

## Calculation of concordance and discordance counts using an AVL search tree

By R. B. Newson

Department of Environmental and Preventive Medicine, Medical College of St.

Bartholomew's Hospital, Charterhouse Square, London EC1M 6BQ.

*Keywords:* AVL search tree; concordance and discordance counts; interval estimation; jackknife method; Kendall's tau-a

## Language

Fortran 77

## Purpose

The concordance of two bivariate observations  $(x, y)$  and  $(x', y')$  is defined as

$$a(x, y, x', y') = \begin{cases} 1, & x < x', y < y' \text{ or } x' < x, y' < y \\ 0, & \text{otherwise,} \end{cases} \quad (1)$$

and their discordance is defined as

$$b(x, y, x', y') = \begin{cases} 1, & x < x', y' < y \text{ or } x' < x, y < y' \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Consider a Fortran data matrix whose number of rows is stored in the *INTEGER* variable *NOBS* and whose columns are parallel arrays *X* and *Y*, of type *REAL*, and *FREQ*, of type *INTEGER*, serving as a frequency weighting variable. If the rows of this matrix are sorted in nondescending order of *Y*, then the algorithm presented here augments the matrix with two additional columns, *INTEGER* arrays *A* and *B*, such that, for each index *I*,

$$\begin{aligned} A(I) &= \sum_{J=1}^{NOBS} FREQ(J) * a(X(I), Y(I), X(J), Y(J)), \\ B(I) &= \sum_{J=1}^{NOBS} FREQ(J) * b(X(I), Y(I), X(J), Y(J)). \end{aligned} \quad (3)$$

The *A(I)* and *B(I)* are termed concordance counts and discordance counts, respectively. If the

algorithm calculated the individual  $a(X(I), Y(I), X(J), Y(J))$  and  $b(X(I), Y(I), X(J), Y(J))$ , then the computing time would be of the order of the square of *NOBS*. The present algorithm produces a search tree of all the observed  $X$ -values and uses it to compute the concordance and discordance counts in a time of order  $NOBS \cdot \log(NXVALS)$ , where  $NXVALS$  is the number of distinct  $X$ -values.

### Theory

The algorithm was developed to assist in the interval estimation of concordance-discordance functionals such as Kendall's tau-a, which, given a discrete bivariate probability mass function  $f(\cdot, \cdot)$ , can be defined as

$$\tau = 2 \sum_x \sum_y \sum_{x' < x} \sum_{y' < y} \{f(x, y)f(x', y') - f(x, y')f(x', y)\}. \quad (4)$$

Given a sample of independent bivariate observations  $\{(X_i, Y_i): 1 \leq i \leq n\}$ , sampled from a common population, we define

$$\begin{aligned} t_{ij} &= a(X_i, Y_i, X_j, Y_j) - b(X_i, Y_i, X_j, Y_j) \\ t_{i.} &= \sum_{j=1}^n t_{ij} \\ t_{..} &= \sum_{i=1}^n t_{i.} \\ \hat{\tau} &= n^{-1}(n-1)^{-1} t_{..} \end{aligned} \quad (5)$$

The statistic  $\hat{\tau}$  is the  $\tau_a$  of Kendall (1970), Chapter 3, Section 3.4, and is unbiased for the  $\tau$  of (4). As it is a U-statistic, it can be used in interval estimation of  $\tau$  by the jackknife method proposed in Arvesen (1969). The  $i$ 'th pseudo-value is

$$v_i = (n-1)^{-1} t_{..} - (n-2)^{-1} (t_{..} - 2t_{i.}). \quad (6)$$

This method can be extended to the interval estimation of the difference between two tau-a functionals, which might be of interest if we wish to compare variables  $W$  and  $X$ , say, as

predictors of a variable  $Y$ . Given algorithms for sorting and for calculating sums and sums of squares, the key to calculating the jackknife statistics is the efficient calculation of the  $t_i$ , which is trivial given the concordance and discordance counts.

### Method

#### *Definitions: search tree data structure*

The search tree takes the form of a second matrix, whose columns are the *REAL* array *XVAL* and the *INTEGER* arrays *LEFT*, *RIGHT*, *NEQ*, *NLS* and *NRS*. A possible search tree is illustrated in Fig. 1. The indices of the tree matrix correspond to the nodes of the tree, one node for each of the distinct  $X$ -values encountered in the input data matrix, and the  $P$ th  $X$ -value encountered is stored in  $XVAL(I)$ . The total number of  $X$ -values is stored in the *INTEGER* location *NXVALS*. The arrays *LEFT* and *RIGHT* are of pointers, such that  $LEFT(I)$  and  $RIGHT(I)$  point to nothing if they are zero and to the indices of the left and right daughter nodes, respectively, if they are positive. For indices  $L$  and  $M$ , a path from  $L$  to  $M$  is a sequence of indices  $(J_1, \dots, J_k)$  such that  $J_1=L$ ,  $J_k=M$ , and, if  $1 \leq i < k$ , then either  $J_{i+1}=LEFT(J_i)$  or  $J_{i+1}=RIGHT(J_i)$ .

The set of indices forms a tree because there is an index, stored in the *INTEGER* location *ROOT*, such that, for each index  $I$ , there is a unique path from *ROOT* to  $I$ . For each index  $I$ , we define  $tree(I)$  as the set of indices  $J$  such that a path exists from  $I$  to  $J$ . The left subtree of  $I$ , denoted  $ltree(I)$ , is defined as the empty set if  $LEFT(I)=0$  and as  $tree(LEFT(I))$  otherwise, and the right subtree of  $I$ , denoted  $rstree(I)$ , is the empty set if  $RIGHT(I)=0$  and  $tree(RIGHT(I))$  otherwise. The set of all indices,  $tree(ROOT)$ , is a search tree because, for each index  $I$ ,  $XVAL(J) < XVAL(I)$  for all  $J \in ltree(I)$  and  $XVAL(J) > XVAL(I)$  for all  $J \in rstree(I)$ .

The array *NEQ* contains frequencies, so that the tree can represent a frequency distribution of  $X$ -values. For each index  $J$ ,  $NEQ(J)$  is the frequency of  $XVAL(J)$ . The location  $NLS(I)$  is used to store the sum of  $NEQ(J)$  over all  $J \in ltree(I)$ , and the location  $NRS(I)$  is used to store the sum of  $NEQ(J)$  over all  $J \in rstree(I)$ .

The tree is constructed to have the AVL property defined in Adel'son-Vel'skii and Landis (1962) and discussed in Wirth (1976), Subsection 4.4.6. This property has the consequence that the possible length of a path from *ROOT* to an index *I*, and therefore the computational time taken to access the *I*th row of the tree matrix, is bounded above by an expression of the order of the logarithm of the number of *X*-values. Fig. 1 represents a tree with the AVL property. Each index *I* is represented by a box, labelled with the value *I*, and containing *NEQ(I)*, *NLS(I)*, *NRS(I)*, *LEFT(I)* and *RIGHT(I)*. The box is joined by lines to *LEFT(I)* and *RIGHT(I)*, if these are nonzero. The tree could represent a frequency distribution in which the only *X*-values are 2, 4, 6, 8 and 9, with total frequencies 2, 4, 1, 3 and 2, respectively.

*Definitions: elementary tree search operations*

The method of the algorithm is built up out of elementary tree search operations, involving an individual *X*-value stored in a *REAL* location *XCUR* and the location of that value in the tree by iteration along the unique path (*J*<sub>1</sub>, ... , *J*<sub>*k*</sub>), such that *J*<sub>1</sub>=*ROOT* and *XVAL(J*<sub>*k*</sub>)=*XCUR*. These operations are as follows:

1. The extraction of the left and right subtotals of *XCUR* from the tree.
2. The updating and downdating of the tree with the value *XCUR* and a frequency stored in an *INTEGER* location *FRCUR*.

The left subtotal of *XCUR* is defined as the sum of all *NEQ(I)* for all *I* such that *XVAL(I)*<*XCUR*, and is equal to

$$NLS(J_k) + \sum_{i: XVAL(J_i) < XCUR} \{NEQ(J_i) + NLS(J_i)\}, \quad (7)$$

and, similarly, the right subtotal is the sum of all *NEQ(I)* for *I* such that *XVAL(I)*>*XCUR*, and is equal to

$$NRS(J_k) + \sum_{i: XVAL(J_i) > XCUR} \{NEQ(J_i) + NRS(J_i)\}. \quad (8)$$

The update of the tree with *X*-value *XCUR* and frequency *FRCUR* is the addition of *FRCUR* to the frequency of the value *XCUR* in the frequency distribution represented by the

tree. It is performed by iterating along the path  $(J_1, \dots, J_k)$  and performing the following assignments for  $i$  from 1 to  $k$ :

1. If  $XCUR < XVAL(J_i)$ , then increment  $NLS(J_i)$  by  $FRCUR$ .
2. If  $XCUR > XVAL(J_i)$ , then increment  $NRS(J_i)$  by  $FRCUR$ .
3. If  $XCUR = XVAL(J_i)$ , then increment  $NEQ(J_i)$  by  $FRCUR$ .

The downdate of the tree with  $X$ -value  $XCUR$  and frequency  $FRCUR$  is the subtraction of  $FRCUR$  from the frequency of the value  $XCUR$  in the frequency distribution represented by the tree. It is performed as the update, except for substituting "decrement" for "increment" throughout.

