00					
G1	13/781.0 1/ 781 0.00		75/333.0 6/ 1998 188.46		13/150.0 1/ 150 0.00				
3LL	14/380.0 1/ 380 0.00	57/277.0 4/ 1108 136.61		29/481.5 2/ 963 123.74					
3LR		25/262.0 1/ 262 0.00	25/102.0 1/ 102 0.00		50/563.5 2/ 1127 57.27				
2L.		33/ 98.0 1/ 98 0.00		33/ 31.0 1/ 31 0.00		33/845.0 1/ 845 0.00			
3RL							100/ 765 1/ 765 0.00		
3RR								100/1013 1/ 1013 0.00	
2R.									
1.									
SUM=	1161	1959	2100	1108	1277	845	765	1013	01

TOTAL TIME IN MATRIX = 10228

APPENDIX E

Computer Program Comprehension

The purpose of this research is to learn more about the thought processes involved when a person attempts to understand someone else's computer program.

In this experiment you will be asked to comprehend a Pascal program of moderate length (about 150 lines). The program will be displayed on a CRT using a specialized scrolling program. You will be given 10-15 minutes to familiarize yourself with the scrolling program before the main part of the experiment actually begins. After this period, a description of the program which you are to comprehend will appear on the screen. Your task is to notify the experimenter when you feel that you have comprehended the program fully. You will not be able to take notes during the task.

When you have comprehended the program, you will be tested on your comprehension. It is important that you comprehend the program FULLY. You should aim to comprehend the workings of program as well as if you had written it yourself. We have found that people who don't develop a high degree of familiarity with the program exhibit poor performance in the testing phase of the experiment.

You are free to leave at any time if you should decide that you no longer wish to participate but if the experimenter has not informed you that you have completed the task, you will be paid only for the amount of time you have spent. However, if you finish the task in less than 1 1/2 hrs., you will be paid the full $10.00.

Feel free to ask questions about the task at any point. At the end of the experiment the experimenter will answer any questions about the purpose and methods of the experiment.

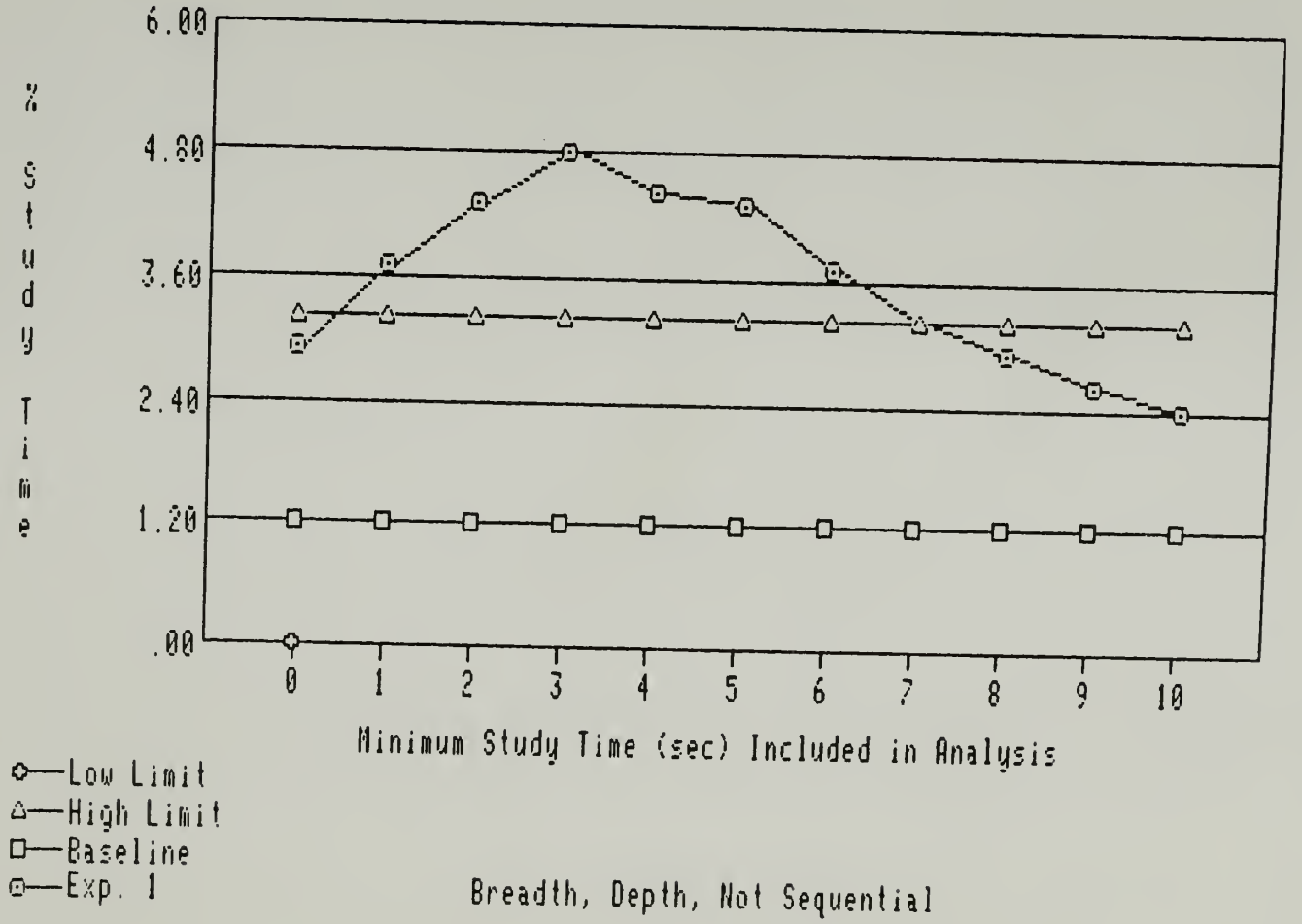
If you would like any additional information about the experiment and its results, you can contact:

Christopher Young
431 Tobin Hall
Department of Psychology
University of Massachusetts
Amherst, MA. 01003

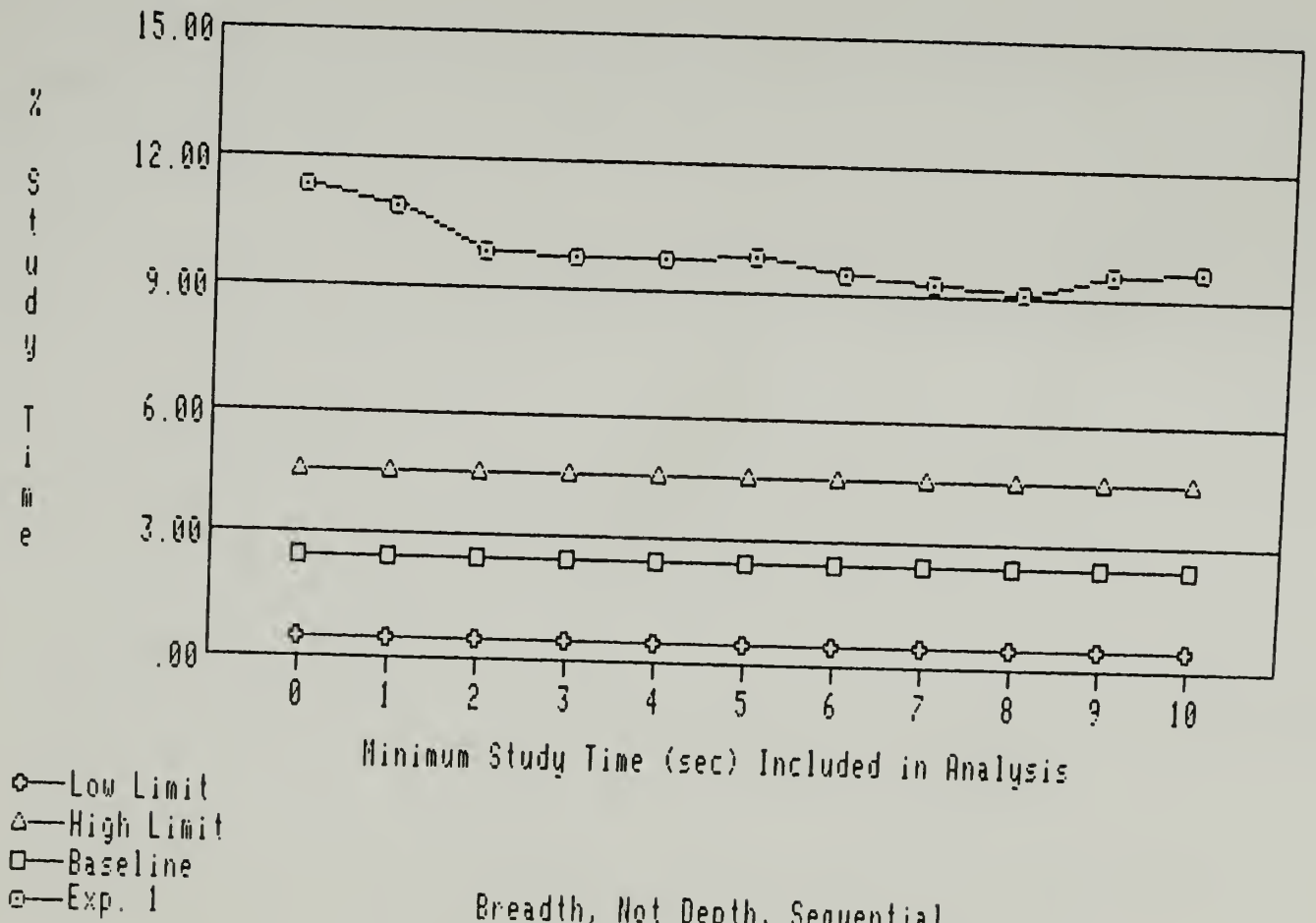
I have read the above instructions and understand the task and my rights.

Name..... Date.....

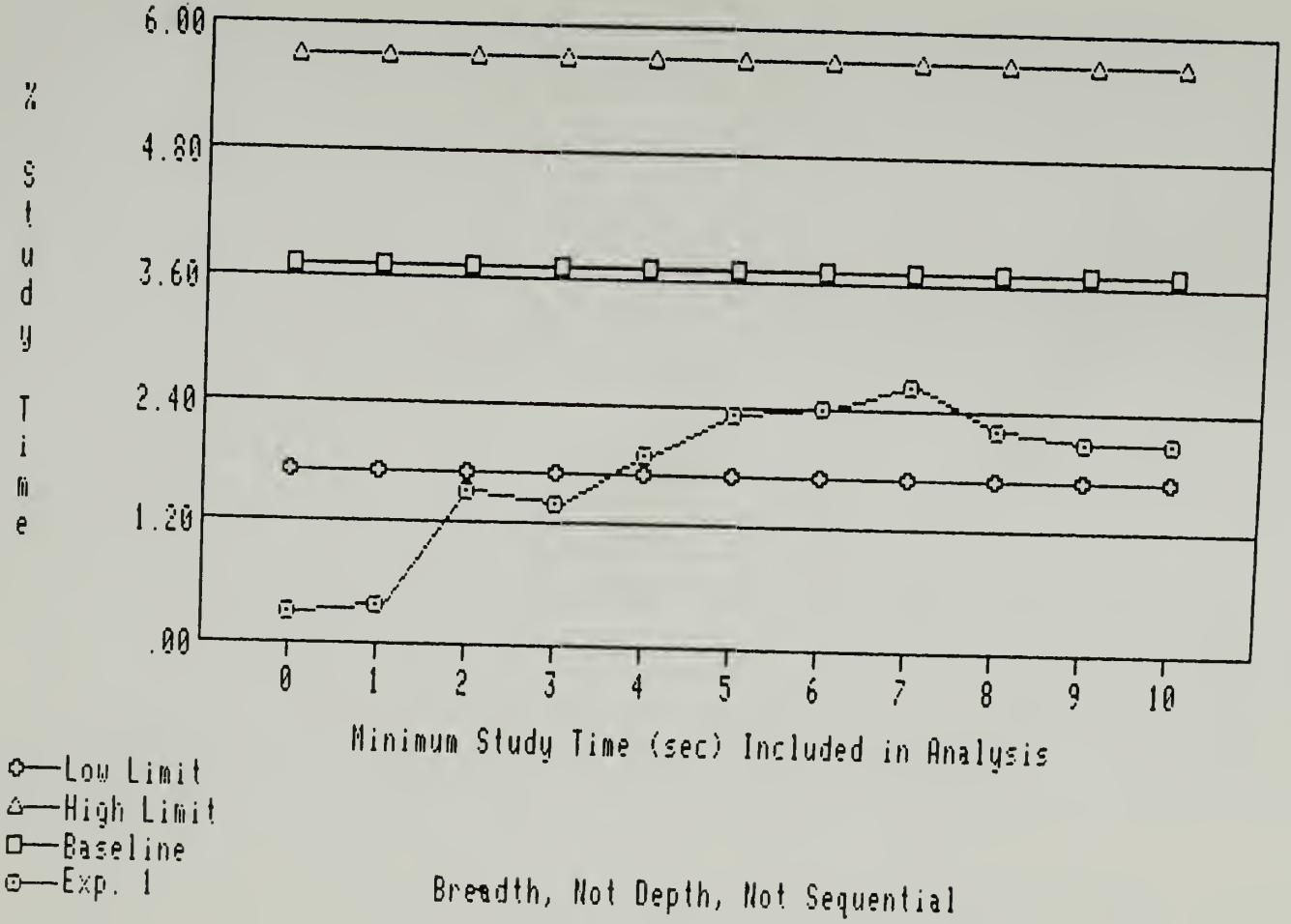
APPENDIX F



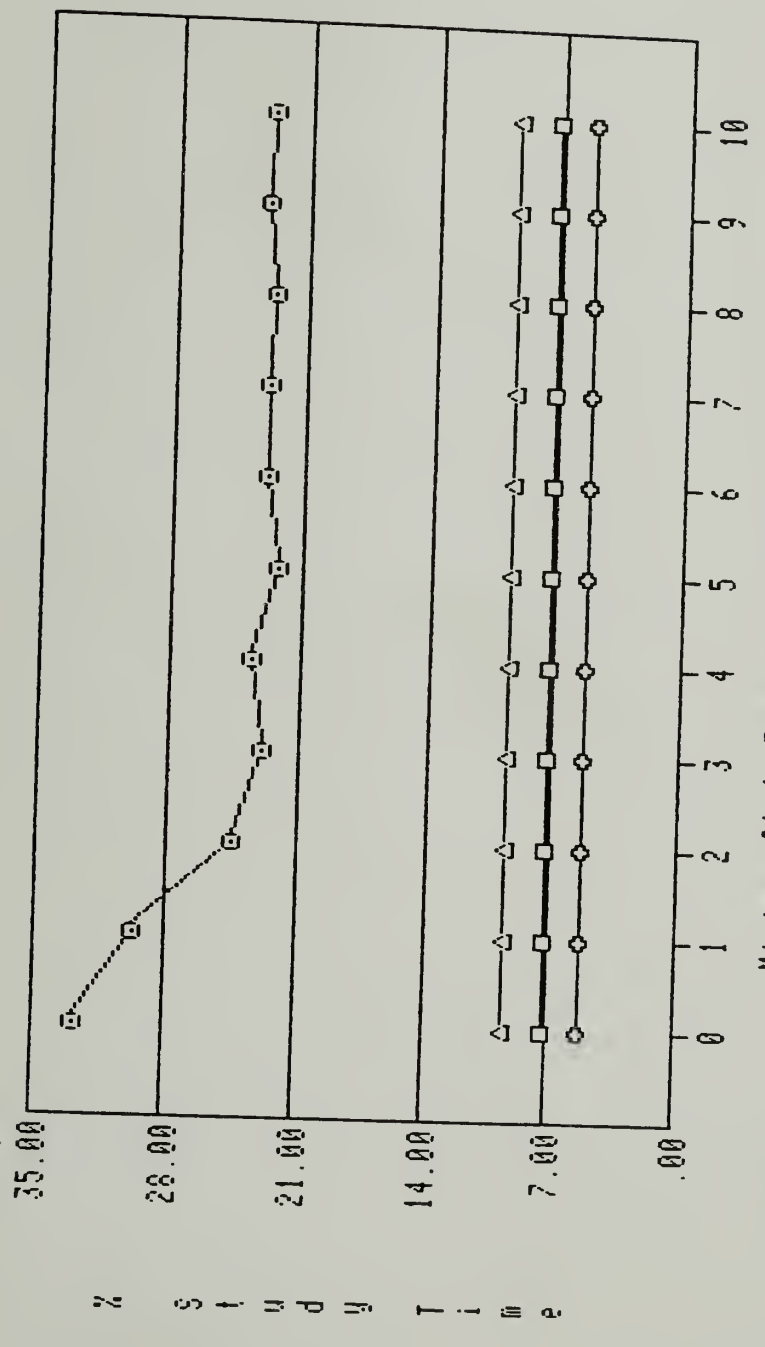
Breadth, Depth, Not Sequential

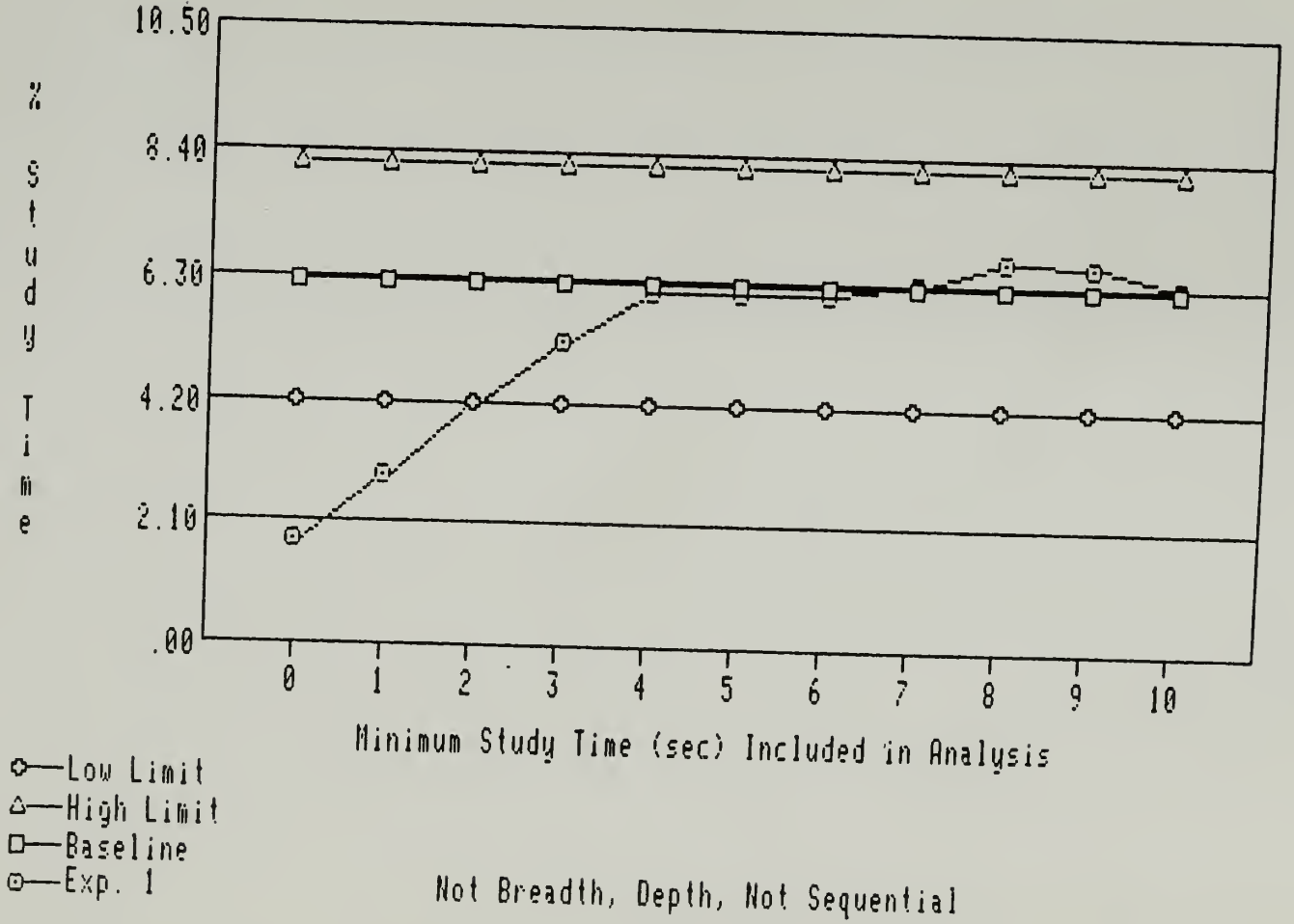


Breadth, Not Depth, Sequential

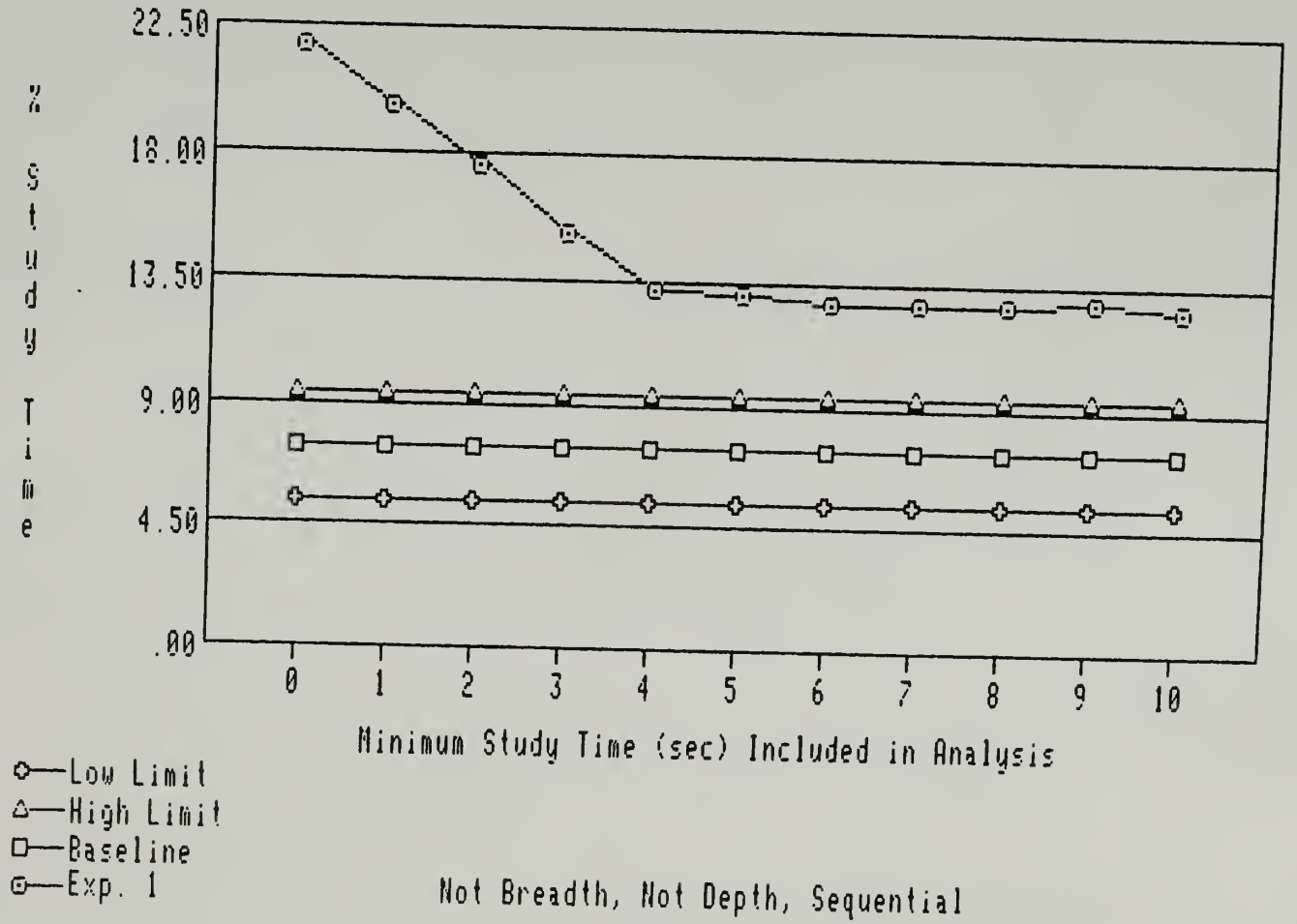