#### *Summary of the method*

The search tree is constructed in one pass through the data matrix. The values of  $NEQ(I)$ ,  $NLS(I)$  and  $NRS(I)$  are then initialised to zero for all  $I$ . The calculation of concordance and discordance counts is carried out in two further passes through the data matrix, using a *REAL* location  $YCUR$ , which, in each pass, takes on successive  $Y$ -values, in ascending order from the lowest to the highest.

In the second pass through the data matrix, the successive values of  $YCUR$  are processed as follows:

1. For each index  $J$  such that  $Y(J) = YCUR$ , set  $A(J)$  equal to the left subtotal of  $X(J)$  and set  $B(J)$  to the right subtotal of  $X(J)$ .
2. For each index  $J$  such that  $Y(J) = YCUR$ , update the tree with value  $X(J)$  and frequency  $FREQ(J)$ .

In the third pass through the data matrix, the successive values of  $YCUR$  are processed as follows:

1. For each index  $J$  such that  $Y(J) = YCUR$ , downdate the tree with value  $X(J)$  and frequency  $FREQ(J)$ .
2. For each index  $J$  such that  $Y(J) = YCUR$ , add the left subtotal of  $X(J)$  to  $B(J)$  and add the right subtotal of  $X(J)$  to  $A(J)$ .

## Structure

*SUBROUTINE CONDIS*(*NOBS*, *X*, *Y*, *FREQ*, *A*, *B*, *MXVAL*, *XVAL*, *NIWORK*, *IWORK*,  
*IFAULT*)

*Formal parameters*

<i>NOBS</i>	Integer	input: the number of observations (rows) of the data matrix ( $\geq 1$ )
<i>X</i>	Real array of dimension $\geq NOBS$	input: the <i>X</i> -variate of the data matrix
<i>Y</i>	Real array of dimension $\geq NOBS$	input: the <i>Y</i> -variate of the data matrix, which is assumed to be sorted in a nondescending order of <i>Y</i>
<i>FREQ</i>	Integer array of dimension $\geq NOBS$	input: the frequency weightings of the data matrix
<i>A</i>	Integer array of dimension $\geq NOBS$	output: the concordance counts of <i>X</i> and <i>Y</i> , weighted by <i>FREQ</i>
<i>B</i>	Integer array of dimension $\geq NOBS$	output: the discordance counts of <i>X</i> and <i>Y</i> , weighted by <i>FREQ</i>

<i>MXVAL</i>	Integer	input: the maximum number of <i>X</i> -values ( $\geq 1$ )
<i>XVAL</i>	Real array of dimension $\geq MXVAL$	workspace: the <i>X</i> -values encountered in the array <i>X</i> , forming a search tree
<i>NIWORK</i>	Integer	input: the dimension of the workspace array <i>IWORK</i> : $NIWORK \geq 5 * MXVAL$
<i>IWORK</i>	Integer array of dimension $\geq NIWORK$	workspace: the integer arrays used to create and define the search tree
<i>IFAULT</i>	Integer	output: a fault indicator = 1 if $NOBS < 1$ = 2 if $MXVAL < 1$ = 3 if $NIWORK < 5 * MXVAL$ = 4 if number of <i>X</i> -values encountered > $MXVAL$

Subroutine *CONDIS* checks for errors in the input arguments. If there are none, then *CONDIS* calls *MAKTRE* in an attempt to build the search tree. If there are no more than *MXVAL* distinct *X*-values, then the tree is built successfully, and *CONDIS* calls *CALCD*, which uses the tree to calculate *A* and *B*. If there are errors in the input arguments, or more than *MXVAL* distinct *X*-values, then *CONDIS* sets all entries of *A* and *B* to zero and terminates with the appropriate error code in *IFAULT*.

*SUBROUTINE MAKTRE*(*NOBS*, *X*, *MXVAL*, *NXVAL*, *ROOT*, *XVAL*, *LEFT*, *RIGHT*,  
*MOTHER*, *BALANG*, *IFault* )

*Formal parameters*

<i>NOBS</i>	Integer	input: as in <i>CONDIS</i>
<i>X</i>	Real array of dimension $\geq NOBS$	Input: as in <i>CONDIS</i>
<i>MXVAL</i>	Integer	Input: as in <i>CONDIS</i>
<i>NXVAL</i>	Integer	Output: number of distinct values encountered in <i>X</i> if this is no more than <i>MXVAL</i> , otherwise set to <i>MXVAL</i> (or to 0 if either <i>NOBS</i> or <i>MXVAL</i> is less than 1)
<i>ROOT</i>	Integer	Output: index pointing to root node of search tree (or 0 if either <i>NOBS</i> or <i>MXVAL</i> is less than 1)
<i>XVAL</i>	Real array of dimension $\geq MXVAL$	Output: the first <i>NXVAL</i> entries contain the first <i>NXVAL</i> distinct values encountered in the array <i>X</i>



<i>LEFT</i>	Integer array of dimension $\geq MXVAL$	Output: the first <i>NXVAL</i> entries contain the left daughter node indices for the search tree of values in <i>XVAL</i>
<i>RIGHT</i>	Integer array of dimension $\geq MXVAL$	Output: the first <i>NXVAL</i> entries contain the right daughter node indices for the search tree of values in <i>XVAL</i>
<i>MOTHER</i>	Integer array of dimension $\geq MXVAL$	Output: the first <i>NXVAL</i> entries contain the mother node indices for the search tree of values in <i>XVAL</i> (except <i>MOTHER(ROOT)</i> , which contains zero if <i>ROOT</i> >0)
<i>BALANC</i>	Integer array of dimension $\geq MXVAL$	Output: the first <i>NXVAL</i> entries contain the indicators of balance between left and right subtrees for the nodes of the search tree of values in <i>XVAL</i> (see Wirth (1976), Subsection 4.4.6)
<i>IFault</i>	Integer	Output: a fault indicator =1 if <i>NOBS</i> <1 =2 if <i>MXVAL</i> <1 =3 if number of distinct <i>X</i> -values exceeds <i>MXVAL</i>

Subroutine *MAKTRE* is a potentially stand-alone subroutine to produce an AVL search

tree of values of  $X$ . When *MAKTRE* is called by *CONDIS*, the outputs *MOTHER* and *BALANC* are not used by *CONDIS* after return, and their space is reused for other purposes by *CALCD*, so they are effectively workspace for use inside *MAKTRE* only.

*SUBROUTINE CALCD(NOBS, X, Y, FREQ, A, B, NXVAL, ROOT, XVAL, LEFT, RIGHT, NEQ, NLS, NRS )*

*Formal parameters*

<i>NOBS</i>	Integer	Input: as in <i>CONDIS</i>
<i>X, Y</i>	Real arrays of dimension $\geq NOBS$	Input: as in <i>CONDIS</i>
<i>FREQ</i>	Integer array of dimension $\geq NOBS$	Input: as in <i>CONDIS</i>
<i>A, B</i>	Integer arrays of dimension $\geq NOBS$	Output: as in <i>CONDIS</i>
<i>NXVAL, ROOT</i>	Integer	Input: as output from <i>MAKTRE</i>
<i>XVAL</i>	Real array of dimension $\geq NXVAL$	Input: as output from <i>MAKTRE</i>

*LEFT, RIGHT* Integer arrays            Input: as output from *MAKTRF*  
                   of dimension  
                    $\geq NXVAL$

*NEQ, NLS, NRS* Integer arrays        Workspace: as described above in Method  
                   of dimension  
                    $\geq NXVAL$

*CALCD* is called from *CONDIS* and performs the second and third passes through the data matrix, as described above in the summary of the method, to calculate the concordance counts in *A* and the discordance counts in *B*.

#### Restrictions

The algorithm does not perform the original sort by nondescending *Y*, or check that the data matrix is indeed sorted in this way. (Successive, distinct values of *Y* encountered by *CALCD* are assumed to be ascending. It was thought that users would probably have access to better sorting routines than could be provided by the present author.)

#### Time

The algorithm was tested for performance against an alternative subroutine, *SUBROUTINE CDTRIV(NOBS, X, Y, FREQ, A, B, IFAULT)*, whose parameters function as those of the same name for *CONDIS*, and which calculates the concordance and discordance counts "trivially" by counting the individual concordances and discordances. The programs were tested in batch jobs on a VAX 3600 computer under the VMS operating system.

Each test involved the processing of 20 data matrices of equal size, whose *X, Y*-pairs were taken from a stream of pseudorandom bivariate normal data with zero *X, Y*-correlation. The data matrices were stored in an unformatted binary disk file and read by simple driver programs, which called the appropriate concordance-discordance program and output the

augmented matrices to another unformatted binary disk file. For a fair comparison, the processing time for *CONDIS* should include the time taken to sort the data matrices by nondescending  $Y$  before calling *CONDIS* and to sort them back to the original order afterwards. There were therefore two driver programs written to call *CONDIS*, one calling subroutines to carry out both sorts using the Heapsort method (see Wirth (1976), Section 2.2.5), and the other performing no sorts. To assess the contribution to CPU time of input-output and of fixed time costs unrelated to the number of observations, a further driver program was written, performing the same input-output functions as the others but not calling any concordance-discordance subroutines.

Table 1 gives the CPU times for the input-output only, the tree method with and without sorting, and the trivial method, for varying numbers of observations per data matrix. The input (and output) data matrices were sorted by  $Y$  for the tree method without sorting, and presented in the original order for the three other methods. Fig. 2 gives the same data graphically for data matrix sizes up to 1000, above which the CPU time for the trivial method "explodes" quadratically. It can be seen that the time saved by using the tree method becomes considerable for matrix sizes in the middle to upper hundreds, for which the time consumed internally by the trivial method becomes comparable to that consumed by input-output and fixed operations.

#### Test data

A small matrix of test data, representing a 3x3 contingency table, is given in Table 2, together with the correct values of  $A$  and  $B$ .

#### Remarks

A trivial extension to the above algorithm is to allow the possibility that the  $X$ -values, the  $Y$ -values or both may be possibly censored lifetimes rather than data of known value. This is essentially done in Newson (1988), Chapter 4, although the algorithm there does not guarantee that the tree will have the AVL property.

## References

- Adel'son-Vel'skii, G. M. & Landis, E. M. (1962) An algorithm for the organization of information. *Soviet Mathematics*, **3**, 1259-1263. Translated from the Russian original in *Dokladii Akademii Nauk SSSR*, **146**, 263-266 (1962).
- Arvesen, J. N. (1969) Jackknifing U-statistics. *Annals of Mathematical Statistics*, **40**, 2076-2100.
- Kendall, M. G. (1970) *Rank correlation methods*. Fourth edition. London: Griffin.
- Newson, R. B. (1988) *An analysis of cinematographic cell division data using U-statistics*. Ph.D. thesis, University of Sussex.
- Wirth, N. (1976) *Algorithms + Data structures = Programs*. Englewood Cliffs, New Jersey: Prentice-Hall.

TABLE 1

*Time in CPU seconds to process 20 samples of sample size NOBS using various algorithms*

<i>NOBS</i>	<i>Algorithm</i>			
	<i>I/O only</i>	<i>Tree, no sort</i>	<i>Tree, with sort</i>	<i>Trivial</i>
20	2.74	3.00	2.95	2.94
40	3.25	3.34	3.42	3.24
60	3.44	3.75	3.85	3.72
80	3.82	4.15	4.29	4.28
100	4.23	4.56	4.76	4.72
200	5.88	6.82	7.37	7.80
300	7.71	9.23	9.90	11.99
400	9.30	11.28	12.61	16.82
500	11.13	13.85	15.25	22.81
600	12.91	16.30	18.17	29.72
700	14.68	18.52	21.27	37.92
800	16.01	21.19	23.80	47.26
900	17.99	23.51	26.80	58.08
1000	19.37	25.55	29.71	67.76
2000	36.60	51.67	60.86	237.82
3000	53.78	76.53	93.68	507.55
4000	70.86	104.83	125.30	889.85
5000	86.81	127.93	158.40	1394.04

TABLE 2

*A test data matrix, augmented with correct values of A and B*

<i>X</i>	<i>Y</i>	<i>FREQ</i>	<i>A</i>	<i>B</i>
1.0	1.0	1	28	0
2.0	1.0	2	15	11
3.0	1.0	3	0	24
1.0	2.0	4	17	5
2.0	2.0	5	10	10
3.0	2.0	6	3	15
1.0	3.0	7	0	16
2.0	3.0	8	5	9
3.0	3.0	9	12	0

Figure legends

Fig. 1. An AVL search tree as used by the algorithm

Fig. 2. CPU times for 20 samples as a function of sample size



```

SUBROUTINE CONDIS(NOBS,X,Y,FREQ,A,B,MXVAL,XVAL,NIWORK,IWORK,
+IFAULT)
C SUBROUTINE CONDIS TAKES, AS INPUT, A DATA MATRIX OF ARRAYS
C WITH VARIABLES X, Y AND FREQ, SORTED BY Y,
C AND USES, AS WORKSPACE, THE REAL ARRAY XVAL OF DIMENSION MXVAL
C AND THE INTEGER ARRAY IWORK OF DIMENSION NIWORK
C TO CREATE AND WORK WITH AN AVL SEARCH TREE
C AND GIVES, AS OUTPUT, THE SAME MATRIX AUGMENTED BY VARIABLES A AND B
C (CONCORDANCE AND DISCORDANCE TOTALS BETWEEN X AND Y)
C AND THE ERROR INDICATOR IFAULT
C BEGINNING OF DATA DEFINITION SECTION
  INTEGER NOBS
  REAL X(NOBS),Y(NOBS)
  INTEGER FREQ(NOBS),A(NOBS),B(NOBS)
C NOBS IS NUMBER OF OBSERVATIONS IN DATA MATRIX
C X AND Y ARE X- AND Y-VARIABLES IN DATA MATRIX
C FREQ IS FREQUENCY VARIABLE IN DATA MATRIX
C A AND B ARE CONCORDANCE AND DISCORDANCE VARIABLES IN DATA MATRIX.
  INTEGER MXVAL
  REAL XVAL(MXVAL)
  INTEGER NIWORK
  INTEGER IWORK(NIWORK)
C MXVAL IS MAXIMUM NUMBER OF X-VALUES IN SEARCH TREE
C XVAL IS X-VALUES FOR SEARCH TREE
C NIWORK IS MAXIMUM INTEGER WORK SPACE DIMENSION (MUST BE AT LEAST 5*MXVAL)
C IWORK IS INTEGER WORK SPACE FOR SEARCH TREE
  INTEGER IFAULT
C IFAULT IS INDICATOR THAT SUBROUTINE CONDIS HAS FAILED
C AND IS SET TO 0 IF COMPLETED SUCCESSFULLY
C AND TO 1 IF NOBS LESS THAN 1, 2 IF MXVAL LESS THAN 1,
C 3 IF NWORK LESS THAN 5*MXVAL,
C 4 IF NUMBER OF X-VALUES GREATER THAN MXVAL
  INTEGER NXVAL,ROOT
C NXVAL IS ACTUAL NUMBER OF NODES IN X-VALUE TREE
C ROOT IS POINTER TO ROOT NODE OF X-VALUE TREE
  INTEGER ILEFT,IRIGHT,INEQ,INLS,INRS
C ILEFT,IRIGHT,INEQ,INLS AND INRS ARE ARRAY STARTING ADDRESSES
C IN INTEGER WORK SPACE IWORK
C FOR ARRAYS LEFT, RIGHT, NEQ, NLS AND NRS
C PASSED AS PARAMETERS TO SERVANT SUBROUTINES
C LEFT AND RIGHT ARE ARRAYS OF LEFT AND RIGHT POINTERS FOR NODES
C CORRESPONDING TO X-VALUES IN TREE MATRIX
C NEQ, NLS AND NRS ARE ARRAYS OF NODE TOTALS, LEFT SUBTREE TOTALS
C AND RIGHT SUBTREE TOTALS FOR X-VALUES IN TREE MATRIX
  INTEGER IFAIL
C IFAIL IS ERROR INDICATOR PASSED TO SERVANT SUBROUTINES
  INTEGER OBS
C OBS IS OBSERVATION INDEX FOR DATA MATRIX
C END OF DATA DEFINITION SECTION
C INITIALISE IFAULT TO NO ERROR
  IFAULT=0
C BEGINNING OF INITIAL SCREEN FOR ILL-DEFINED PARAMETERS
  IF(NOBS.LT.1)THEN
    IFAULT=1
    GOTO 1
  ELSE IF(MXVAL.LT.1)THEN
    IFAULT=2
    GOTO 1
  ELSE IF(NIWORK.LT.(5*MXVAL))THEN
    IFAULT=3
    GOTO 1
  END IF
C END OF INITIAL SCREEN FOR ILL-DEFINED PARAMETERS
C BEGINNING OF ARRAY PARAMETER STARTING ADDRESS INITIALISATION SECTION
  ILEFT=1

```

```

IRIGHT=ILEFT+MXVAL
INEQ=IRIGHT+MXVAL
INLS=INEQ+MXVAL
INRS=INLS+MXVAL

```

```

C END OF ARRAY PARAMETER STARTING ADDRESS INITIALISATION SECTION
C BEGINNING OF SEARCH TREE INITIALISATION SECTION
C (ENTAILING FIRST PASS THROUGH DATA MATRIX BY SUBROUTINE MAKTRE)
  CALL MAKTRE(NOBS,X,MXVAL,NXVAL,ROOT,XVAL,
  +IWORK(ILEFT),IWORK(IRIGHT),IWORK(INLS),IWORK(INRS),IFAIL)
C LEAVE FOR FAILURE SALVAGE SECTION IF SEARCH TREE CANNOT BE BUILT
  IF(IFAIL.GT.0)THEN
    IFAULT=4
    GOTO 1
  END IF
C END OF SEARCH TREE INITIALISATION SECTION
C (ENTAILING FIRST PASS THROUGH DATA MATRIX BY SUBROUTINE MAKTRE)
C BEGINNING OF SUCCESSFUL COMPLETION SECTION
C (ENTAILING SECOND AND THIRD PASSES THROUGH DATA MATRIX BY SUBROUTINE CALCD)
  CALL CALCD(NOBS,X,Y,FREQ,A,B,NXVAL,ROOT,XVAL,
  +IWORK(ILEFT),IWORK(IRIGHT),IWORK(INEQ),IWORK(INLS),IWORK(INRS))
  RETURN
C END OF SUCCESSFUL COMPLETION SECTION
C (ENTAILING SECOND AND THIRD PASSES THROUGH DATA MATRIX BY SUBROUTINE CALCD)
C BEGINNING OF FAILURE SALVAGE SECTION
  1 DO 2 OBS=1,NOBS
    A(OBS)=0
  2 B(OBS)=0
  RETURN
C END OF FAILURE SALVAGE SECTION
END