Breadth, Not Depth, Not Sequential

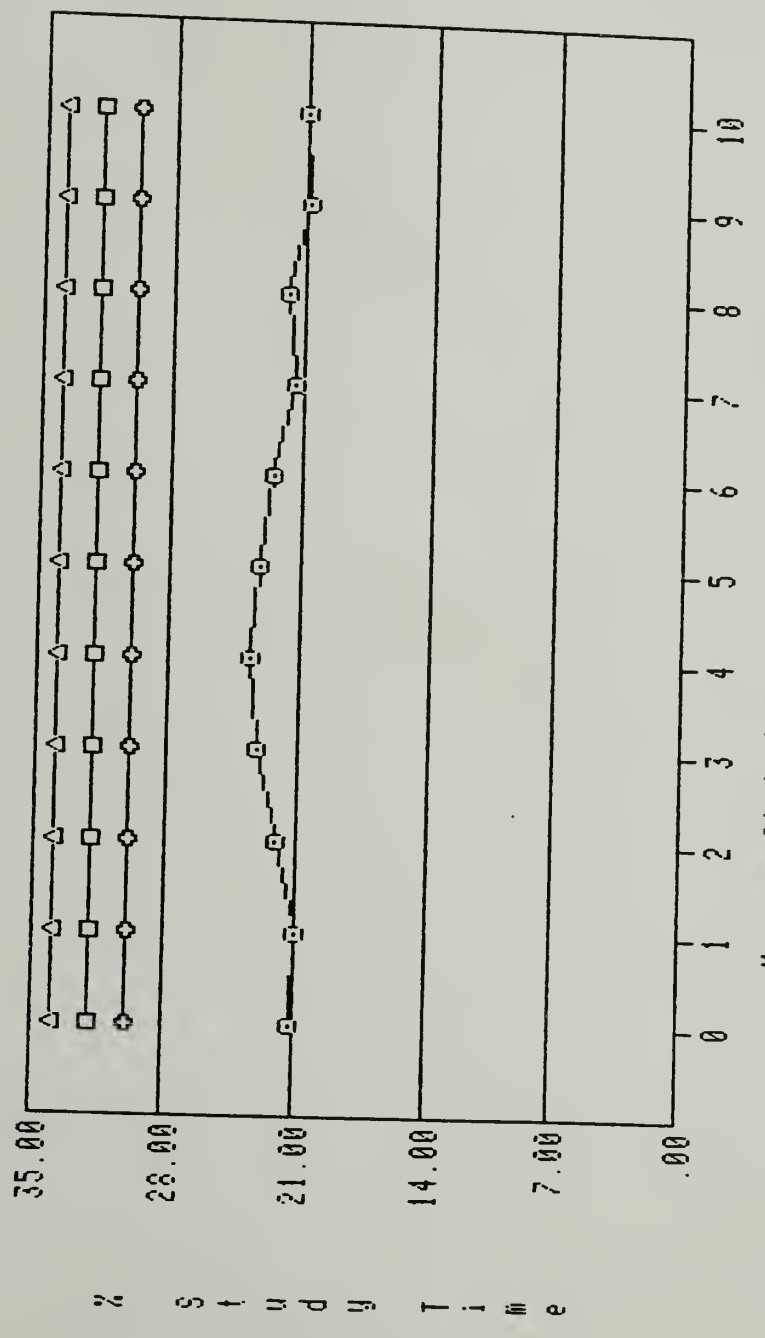

35.00
28.00
21.00
14.00
7.00
.00
% Study Time
0 1 2 3 4 5 6 7 8 9 10
Minimum Study Time (sec) Included in Analysis
Low Limit
High Limit
Baseline
Exp. 1
Not Breadth, Depth, Sequential