```

```

SUBROUTINE MAKTRE(NOBS,X,MXVAL,NXVAL,ROOT,XVAL,
+LEFT,RIGHT,MOTHER,BALANC,IFAU)

```

```

C SUBROUTINE MAKTRE USES THE VALUES IN THE ARRAY X TO PROVIDE NODES
C TO CONSTRUCT AN AVL SEARCH TREE WITH NODE VALUES STORED IN XVAL,
C LEFT AND RIGHT DAUGHTER NODE POINTERS STORED IN LEFT AND RIGHT,
C ROOT POINTER STORED IN ROOT,
C MOTHER NODE POINTERS STORED IN MOTHER
C AND BALANCE INDICATORS STORED IN BALANC
C AND USING IFAU AS ERROR CODE
C BEGINNING OF DATA DEFINITION SECTION
  INTEGER NOBS
  REAL X(NOBS)
C X IS AN ARRAY OF SIZE NOBS WHOSE VALUES ARE MADE INTO SEARCH TREE
  INTEGER MXVAL,NXVAL,ROOT
  REAL XVAL(MXVAL)
  INTEGER LEFT(MXVAL),RIGHT(MXVAL),MOTHER(MXVAL),BALANC(MXVAL)
C MXVAL, NXVAL, ROOT, XVAL, LEFT AND RIGHT ARE
C AS DEFINED IN SUBROUTINE CONDIS
C MOTHER IS ARRAY OF POINTERS TO MOTHER NODES FOR INDICES OF TREE MATRIX
C BALANC IS ARRAY OF INDICATORS AS TO WHETHER THE LEFT OR RIGHT SUBTREE
C HAS THE GREATER HEIGHT, FOR INDICES OF TREE MATRIX
  INTEGER IFAU
C IFAU IS ERROR INDICATOR
C AND IS SET TO 0 IF COMPLETION SUCCESSFUL
C 1 IF NOBS LESS THAN 1
C 2 IF MXVAL LESS THAN 1
C 3 IF NUMBER OF X-VALUES GREATER THAN MXVAL
  INTEGER OBS,TP0,TP1,TP2,TP3,TP4,TP5
C OBS IS INDEX FOR ARRAY X
C TP0, TP1, TP2, TP3, TP4 AND TP5 ARE POINTERS TO NODES OF SEARCH TREE
  REAL XCUR
C XCUR IS VALUE OF X(OBS) CURRENTLY BEING PLACED IN SEARCH TREE
C END OF DATA DEFINITION SECTION
C INITIALISE ERROR INDICATOR TO ZERO

```

```

    IFAULT=0
C SCREEN FOR ILL-CONDITIONED PARAMETERS
    IF(NOBS.LT.1)THEN
        IFAULT=1
        NXVAL=0
        ROOT=0
        GOTO 7
    ELSE IF(MXVAL.LT.1)THEN
        IFAULT=2
        NXVAL=0
        ROOT=0
        GOTO 7
    END IF
C CREATE ROOT NODE OF SEARCH TREE
    NXVAL=1
    ROOT=1
    MOTHER(1)=0
    XVAL(1)=X(1)
    LEFT(1)=0
    RIGHT(1)=0
    BALANC(1)=0
C BEGINNING OF X OBSERVATION LOOP
    DO 1 OBS=2,NOBS
        XCUR=X(OBS)
C ENTER SEARCH TREE AT ROOT
        TP1=ROOT
C TEST CURRENT OBSERVATION X-VALUE AGAINST SEARCH TREE NODE X-VALUE
        2 IF(XCUR-XVAL(TP1))3,1,4
C X(OBS) LESS THAN XVAL(TP1)
        3 IF(LEFT(TP1).EQ.0)GOTO 5
C ENTER LEFT SUBTREE
        TP1=LEFT(TP1)
        GOTO 2
C BEGINNING OF SECTION CREATING NEW NODE ON LEFT
        5 NXVAL=NXVAL+1
C LEAVE FOR FAILURE SALVAGE SECTION IF TOO MANY X-VALUES
        IF(NXVAL.GT.MXVAL)THEN
            IFAULT=3
            NXVAL=MXVAL
            GOTO 7
        END IF
        LEFT(TP1)=NXVAL
        MOTHER(NXVAL)=TP1
        XVAL(NXVAL)=XCUR
        LEFT(NXVAL)=0
        RIGHT(NXVAL)=0
        BALANC(NXVAL)=0
C LEAVE FOR NEXT OBSERVATION IF HEIGHT OF SUBTREE NOT INCREASED,
C OTHERWISE BACKTRACK TO ENSURE PRESERVATION OF AVL PROPERTY
        IF(BALANC(TP1).GT.0)THEN
            BALANC(TP1)=0
            GOTO 1
        END IF
        BALANC(TP1)=-1
        TP2=TP1
        TP1=MOTHER(TP1)
        GOTO 8
C END OF SECTION CREATING NEW NODE ON LEFT
C X(OBS) GREATER THAN XVAL(TP1)
        4 IF(RIGHT(TP1).EQ.0)GOTO 6
C ENTER RIGHT SUBTREE
        TP1=RIGHT(TP1)
        GOTO 2
C BEGINNING OF SECTION CREATING NEW NODE ON RIGHT
        6 NXVAL=NXVAL+1

```

```

C LEAVE FOR FAILURE SALVAGE SECTION IF TOO MANY X-VALUES
  IF(NXVAL.GT.MXVAL)THEN
    IFAULT=3
    NXVAL=MXVAL
    GOTO 7
  END IF
  RIGHT(TP1)=NXVAL
  MOTHER(NXVAL)=TP1
  XVAL(NXVAL)=XCUR.
  LEFT(NXVAL)=0
  RIGHT(NXVAL)=0
  BALANC(NXVAL)=0
C LEAVE FOR NEXT OBSERVATION IF HEIGHT OF SUBTREE NOT INCREASED,
C OTHERWISE BACKTRACK TO ENSURE PRESERVATION OF AVL PROPERTY
  IF(BALANC(TP1).LT.0)THEN
    BALANC(TP1)=0
    GOTO 1
  END IF
  BALANC(TP1)=1
  TP2=TP1
  TP1=MOTHER(TP1)
C END OF SECTION CREATING NEW NODE ON RIGHT
C BEGINNING OF TREE BACKTRACKING AND BALANCING SECTION
  8 IF(TP1.EQ.0)GOTO 1
  IF(TP2.EQ.RIGHT(TP1))GOTO 9
C BEGINNING OF SECTION FOR WHEN NODE TP1 IS ENTERED FROM LEFT SUBTREE
  IF(BALANC(TP1))12,10,11
  10 BALANC(TP1)=-1
  TP2=TP1
  TP1=MOTHER(TP1)
  GOTO 8
  11 BALANC(TP1)=0
  GOTO 1
C BEGINNING OF SECTION FOR RESTORING AVL PROPERTY WHEN LEFT SUBTREE
C IS TOO HIGH
  12 IF(BALANC(TP2).LT.0)THEN
C LEFT SUBTREE OF LEFT DAUGHTER TOO HIGH
  TPO=MOTHER(TP1)
  TP3=RIGHT(TP2)
  IF(TPO.EQ.0)THEN
    ROOT=TP2
  ELSE IF(TP1.EQ.LEFT(TPO))THEN
    LEFT(TPO)=TP2
  ELSE
    RIGHT(TPO)=TP2
  END IF
  MOTHER(TP2)=TPO
  RIGHT(TP2)=TP1
  MOTHER(TP1)=TP2
  LEFT(TP1)=TP3
  IF(TP3.GT.0)MOTHER(TP3)=TP1
  BALANC(TP1)=0
  BALANC(TP2)=0
  ELSE
C RIGHT SUBTREE OF LEFT DAUGHTER TOO HIGH
  TPO=MOTHER(TP1)
  TP3=RIGHT(TP2)
  TP4=RIGHT(TP3)
  TP5=LEFT(TP3)
  IF(TPO.EQ.0)THEN
    ROOT=TP3
  ELSE IF(TP1.EQ.LEFT(TPO))THEN
    LEFT(TPO)=TP3
  ELSE
    RIGHT(TPO)=TP3