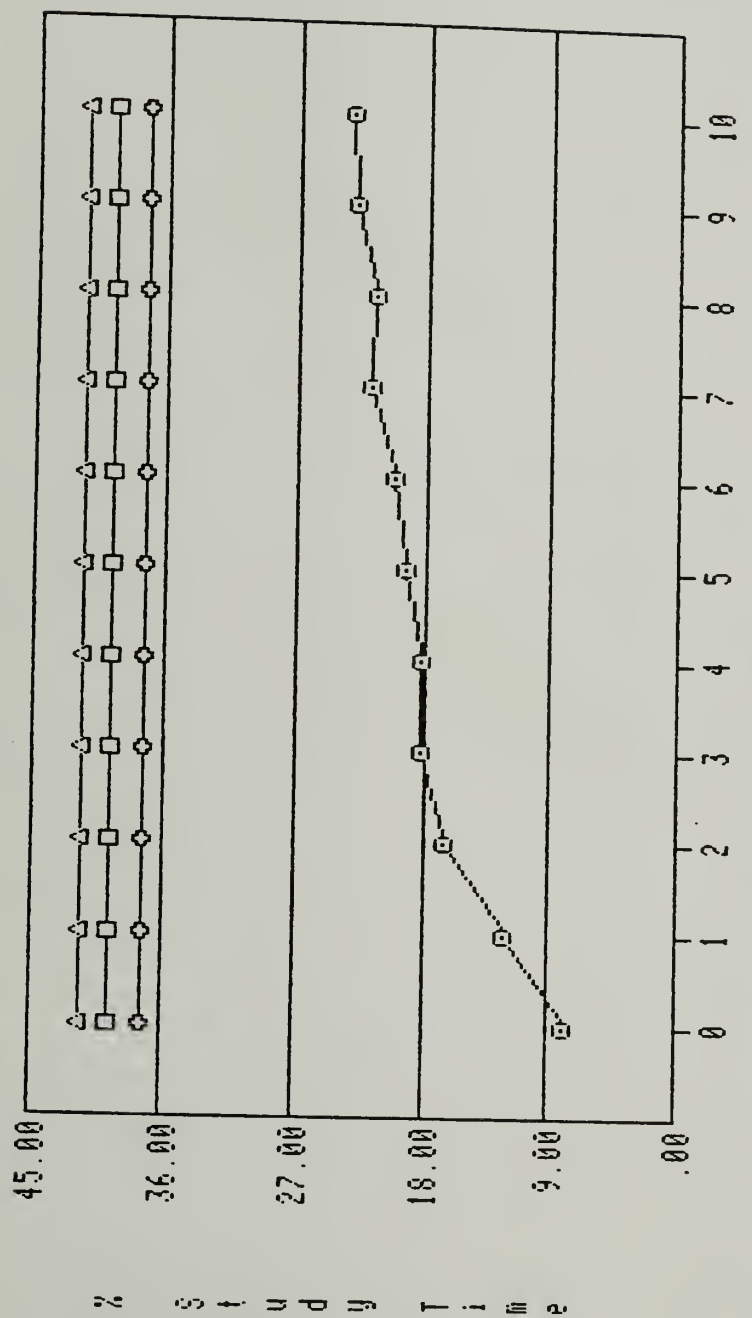


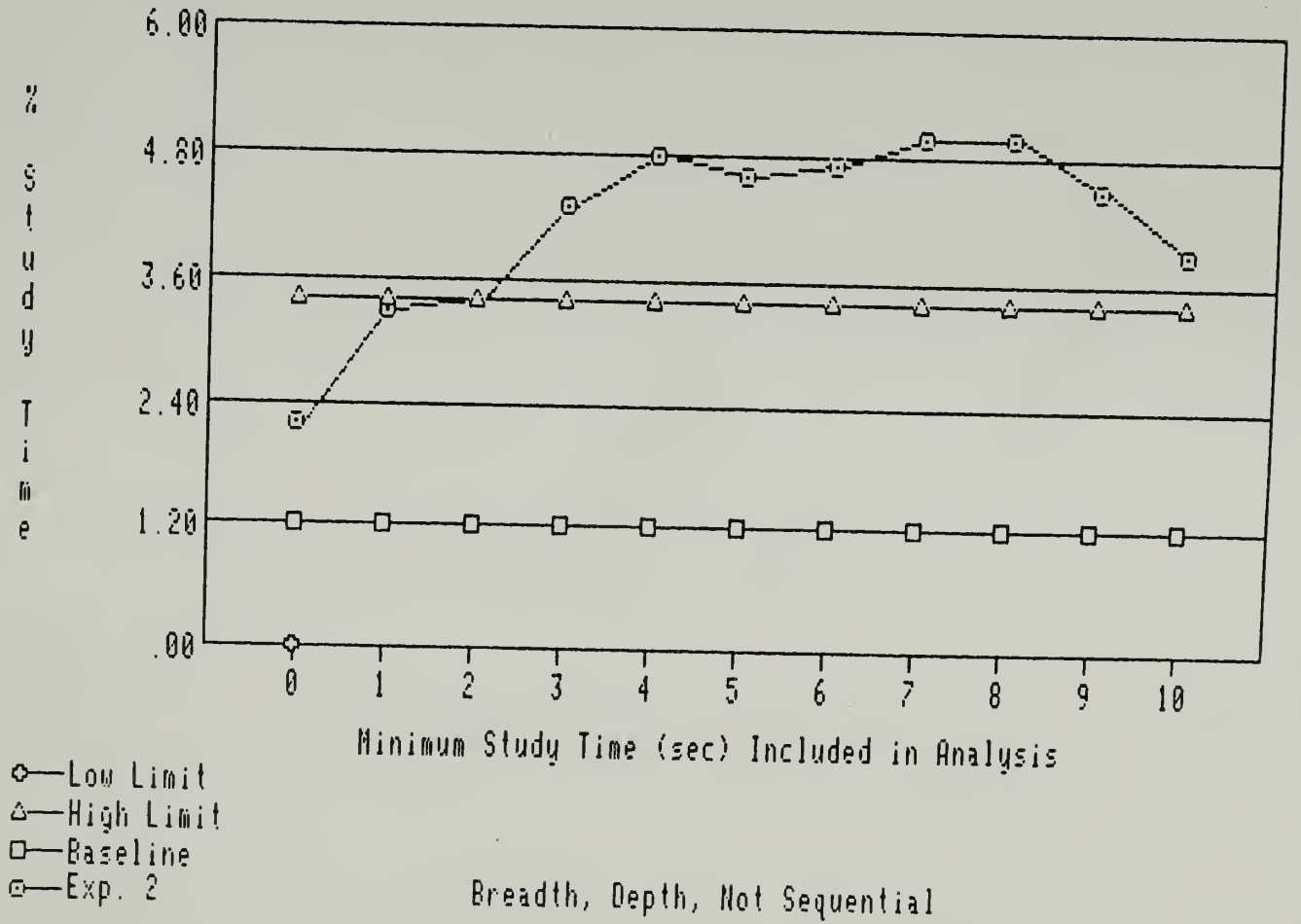
Not Breadth, Depth, Not Sequential



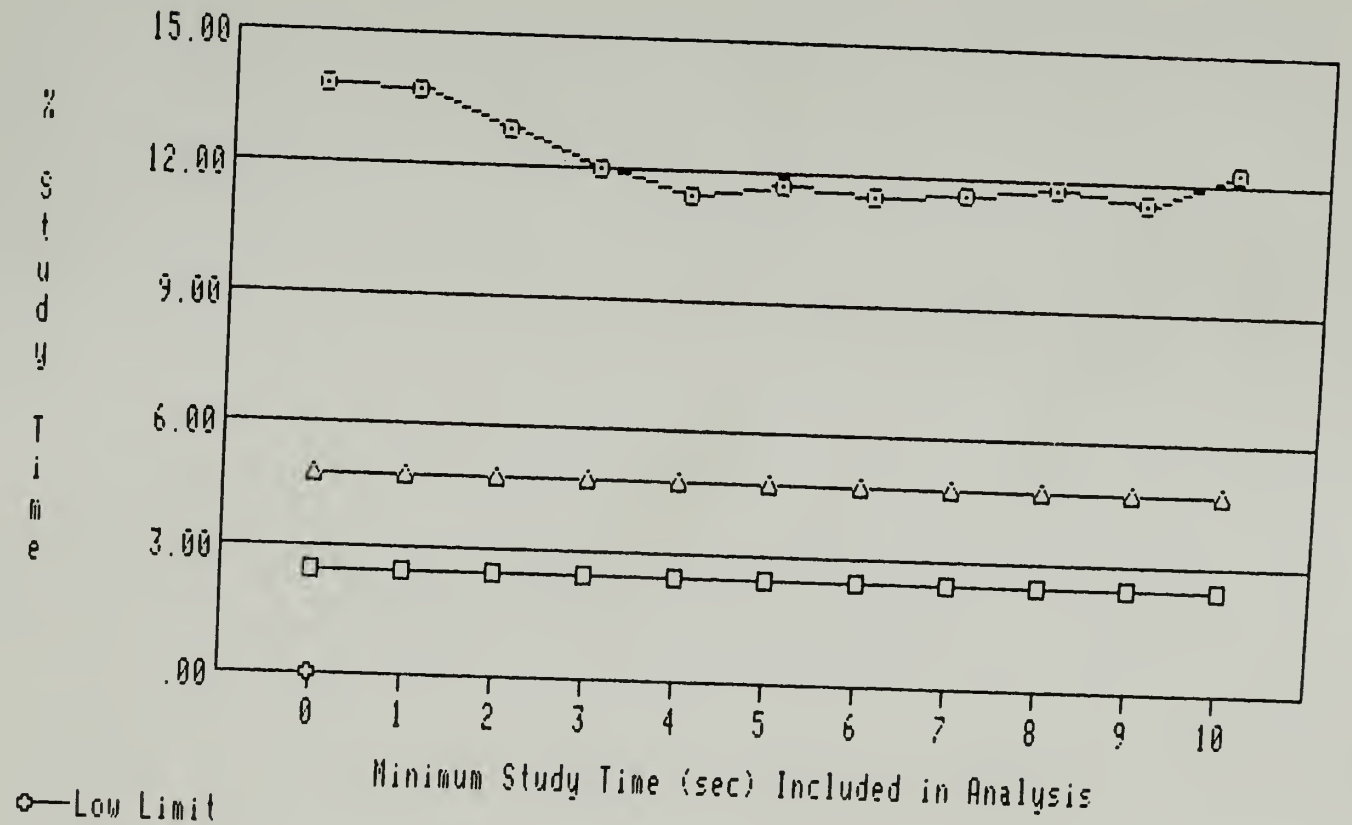
Not Breadth, Not Depth, Sequential


35.00
28.00
21.00
14.00
7.00
.00
% Study Time
0 1 2 3 4 5 6 7 8 9 10
Minimum Study Time (sec) Included in Analysis
Low Limit
High Limit
Baseline
Exp. 1
Global Declarations and Comments

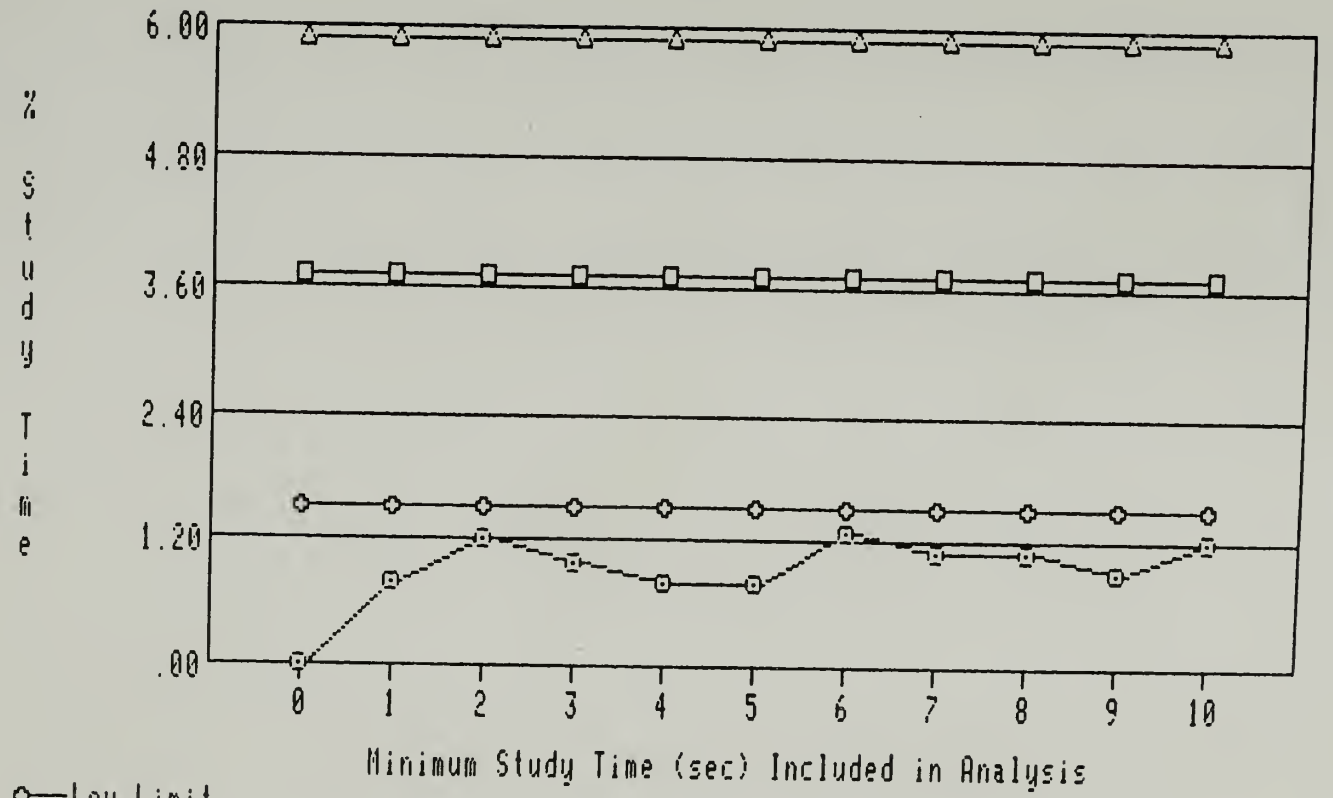

45.00
36.00
27.00
18.00
9.00
.00
% Study Time
0 1 2 3 4 5 6 7 8 9 10
Minimum Study Time (sec) Included in Analysis
Low Limit
High Limit
Baseline
Exp. 1
Uninterpretable