```

```

    END IF
    MOTHER(TP3)=TP0
    LEFT(TP3)=TP2
    MOTHER(TP2)=TP3
    RIGHT(TP3)=TP1
    MOTHER(TP1)=TP3
    LEFT(TP1)=TP4
    IF(TP4.GT.0)MOTHER(TP4)=TP1
    RIGHT(TP2)=TP5
    IF(TP5.GT.0)MOTHER(TP5)=TP2
    IF(BALANC(TP3).LT.0)THEN
        BALANC(TP2)=0
        BALANC(TP1)=1
    ELSE
        BALANC(TP1)=0
        BALANC(TP2)=-1
    END IF
    BALANC(TP3)=0
END IF
GOTO 1
C END OF SECTION FOR RESTORING AVL PROPERTY WHEN LEFT SUBTREE
C IS TOO HIGH
C END OF SECTION FOR WHEN NODE TP1 IS ENTERED FROM LEFT SUBTREE
C BEGINNING OF SECTION FOR WHEN NODE TP1 IS ENTERED FROM RIGHT SUBTREE
    9 IF(BALANC(TP1))14,13,15
    13 BALANC(TP1)=1
        TP2=TP1
        TP1=MOTHER(TP1)
        GOTO 8
    14 BALANC(TP1)=0
        GOTO 1
C BEGINNING OF SECTION FOR RESTORING AVL PROPERTY WHEN RIGHT SUBTREE
C IS TOO HIGH
    15 IF(BALANC(TP2).GT.0)THEN
C RIGHT SUBTREE OF RIGHT DAUGHTER TOO HIGH
        TPO=MOTHER(TP1)
        TP3=LEFT(TP2)
        IF(TPO.EQ.0)THEN
            ROOT=TP2
        ELSE IF(TP1.EQ.RIGHT(TPO))THEN
            RIGHT(TPO)=TP2
        ELSE
            LEFT(TPO)=TP2
        END IF
        MOTHER(TP2)=TPO
        LEFT(TP2)=TP1
        MOTHER(TP1)=TP2
        RIGHT(TP1)=TP3
        IF(TP3.GT.0)MOTHER(TP3)=TP1
        BALANC(TP1)=0
        BALANC(TP2)=0
    ELSE
C LEFT SUBTREE OF RIGHT DAUGHTER TOO HIGH
        TPO=MOTHER(TP1)
        TP3=LEFT(TP2)
        TP4=LEFT(TP3)
        TP5=RIGHT(TP3)
        IF(TPO.EQ.0)THEN
            ROOT=TP3
        ELSE IF(TP1.EQ.RIGHT(TPO))THEN
            RIGHT(TPO)=TP3
        ELSE
            LEFT(TPO)=TP3
        END IF
        MOTHER(TP3)=TPO

```

```

LEFT(TP3)=TP1
MOTHER(TP1)=TP3
RIGHT(TP3)=TP2
MOTHER(TP2)=TP3
RIGHT(TP1)=TP4
IF(TP4.GT.0)MOTHER(TP4)=TP1
LEFT(TP2)=TP5
IF(TP5.GT.0)MOTHER(TP5)=TP2
IF(BALANC(TP3).GT.0)THEN
  BALANC(TP2)=0
  BALANC(TP1)=-1
ELSE
  BALANC(TP1)=0
  BALANC(TP2)=1
END IF
BALANC(TP3)=0
END IF

```

```

C END OF SECTION FOR RESTORING AVL PROPERTY WHEN RIGHT SUBTREE
C IS TOO HIGH
C END OF SECTION FOR WHEN NODE TP1 IS ENTERED FROM RIGHT SUBTREE
C END OF TREE BACKTRACKING AND BALANCING SECTION
C GO TO NEXT VALUE OF X(OBS) IF POSSIBLE, OTHERWISE RETURN SUCCESSFULLY
  1 CONTINUE
C END OF X OBSERVATION LOOP
C RETURN SUCCESSFULLY
  RETURN
C BEGINNING OF FAILURE SALVAGE SECTION
  7 RETURN
C END OF FAILURE SALVAGE SECTION
  END

```

```

SUBROUTINE CALCD(NOBS,X,Y,FREQ,A,B,NXVAL,ROOT,XVAL,
+LEFT,RIGHT,NEQ,NLS,NRS)

```

```

C SUBROUTINE CALCD TAKES, AS INPUT, A DATA MATRIX OF ARRAYS
C WITH VARIABLES X, Y AND FREQ, SORTED BY Y,
C AND A SEARCH TREE OF POSSIBLE X-VALUES STORED IN XVAL
C WITH POINTERS TO LEFT AND RIGHT DAUGHTERS IN LEFT AND RIGHT,
C AND GIVES, AS OUTPUT, THE CONCORDANCE AND DISCORDANCE COUNTS OF X AND Y
C IN ARRAYS A AND B, RESPECTIVELY
C USING ARRAYS NEQ, NLS AND NRS AS SEARCH TREE WORKSPACE
C BEGINNING OF DATA DEFINITION SECTION
  INTEGER NOBS
  REAL X(NOBS),Y(NOBS)
  INTEGER FREQ(NOBS),A(NOBS),B(NOBS)
C NOBS, X, Y, FREQ, A AND B ARE AS DEFINED IN SUBROUTINE CONDIS
  INTEGER NXVAL,ROOT
  REAL XVAL(NXVAL)
  INTEGER LEFT(NXVAL),RIGHT(NXVAL),NEQ(NXVAL),NLS(NXVAL),NRS(NXVAL)
C NXVAL IS NUMBER OF DISTINCT X-VALUES FORMING NODES OF SEARCH TREE
C ROOT IS ROOT NODE OF SEARCH TREE
C XVAL IS ARRAY OF X-VALUES IN SEARCH TREE
C LEFT AND RIGHT ARE ARRAYS OF POINTERS FOR EACH X-VALUE NODE
C POINTING TO LEFT AND RIGHT DAUGHTER NODES, RESPECTIVELY
C NEQ(TREPTR) CONTAINS FREQUENCY OF X-VALUE XVAL(TREPTR)
C NLS(TREPTR) AND NRS(TREPTR) CONTAIN SUMS OF FREQUENCIES OF X-VALUES
C IN THE LEFT AND RIGHT SUBTREES, RESPECTIVELY, OF SEARCH TREE NODE TREPTR
  REAL XCUR,YCUR
  INTEGER FRCUR,OBS,MINOCY,MAXOCY,TREPTR,LSUBT,RSUBT
  LOGICAL ENDDAT
C XCUR AND YCUR STORE VALUES OF CURRENT INTEREST FROM X AND Y, RESPECTIVELY
C FRCUR STORES A VALUE OF CURRENT INTEREST FROM FREQ
C OBS STORES AN INDEX OF THE DATA MATRIX
C MINOCY AND MAXOCY STORE MINIMUM AND MAXIMUM INDICES, RESPECTIVELY,
C WITH A GIVEN Y-VALUE
C TREPTR STORES A NODE OF THE SEARCH TREE

```

```

C LSUBT AND RSUBT STORE LEFT AND RIGHT SUBTOTALS, RESPECTIVELY
C ENDDAT IS INDICATOR THAT A PASS THROUGH THE DATA MATRIX HAS REACHED THE END
C END OF DATA DEFINITION SECTION
C INITIALISE TREE TO REPRESENT EMPTY SET OF OBSERVATIONS
  DO 1 TREPTR=1,NXVAL
    NEQ(TREPTR)=0
    NLS(TREPTR)=0
    NRS(TREPTR)=0
  1 CONTINUE
C INITIALISE ENDDAT AND MAXOCY FOR FIRST Y-VALUE LOOP
C (VISITING Y-VALUES FROM THE LOWEST TO THE HIGHEST)
  ENDDAT=NOBS.LT.1
  MAXOCY=0
C BEGINNING OF FIRST Y-VALUE LOOP
C SET YCUR TO NEW CURRENT Y-VALUE
  2 IF(ENDDAT)GOTO 30
  MINOCY=MAXOCY+1
  YCUR=Y(MINOCY)
C COMPUTE RANGE OF OBSERVATIONS WITH CURRENT Y-VALUE
  MAXOCY=MINOCY
  3 MAXOCY=MAXOCY+1
  IF(MAXOCY.GT.NOBS)GOTO 4
  IF(Y(MAXOCY).NE.YCUR)GOTO 5
  GOTO 3
  4 ENDDAT=.TRUE.
  5 MAXOCY=MAXOCY-1
C BEGINNING OF FIRST SUBTOTAL EXTRACTION LOOP
  DO 6 OBS=MINOCY,MAXOCY
    XCUR=X(OBS)
    LSUBT=0
    RSUBT=0
    TREPTR=ROOT
  7 IF(XCUR-XVAL(TREPTR))8,9,10
  8 RSUBT=RSUBT+NEQ(TREPTR)+NRS(TREPTR)
    TREPTR=LEFT(TREPTR)
    GOTO 7
  10 LSUBT=LSUBT+NEQ(TREPTR)+NLS(TREPTR)
    TREPTR=RIGHT(TREPTR)
    GOTO 7
  9 LSUBT=LSUBT+NLS(TREPTR)
    RSUBT=RSUBT+NRS(TREPTR)
    A(OBS)=LSUBT
    B(OBS)=RSUBT
  6 CONTINUE
C END OF FIRST SUBTOTAL EXTRACTION LOOP
C BEGINNING OF TREE UPDATE LOOP
  DO 11 OBS=MINOCY,MAXOCY
    XCUR=X(OBS)
    FRCUR=FREQ(OBS)
    TREPTR=ROOT
  12 IF(XCUR-XVAL(TREPTR))13,14,15
  13 NLS(TREPTR)=NLS(TREPTR)+FRCUR
    TREPTR=LEFT(TREPTR)
    GOTO 12
  15 NRS(TREPTR)=NRS(TREPTR)+FRCUR
    TREPTR=RIGHT(TREPTR)
    GOTO 12
  14 NEQ(TREPTR)=NEQ(TREPTR)+FRCUR
  11 CONTINUE
C END OF TREE UPDATE LOOP
  GOTO 2
C END OF FIRST Y-VALUE LOOP
C INITIALISE ENDDAT AND MAXOCY FOR SECOND Y-VALUE LOOP
C (VISITING Y-VALUES FROM THE LOWEST TO THE HIGHEST)
  30 ENDDAT=NOBS.LT.1