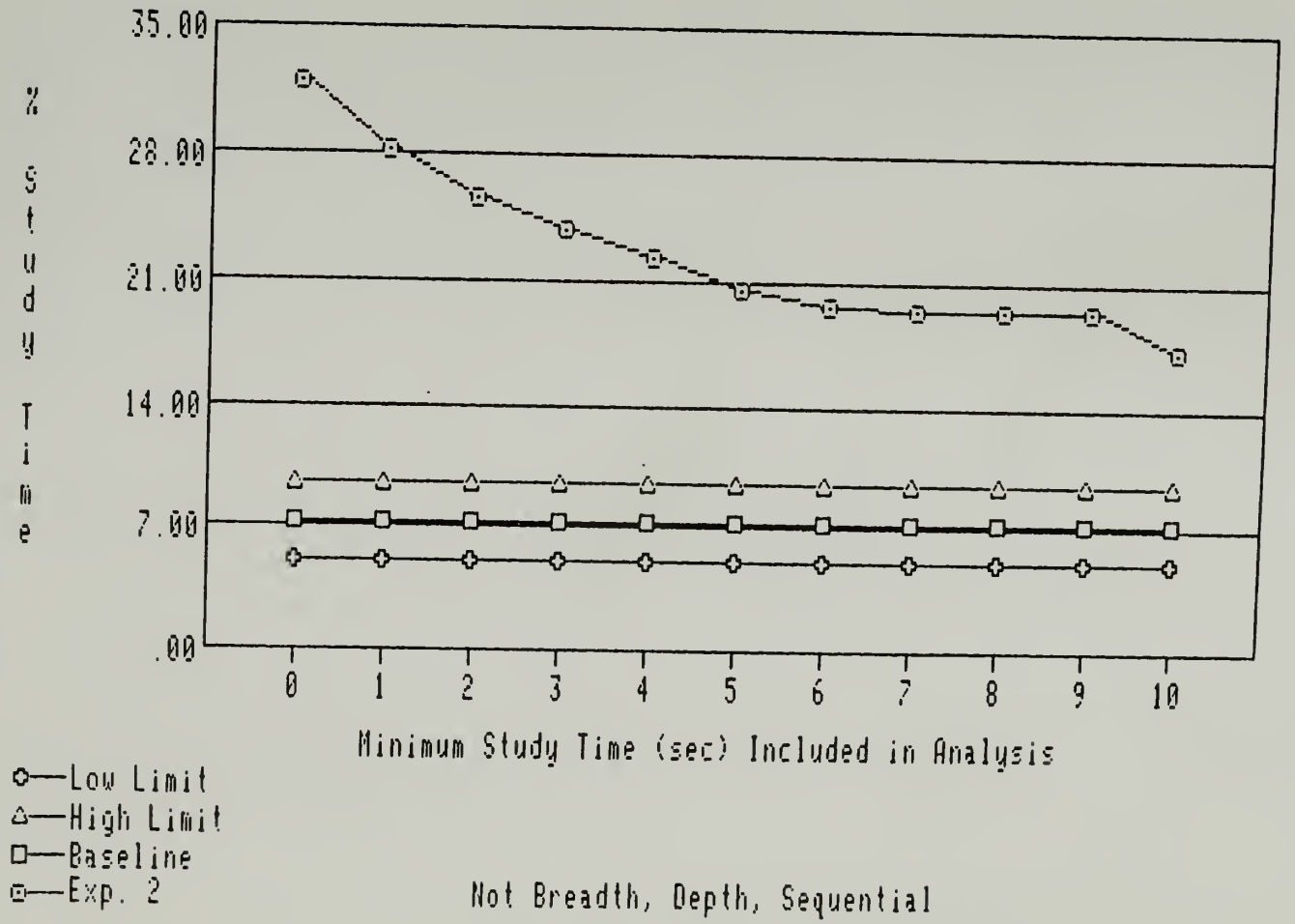
Breadth, Depth, Not Sequential



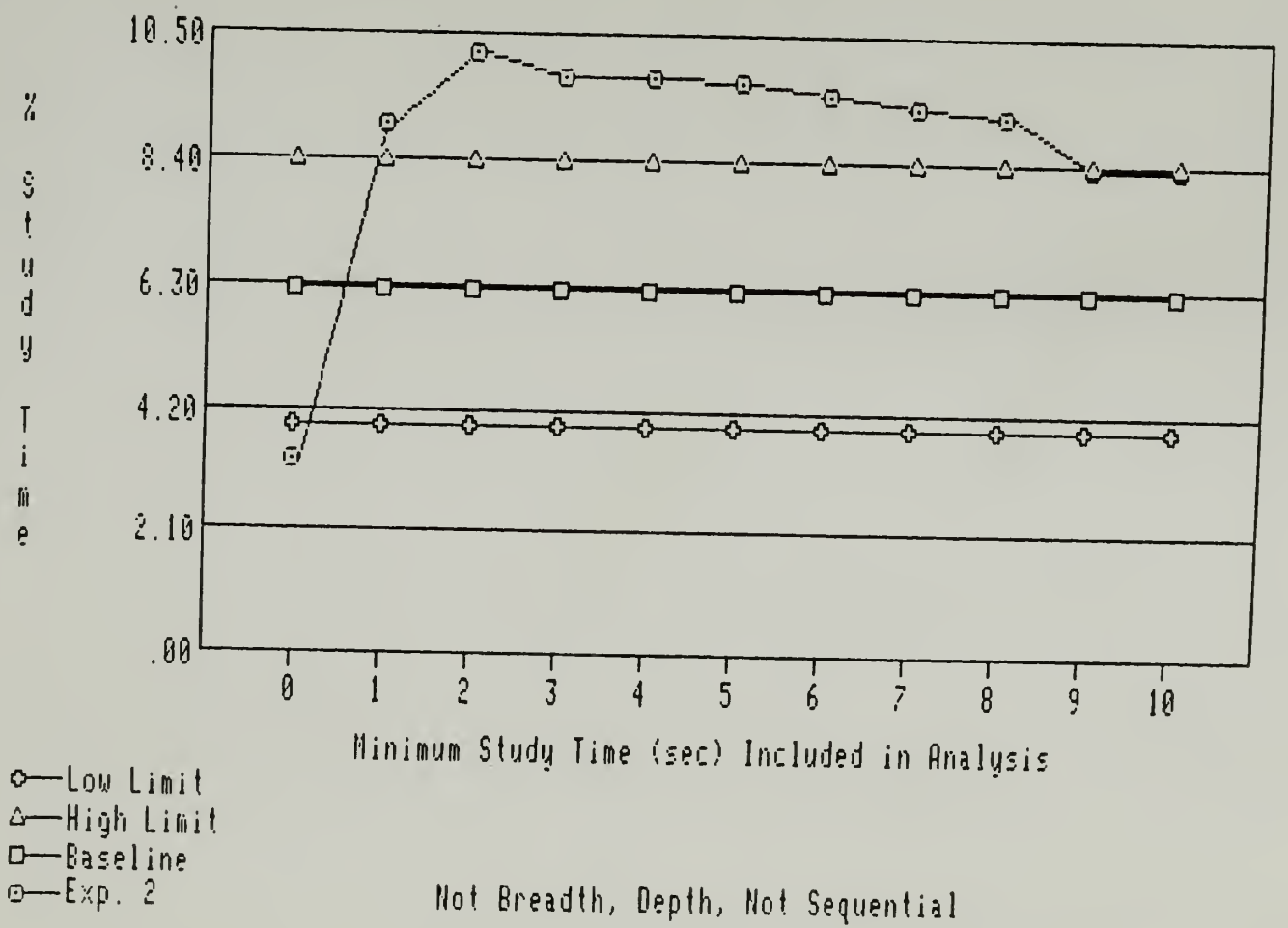
Breadth, Not Depth, Sequential



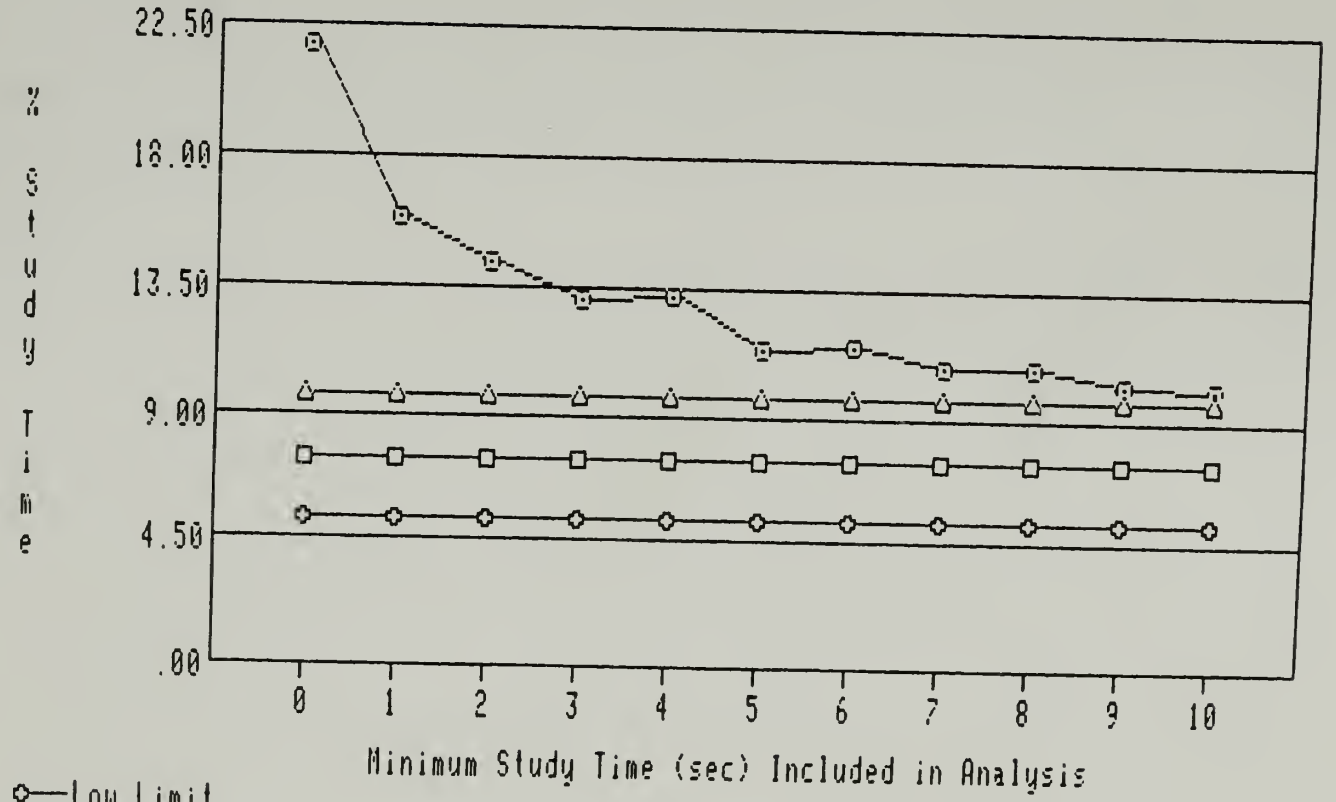
Breadth, Not Depth, Not Sequential



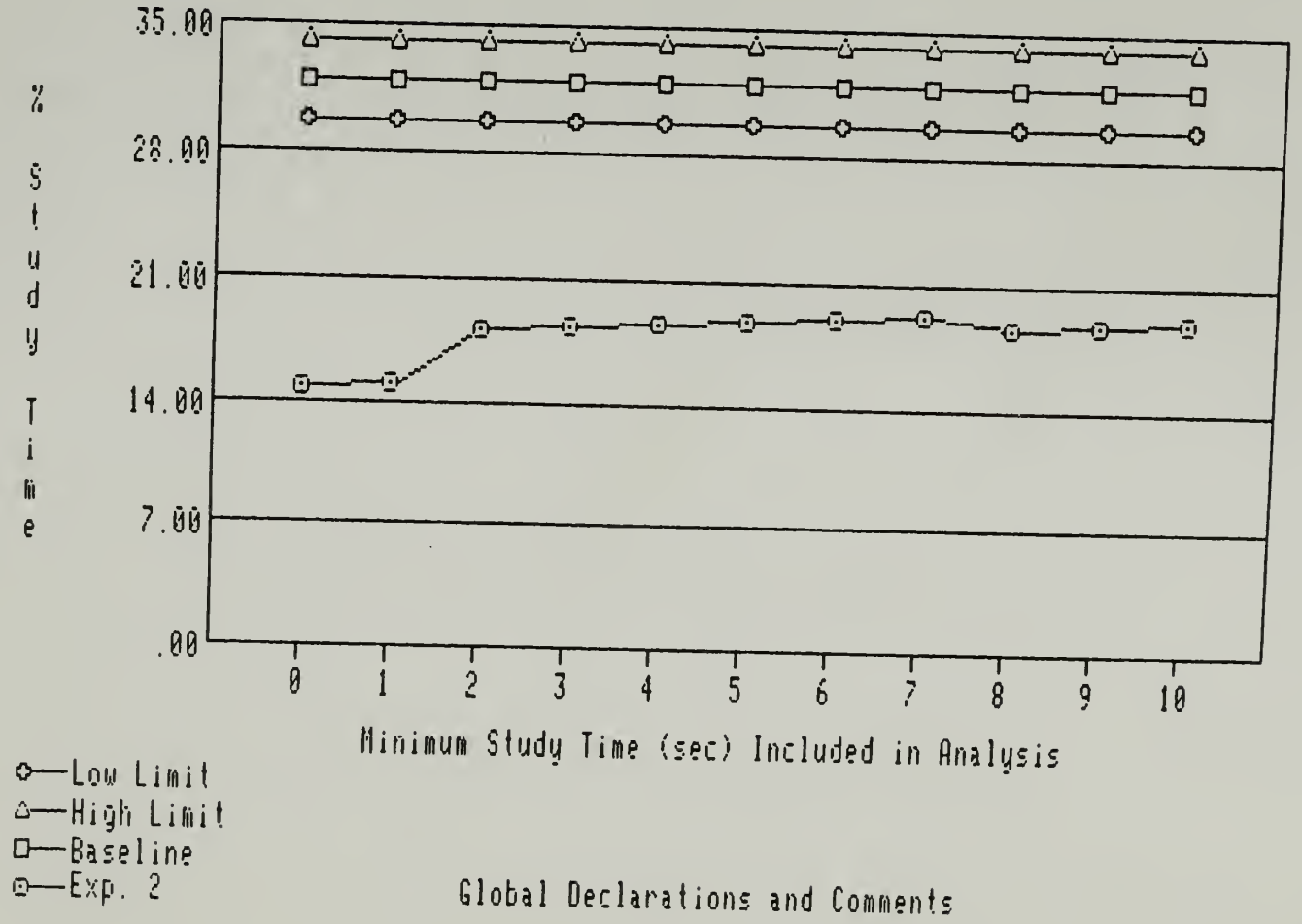
Not Breadth, Depth, Sequential



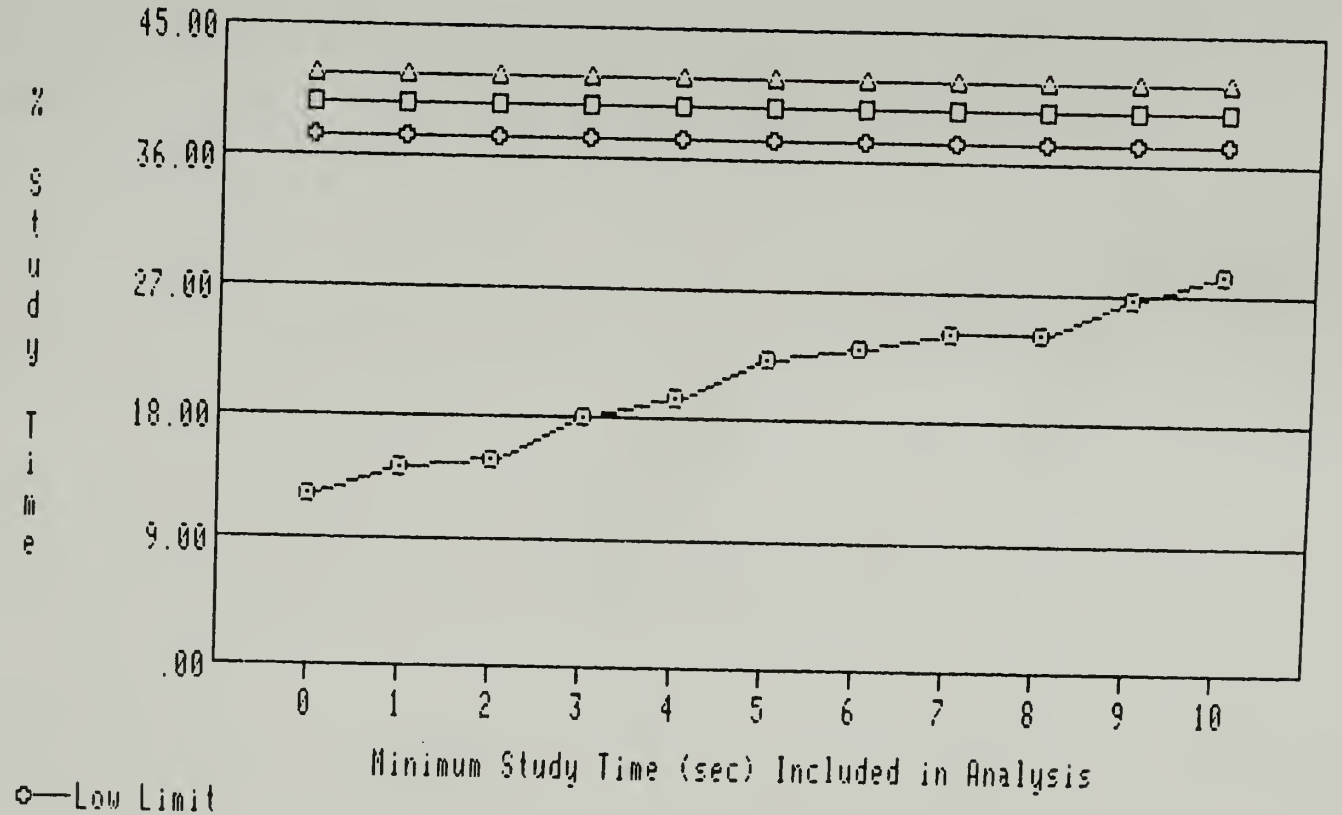