```

```

    MAXOCY=0
C BEGINNING OF SECOND Y-VALUE LOOP
C SET YCUR TO NEW CURRENT Y-VALUE
    16 IF(ENDDAT)GOTO 31
    MINOCY=MAXOCY+1
    YCUR=Y(MINOCY)
C COMPUTE RANGE OF OBSERVATIONS WITH CURRENT Y-VALUE
    MAXOCY=MINOCY
    17 MAXOCY=MAXOCY+1
    IF(MAXOCY.GT.NOBS)GOTO 18
    IF(Y(MAXOCY).NE.YCUR)GOTO 19
    GOTO 17
    18 ENDDAT=.TRUE.
    19 MAXOCY=MAXOCY-1
C BEGINNING OF TREE DOWNDATE LOOP
    DO 20 OBS=MINOCY,MAXOCY
    XCUR=X(OBS)
    FRCUR=FREQ(OBS)
    TREPTR=ROOT
    21 IF(XCUR-XVAL(TREPTR))22,23,24
    22 NLS(TREPTR)=NLS(TREPTR)-FRCUR
    TREPTR=LEFT(TREPTR)
    GOTO 21
    24 NRS(TREPTR)=NRS(TREPTR)-FRCUR
    TREPTR=RIGHT(TREPTR)
    GOTO 21
    23 NEQ(TREPTR)=NEQ(TREPTR)-FRCUR
    20 CONTINUE
C END OF TREE DOWNDATE LOOP
C BEGINNING OF SECOND SUBTOTAL EXTRACTION LOOP
    DO 25 OBS=MINOCY,MAXOCY
    XCUR=X(OBS)
    LSUBT=0
    RSUBT=0
    TREPTR=ROOT
    26 IF(XCUR-XVAL(TREPTR))27,28,29
    27 RSUBT=RSUBT+NEQ(TREPTR)+NRS(TREPTR)
    TREPTR=LEFT(TREPTR)
    GOTO 26
    29 LSUBT=LSUBT+NEQ(TREPTR)+NLS(TREPTR)
    TREPTR=RIGHT(TREPTR)
    GOTO 26
    28 LSUBT=LSUBT+NLS(TREPTR)
    RSUBT=RSUBT+NRS(TREPTR)
    A(OBS)=A(OBS)+RSUBT
    B(OBS)=B(OBS)+LSUBT
    25 CONTINUE
C END OF SECOND SUBTOTAL EXTRACTION LOOP
    GOTO 16
C END OF SECOND Y-VALUE LOOP
    31 RETURN

```



## PROGRAM CORDAN

```

C PROGRAM CORDAN TAKES AS INPUT A DATA MATRIX,
C STORED IN A BCD INPUT FILE ASSIGNED TO CHANNEL 1,
C AND INPUT USING FORMAT STORED IN INFMT,
C WITH VARIABLES BY, ID, X, Y AND FREQ,
C SORTED BY Y WITHIN CONTIGUOUS GROUPS OF ROWS WITH SAME VALUE OF BY,
C AND GIVES AS OUTPUT THE SAME DATA MATRIX,
C STORED IN A BCD OUTPUT FILE ASSIGNED TO CHANNEL 2,
C AND OUTPUT USING FORMAT STORED IN OUTFMT,
C STILL SORTED BY Y WITHIN CONTIGUOUS GROUPS OF ROWS WITH SAME VALUE OF BY,
C AND AUGMENTED WITH VARIABLES A AND B
C (CONCORDANCE AND DISCORDANCE TOTALS BETWEEN X AND Y, WEIGHTED BY FREQ,
C WITHIN CONTIGUOUS GROUPS OF ROWS WITH SAME VALUE OF BY),
C USING SUBROUTINE CONDIS TO CARRY OUT THE CALCULATIONS
C BEGINNING OF DATA DEFINITION SECTION
      INTEGER MOBS
C MOBS IS MAXIMUM LENGTH OF ANY DATA ARRAY
      INTEGER NOBS
C NOBS IS NUMBER OF OBSERVATIONS IN CURRENT BY GROUP
      INTEGER MXVAL
C MXVAL IS MAXIMUM NUMBER OF DISTINCT X-VALUES IN A BY GROUP
      INTEGER NIWORK
C NIWORK IS DIMENSION OF INTEGER WORK SPACE FOR USE IN SEARCH TREE
      INTEGER BY(10000),ID(10000)
      REAL X(10000),Y(10000)
      INTEGER FREQ(10000),A(10000),B(10000)
      REAL XVAL(10000)
      INTEGER IWORK(50000)
C BY IS ARRAY OF BY-GROUP VALUES (ALL SAME WITHIN A BY GROUP)
C ID IS ARRAY OF OBSERVATION IDENTIFIERS (FOR SUBSEQUENT SORTING)
C X AND Y ARE ARRAYS OF X- AND Y- VALUES FOR DATA MATRIX
C FREQ(I) IS NUMBER OF OBSERVATIONS IN ORIGINAL DATA MATRIX
C REPRESENTED BY THE I'TH OBSERVATION IN CURRENT DATA MATRIX
C A AND B ARE CONCORDANCE AND DISCORDANCE VARIABLES ADDED TO DATA MATRIX
C XVAL IS ARRAY OF POSSIBLE X-VALUES FOR USE IN SEARCH TREE
C IWORK IS INTEGER WORK SPACE FOR USE IN SEARCH TREE
      INTEGER BYCUR,BYIN,IDIN
      REAL XIN,YIN
      INTEGER FREQIN
C BYCUR IS VALUE OF BY FOR BY GROUP CURRENTLY BEING PROCESSED
C BYIN, IDIN, XIN, YIN AND FREQIN ARE USED FOR STORAGE
C OF A NEWLY INPUT OBSERVATION OF THE DATA MATRIX
      INTEGER OBS
C OBS IS INDEX OF CURRENT OBSERVATION OF DATA MATRIX
      LOGICAL ENDINF
C ENDINF IS INDICATOR THAT END OF INPUT FILE HAS BEEN REACHED
      INTEGER IFAULT
C IFAULT IS INDICATOR THAT A SUBROUTINE HAS FAILED
C (RETURNED AS ZERO IF SUBROUTINE COMPLETES SUCCESSFULLY)
      CHARACTER*48 INFMT,OUTFMT
C INFMT AND OUTFMT ARE INPUT AND OUTPUT FORMATS
      DATA MOBS,MXVAL,NIWORK/10000,10000,50000/
      DATA INFMT/'(2I10,2F10.4,I10)'/
      DATA OUTFMT/'(2I10,2F10.4,3I10)'/
C END OF DATA DEFINITION SECTION
C BEGINNING OF FILE INITIALISATION SECTION
      OPEN(UNIT=1,FILE='FOR001',STATUS='OLD',
+READONLY)
      OPEN(UNIT=2,FILE='FOR002',STATUS='NEW',
+CARRIAGECONTROL='LIST')
      READ(1,INFMT,END=18,ERR=11)BYIN,IDIN,XIN,YIN,FREQIN
      ENDINF=.FALSE.
      GOTO 16
18 ENDINF=.TRUE.
      GOTO 10

```

```
C END OF FILE INITIALISATION SECTION
C BEGINNING OF BY GROUP LOOP
  16 BYCUR=BYIN
     NOBS=1
C BEGINNING OF INPUT SECTION
  13 BY(NOBS)=BYIN
     ID(NOBS)=IDIN
     X(NOBS)=XIN
     Y(NOBS)=YIN
     FREQ(NOBS)=FREQIN
     READ(1,INFMT,END=17,ERR=11)BYIN, IDIN,XIN,YIN,FREQIN
     IF(BYIN.NE.BYCUR)GOTO 14
     NOBS=NOBS+1
     IF(NOBS.GT.MOBS)GOTO 19
     GOTO 13
  19 WRITE(6,20)BYCUR,MOBS
  20 FORMAT(1X,'WARNING - MORE OBSERVATIONS IN BY GROUP ',
    +18,' THAN THE MAXIMUM (' ,I4,')')
     GOTO 14
  17 ENDINF=.TRUE.
C END OF INPUT SECTION
C CALL SUBROUTINE CONDIS, REPORTING IN CASE OF FAILURE
  14 CALL CONDIS(NOBS,X,Y,FREQ,A,B,MXVAL,XVAL,NIWORK,IWORK,IFAU
    LT)
     IF(IFAU
    LT.GT.0)WRITE(6,21)BYCUR,IFAU
    LT
  21 FORMAT(1X,'WARNING - SUBROUTINE CONDIS HAS FAILED'
    +/1X,'FAILURE OCCURRED IN BY GROUP ',I8,', FAULT CODE ',I4)
C BEGINNING OF OUTPUT SECTION
  DO 15 OBS=1,NOBS
  15 WRITE(2,OUTFMT)BY(OBS), ID(OBS),X(OBS),Y(OBS),FREQ(OBS),
    +A(OBS),B(OBS)
C END OF OUTPUT SECTION
  IF(ENDINF)GOTO 10
  GOTO 16
C END OF BY GROUP LOOP
C BEGINNING OF TERMINATION SECTION
  11 WRITE(6,12)BYCUR
  12 FORMAT(1X,'BAD DATA ENCOUNTERED IN BY GROUP ',I8)
  10 CLOSE(UNIT=1)
     CLOSE(UNIT=2)
     CALL EXIT
C END OF TERMINATION SECTION
  END
```

Notes for assistance of referees

The enclosed disk contains several files, including the algorithm, the driver program, a test data file, an expected output file for the test data, and several others which, it was thought, might be of assistance to referees by providing some background information on the use and testing of the algorithm by the author. This testing was done on a VAX 3600 computer under the VMS operating system, and extensive use was made of the statistical package SAS to carry out "trivial" operations on data input to, and output from, the algorithm.