Not Breadth, Depth, Not Sequential



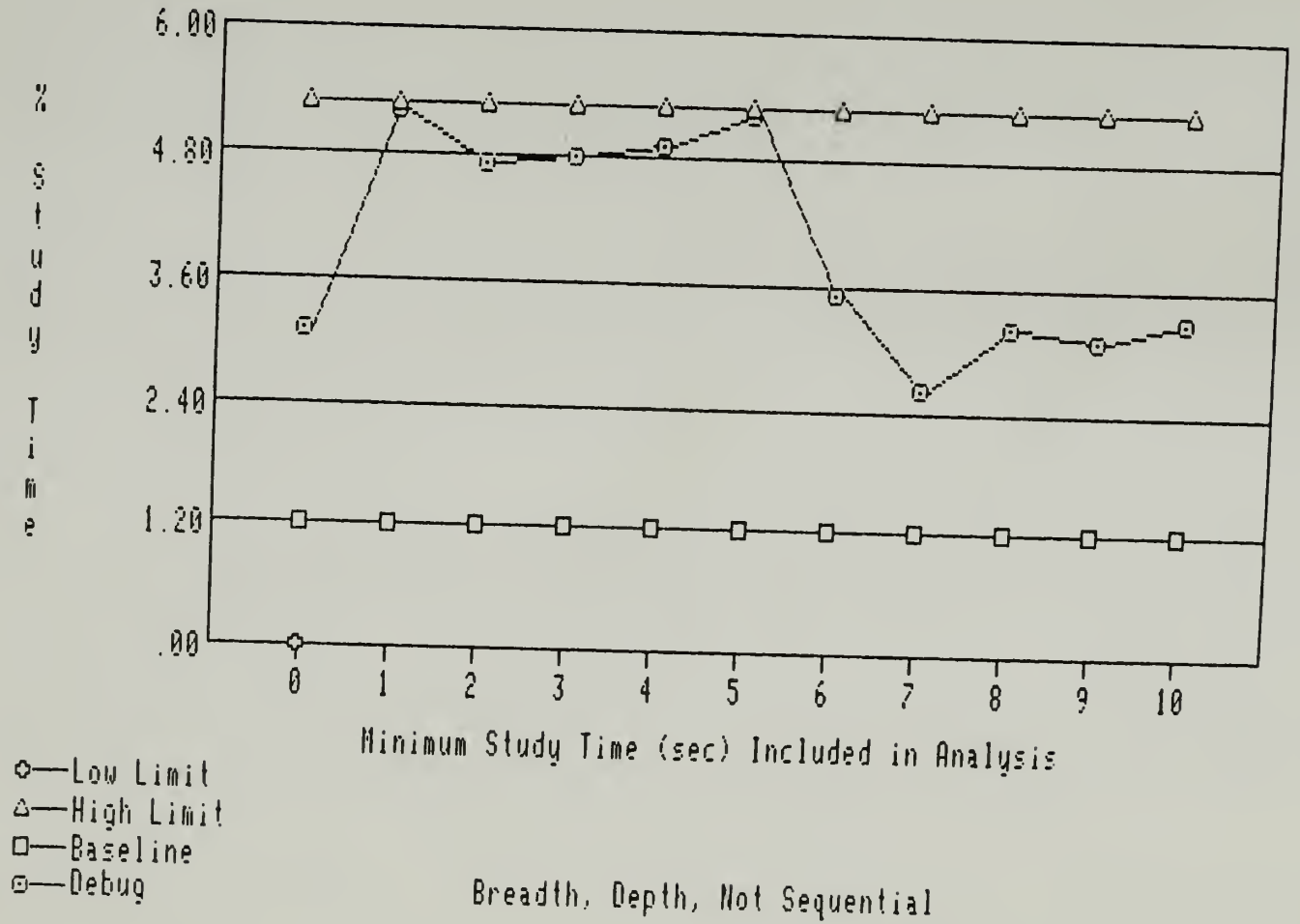
Not Breadth, Not Depth, Sequential



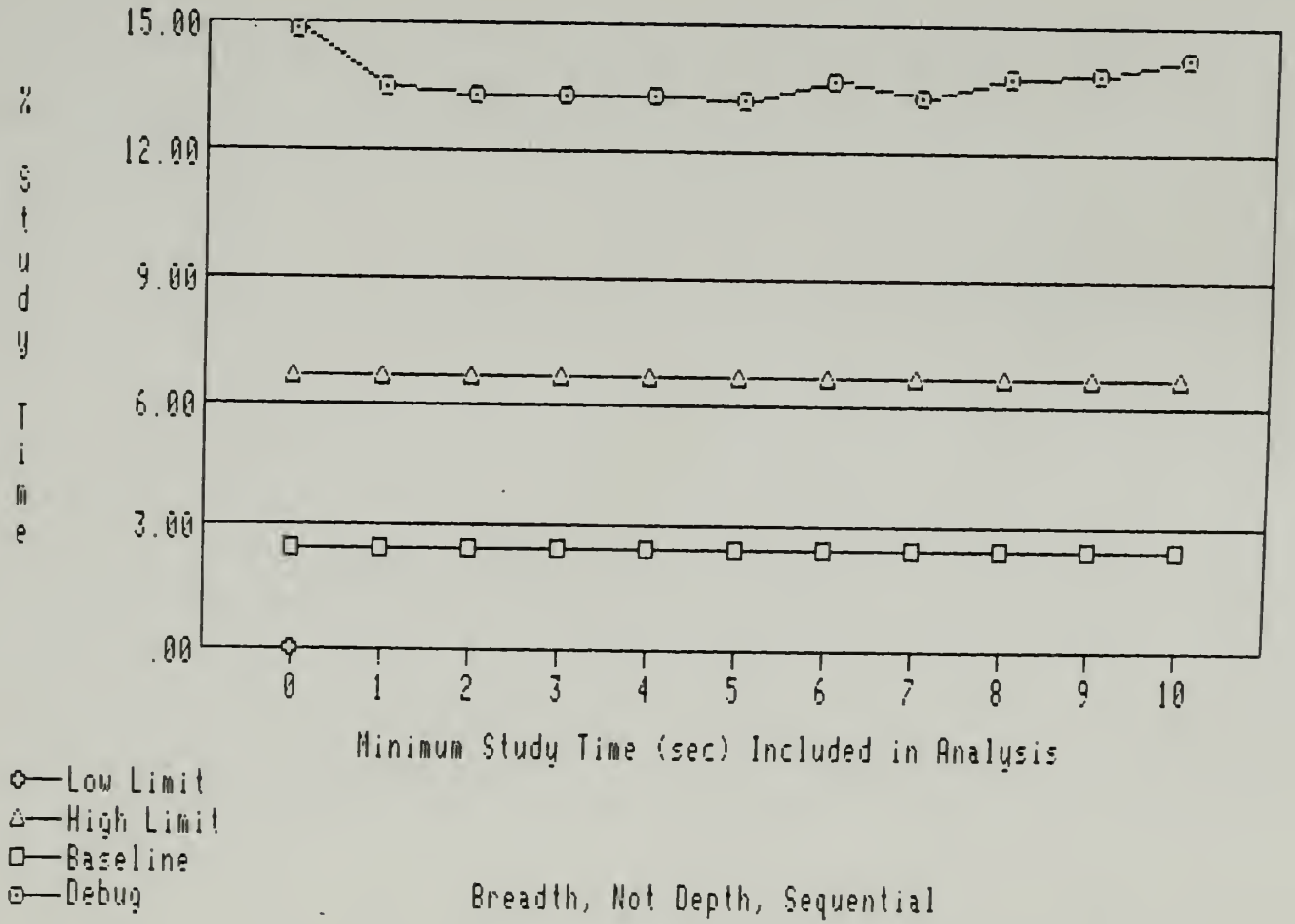
Global Declarations and Comments



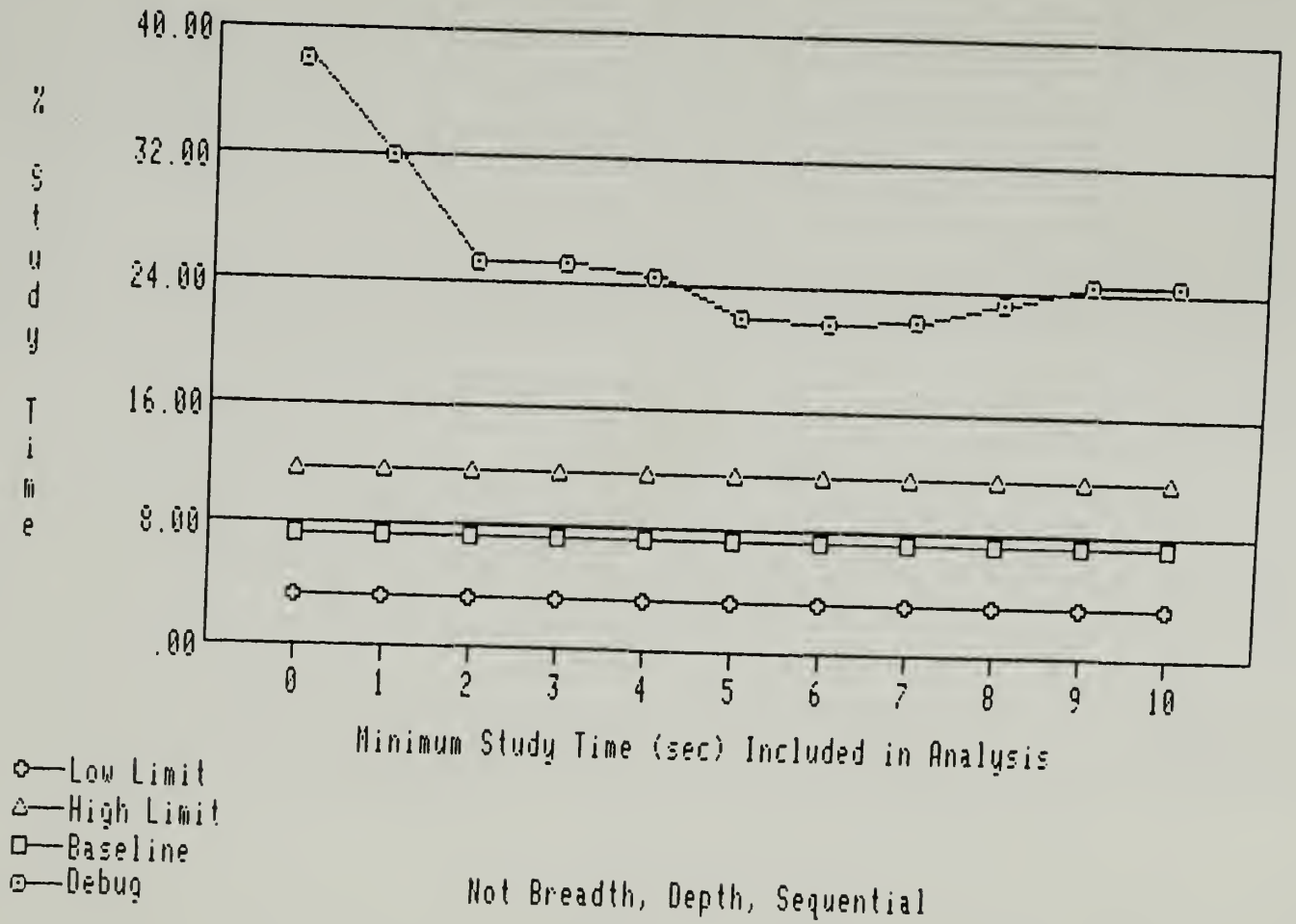
Uninterpretable



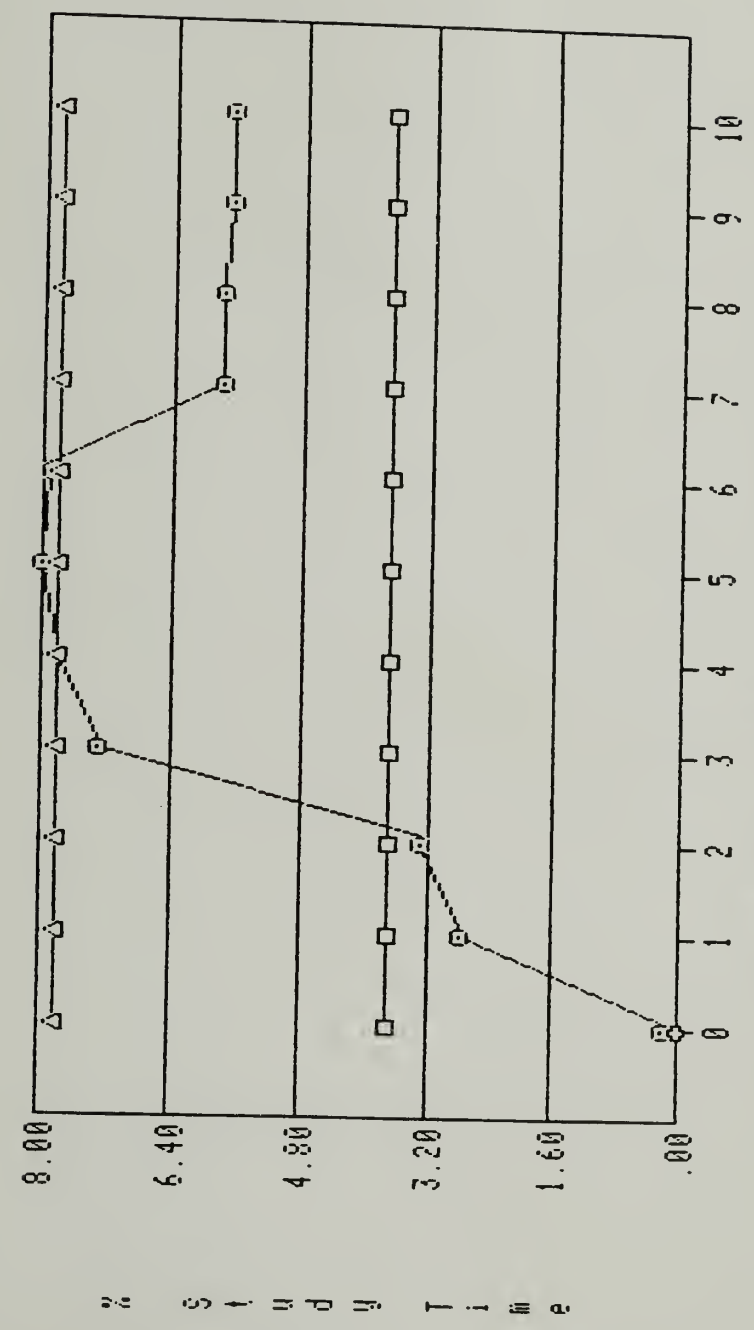
Breadth, Depth, Not Sequential