The most important files on the disk are as follows:

CONDIS.FOR - The algorithm (in machine-readable form).

CORDAN0.FOR - The driver program (in machine-readable form).

TEST1.DAT - A BCD file of test data for input by the driver program.

TEST2.DAT - A BCD file identical to that which should be output by the driver program when TEST1.DAT is the input file.

The driver program inputs from the input file (channel 1) a sequence of data matrices whose columns are the INTEGER arrays BY and ID, the REAL arrays X and Y and the INTEGER array FREQ, inputting each row with a format stored in INFMT, which is set to (2I10,2F10.4,I10). The array BY is a matrix identifier, so that a set of contiguous rows of input data with the same value of BY is treated as a single data matrix. The array ID is a row identifier, enabling the resorting of the data matrix after processing. (The driver program assumes each input data matrix to be sorted by nondescending value of Y.) The arrays X, Y and FREQ correspond to the parameters of the same names for the subroutine CONDIS, as described in the structure section of the introductory text. The output data matrices are output to the output file (channel 2), and are the input data matrices augmented on the right by the INTEGER arrays A and B, which contain the concordance and discordance counts described in the introductory text, corresponding to the parameters of the same names passed to subroutine CONDIS. The output format is stored in OUTFMT, which is set to (2I10,2F10.4,3I10). Diagnostic comments are output to channel 6, the standard output. The driver program was written to run under VMS, but, if it is adapted for use under other systems, it will probably require little modification apart from the file OPEN and CLOSE statements.

The data file TEST1.DAT contains 3 data matrices. The first is the set of test data of Table 2 in the introductory text. The second is an "expanded" version of these test data, where each I'th row in the data matrix of Table 2 is replaced by FREQ(I) rows, each with the value of FREQ set to 1. The third is from H. E. Daniels and M. G. Kendall (1947), *Biometrika*, 34, 197-208, and is the example elucidated in Table 2 of that paper (which, it should be noted, contains an error). The Y-values in this data matrix correspond to column M of Table 1 of Daniels and Kendall, and the X-values correspond to Column A of that Table. Each of the 3 data matrices is already sorted in nondescending order of Y, as the program in CORDAN0.FOR does not perform this sort. The file TEST2.DAT contains the same data as TEST1.DAT, augmented on the right by concordance and discordance counts. If the algorithm and driver program are working correctly, then the output produced when the input file is TEST1.DAT should be identical to TEST2.DAT.

There are several other files which, although not essential to implementing the algorithm as described in the introductory text, provide some background information on the testing of the algorithm. In particular, the FORTRAN programs mentioned in the section on CPU time are all present on the disk. The files containing these programs are as follows:

HEAPY.FOR - A subroutine HEAPY, taking as input/output a data matrix whose columns are the INTEGER array ID, the REAL arrays X and Y, and the INTEGER array FREQ, and using Heapsort to sort the rows of the matrix in a nondescending order of Y.

HEAPID.FOR - A subroutine HEAPID, taking as input/output a data matrix whose columns are the INTEGER array ID, the REAL arrays X and Y, and the INTEGER arrays FREQ, A and B, and using Heapsort to sort the rows of the matrix in a nondescending order of ID.

CDTRIV.FOR - The subroutine CDTRIV mentioned in the section on CPU time, and performing the same function as the subroutine CONDIS and its servants, but performing it "trivially" by counting the individual concordances and discordances.

CORDAN1.FOR - A driver program calling CONDIS, essentially the same as the one in CORDAN0.FOR except that it inputs its data matrices from, and outputs its augmented matrices to, unformatted binary files of a kind used under VMS. (This was done to reduce input/output CPU time to a minimum.) This program, with the necessary subroutines, is the one whose CPU times appear in Table 1 and Fig. 2 as "Tree, no sort."

CORDAN2.FOR - A driver program, as the one in CORDAN1.FOR except that it calls subroutine HEAPY before calling CONDIS, and calls subroutine HEAPID after calling CONDIS, so that it can input data matrices sorted by ascending ID and output augmented data matrices also sorted by ascending ID. This program, with the necessary subroutines, is the one whose CPU times appear in Table 1 and Fig. 2 as "Tree, with sort."

CORDAN3.FOR - A driver program, as the one in CORDAN1.FOR except that it calls subroutine CDTRIV instead of CONDIS, and does not have the workspace arrays required for passing to CONDIS. This program, with the necessary subroutine CDTRIV, is the one whose CPU times appear in Table 1 and Fig. 2 as "Trivial."

INOUT.FOR - A driver program, as the one in CORDAN3.FOR except that there is a CONTINUE statement instead of the call to a concordance-discordance subroutine, so that the program only inputs the data matrix and outputs the same data matrix augmented with "empty" concordance and discordance counts. This program is the one whose CPU times appear in Table 1 and Fig. 2 as "I/O only."

These FORTRAN programs were tested for time performance by running VMS batch jobs which assigned input channels 1 and 2 and then ran the appropriate program. The CPU times in Table 1 and Fig. 2 are the total CPU times for those batch jobs.

The input files for these batch jobs, and also the command files in which these jobs were submitted, were constructed using the SAS package under VMS. The bivariate pseudorandom data for the input files were derived from a SAS data set TIME1 of 100,000 observations, containing pseudorandom normal variables X and Y. Subsets of these observations were written to unformatted binary data files, suitable for input to the FORTRAN programs. The essential SAS command files used for these processes are present on the disk, and were executed from interactive SAS sessions by using the SAS %INCLUDE command after having set values for any SAS macro variables used inside the SAS command files. The SAS command files used in the time tests are as follows:

TIME1.SAS - Creates the SAS data set TIME1.

TIME2.SAS - Extracts from SAS data set TIME1 a number of samples equal to a SAS macro variable NSAMS, each containing a number of observations equal to a SAS macro variable SAMNUM, and outputs these samples as data matrices to an unformatted binary data file, and then constructs and submits a batch job in which this data file is input to a program whose name is specified in the SAS macro variable PROG. (This SAS command file was used to test the programs in INOUT.FOR, CORDAN2.FOR and CORDAN3.FOR.)

TIME4.SAS - As TIME2.SAS except that the data matrices are each sorted by nondescending value of Y before being output to an unformatted binary data file. (This SAS command file was used to test the program in CORDAN1.FOR.)

One other SAS command file is also present on the disk, and illustrates the possibilities for the use of the algorithm in conjunction with SAS under VMS. This command file is:

CAVL5.SAS - Given a SAS data set whose name is stored in a SAS macro variable DS, assumed to contain variables BY and ID and to be sorted by these variables, CAVL5.SAS uses the program in CORDAN1.FOR to calculate concordance and discordance counts for two variables whose names are passed in SAS macro variables X and Y, and then uses these counts to calculate the jackknife pseudovalues for Kendall's tau-a (formula (6) in the introduction text) and adds the pseudovalues to the original data set in a variable whose name is passed in a SAS macro variable PSEUD.

The other file included on the disk is CONDIS1.PUB, produced by the EXP word

Notes for assistance of referees - page 3

processor and containing the introduction description, the algorithm, the driver program, these notes for assistance of referees, and a covering letter.

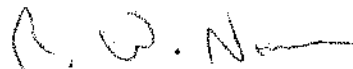
Roger B. Newson,  
Department of Environmental and Preventive Medicine,  
Medical College of St. Bartholomew's Hospital,  
Charterhouse Square,  
London EC1M 6BQ.  
27 November, 1989.

Dear Sir or Madam,

Please find enclosed a description of an algorithm, which I hereby submit to be considered for publication in the Statistical Algorithms section of *Applied Statistics*. The algorithm is intended for use in the interval estimation, by the jackknife method, of concordance-discordance functionals such as Kendall's tau-a and differences between pairs of tau-a functionals. I have implemented this algorithm in FORTRAN under VMS on a VAX computer and under MVS on the Amdahl at the University of London Computing Centre, where I have also implemented a closely related algorithm in VS Pascal.

Please find enclosed, also, a 360K double-sided IBM PC 5.25 inch floppy disk, containing the algorithm (file CONDISE.FOR), the driver program (file CORDAN0.FOR), a file of test input data for the driver program (TEST1.DAT), and the corresponding output file (TEST2.DAT). The disk also contains several other files which might be useful for referees as background information, and details of all of them are given in the accompanying notes headed "Notes for assistance of referees." In particular, I have included on the disk a file, CONDISE1.PUB, produced using the EXP word processor and containing all the human-readable information enclosed here except for the figures, on the off-chance that you are doing your word processing using EXP under MS/DOS on an IBM-compatible PC as I am.