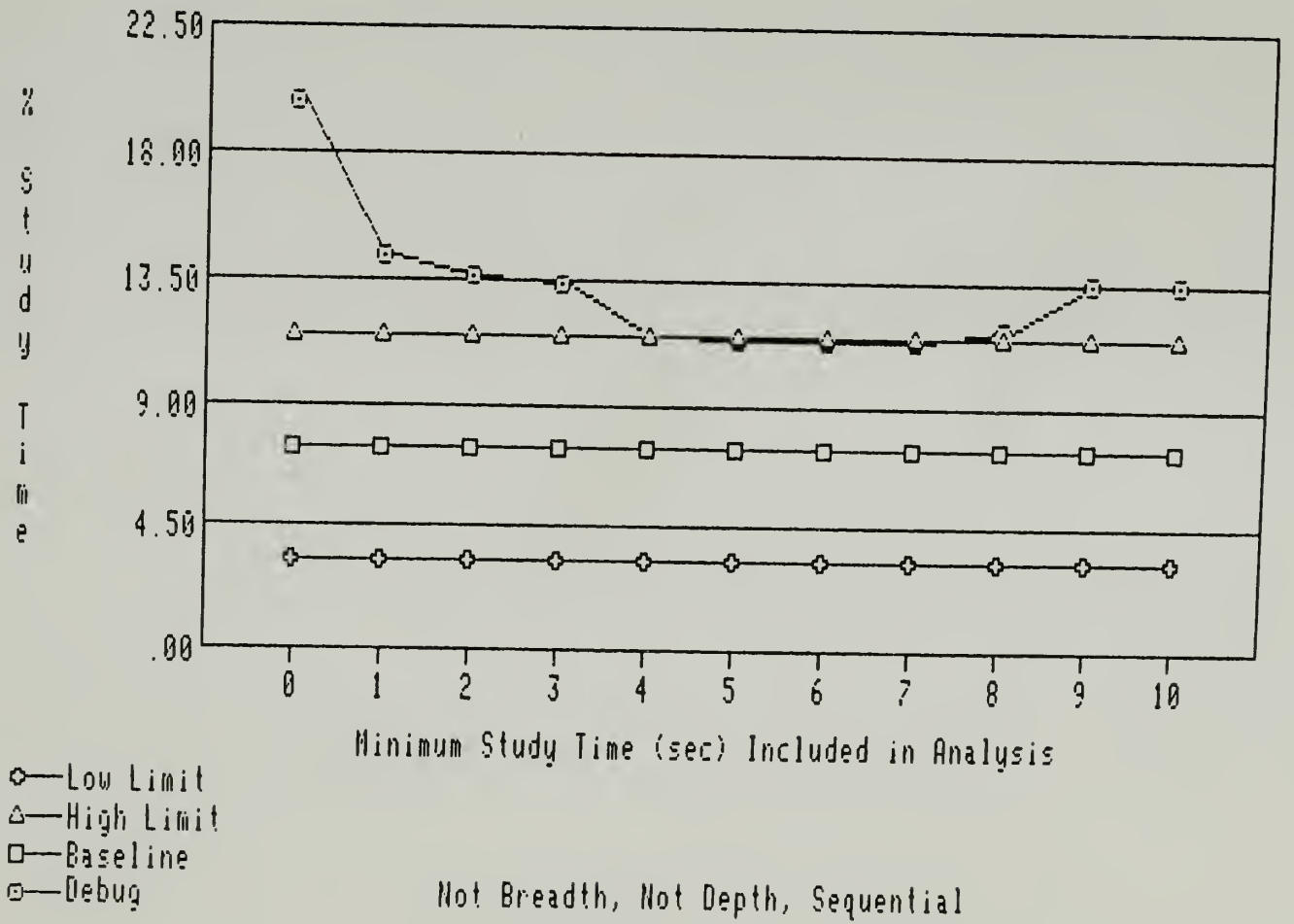


Breadth, Not Depth, Sequential

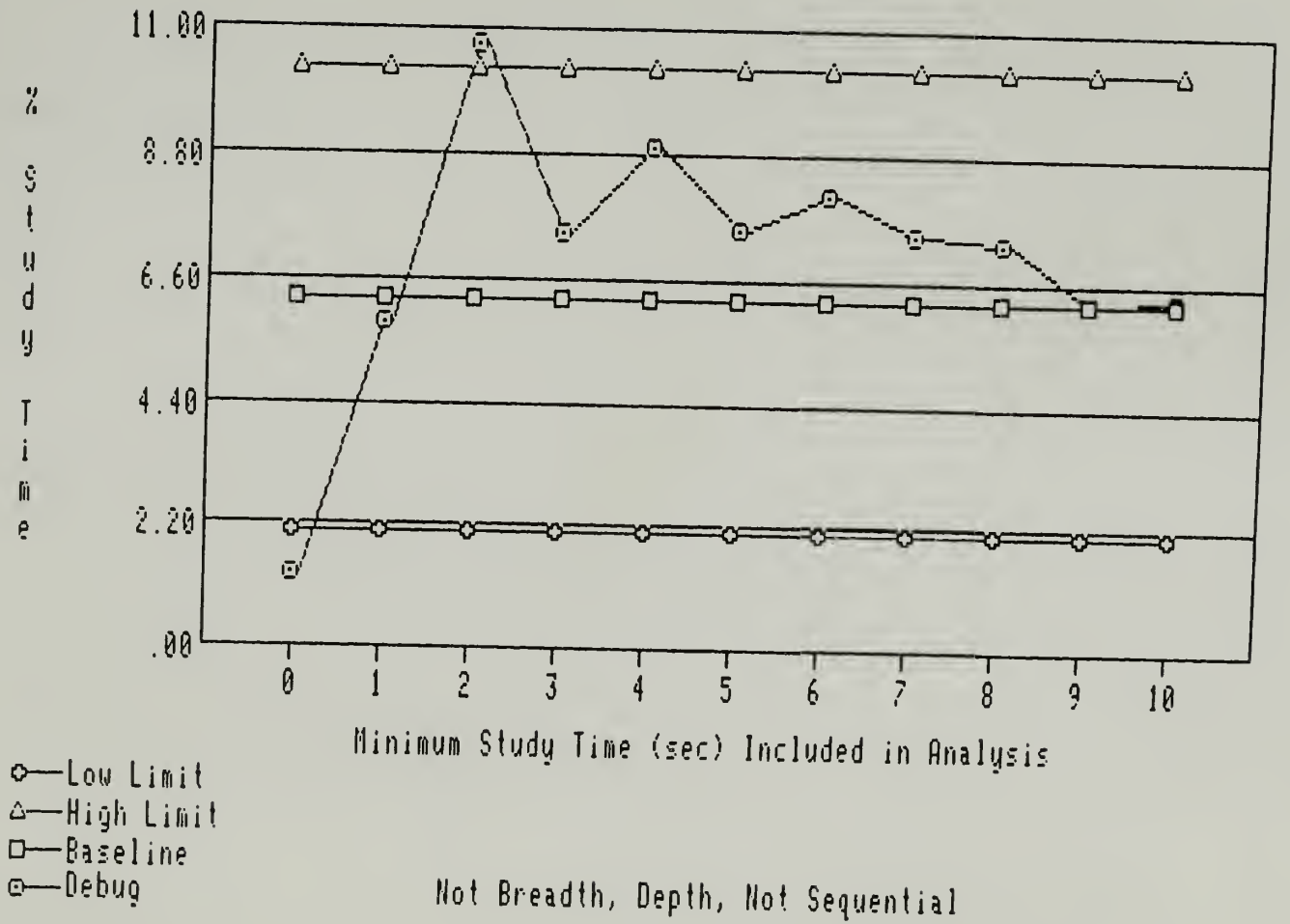


Not Breadth, Depth, Sequential

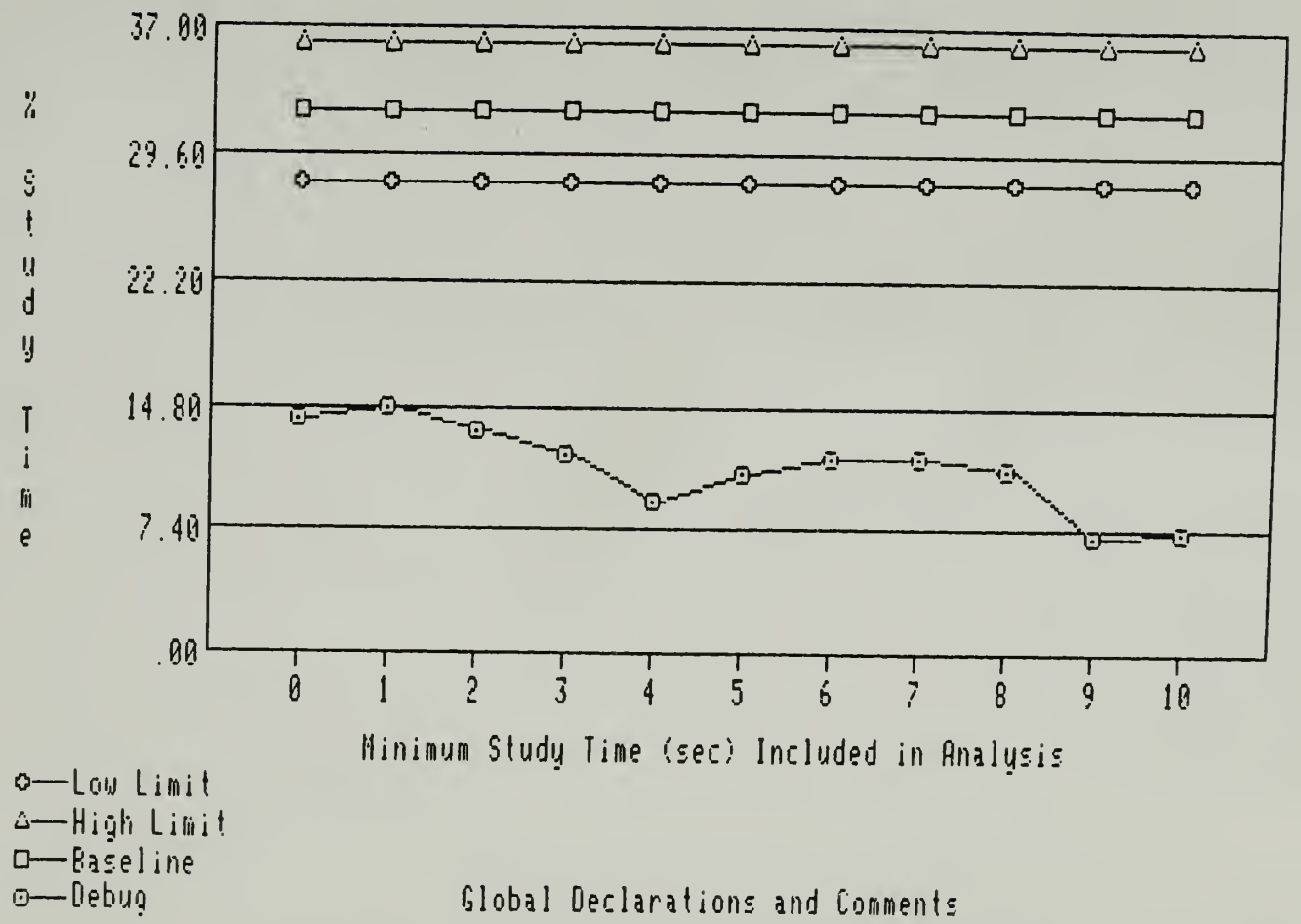

8.00
6.40
4.80
3.20
1.60
.00
Study Time
0 1 2 3 4 5 6 7 8 9 10
Minimum Study Time (sec) Included in Analysis
Low Limit
High Limit
Baseline
Debug
Breadth, Not Depth, Not Sequential



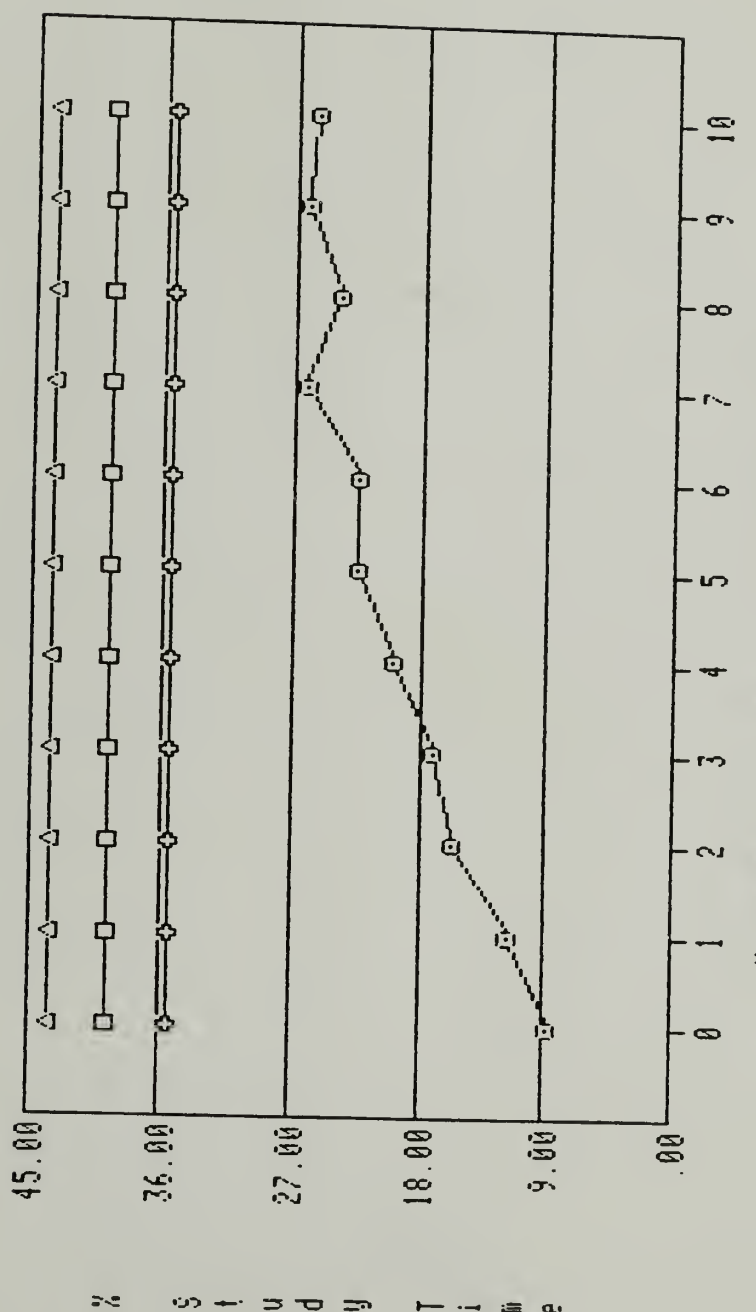
Not Breadth, Not Depth, Sequential



Not Breadth, Depth, Not Sequential



Global Declarations and Comments


45.00
36.00
27.00
18.00
9.00
.00
% Study Time
0 1 2 3 4 5 6 7 8 9 10
Minimum Study Time (sec) Included in Analysis
Low Limit
High Limit
Baseline
Debug
Uninterpretable