Yours sincerely,



Roger Newson.

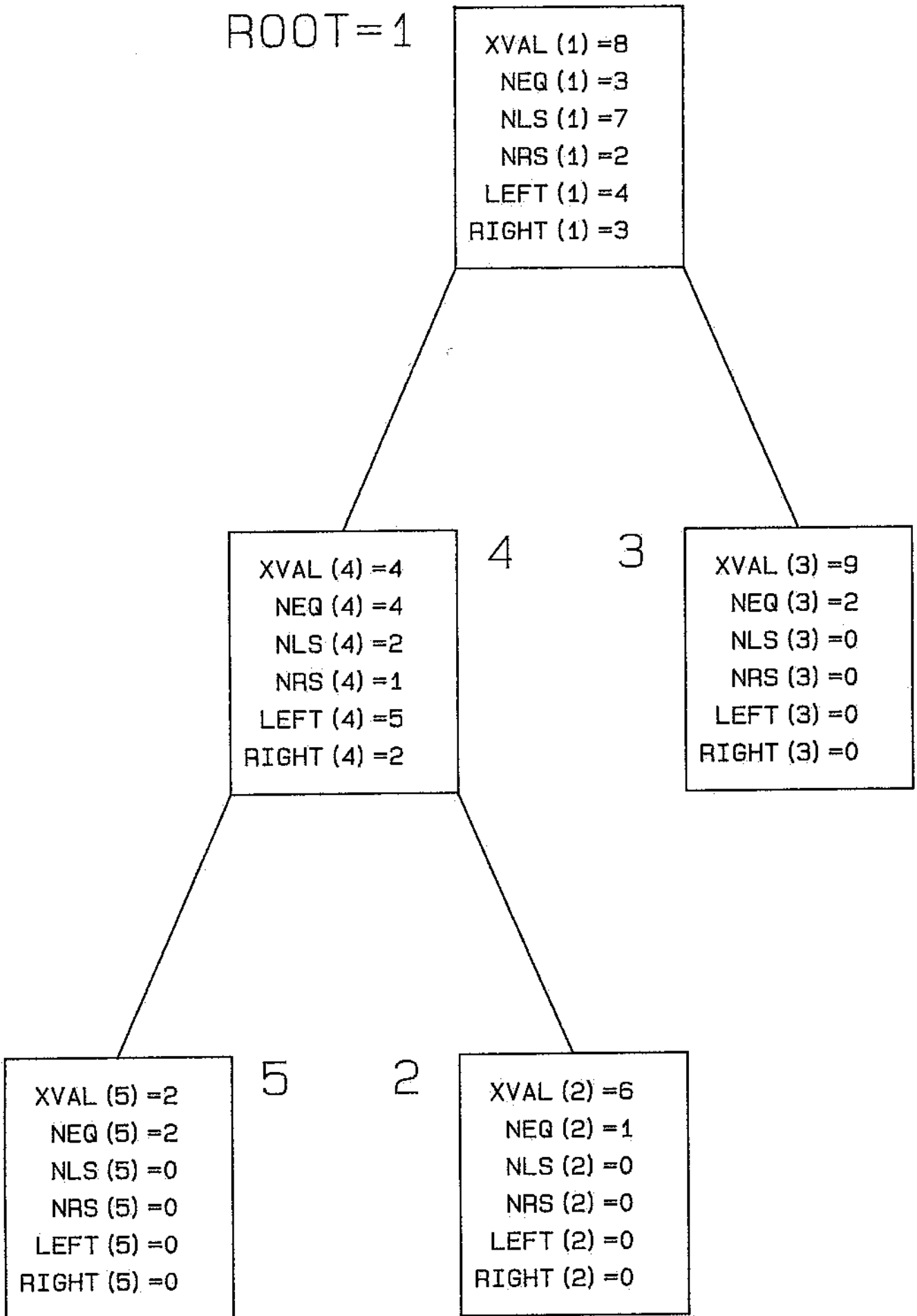
Calculation of concordance and  
discordance counts using an AVL  
search tree

By R. B. NEWSON

Fig. 1. An AVL search tree as used  
by the algorithm

DUI:[350,33]DRAWTREE1.CMD

ROOT=1





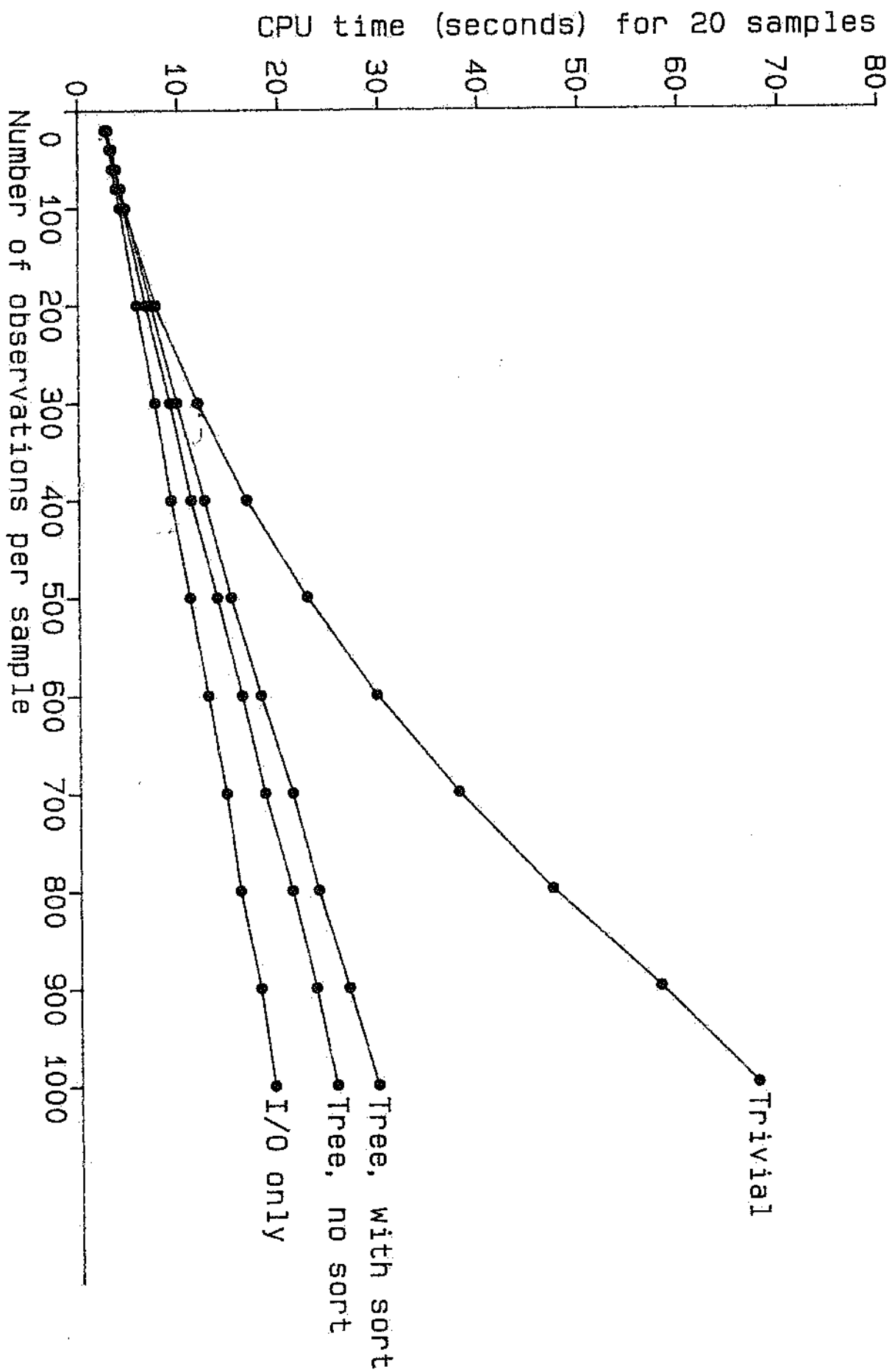
Calculation of concordance and  
discordance counts using an AVL  
search tree.

By R. B. NEWSON

Fig. 2. CPU times for 20 samples as  
a function of sample size

D01: [350,33] TIME1.CMD

\*  
\*  
\*  
\*  
\*





# ROYAL STATISTICAL SOCIETY

Incorporated by Royal Charter, 1887

25 Enford Street,  
London W1H 2BH  
Telephone 01-723 5882  
(International + 441 723 5882)

06 March 1990.

Dr. R.B. Newson,  
Dept. of Environmental & Preventive Medicine,  
Medical College of St. Bartholomew's Hospital,  
Charterhouse Square,  
London.  
EC1M 6BQ.

Dear Dr, Newson,

Alg 735

Your algorithm was classed 'good' or 'very good' by the referee for most attributes of quality of description and programming. The only suggested improvement was a check that input arrays were sorted correctly. However, both he and I are concerned about the applicability of the algorithm. Given current constraints on space available to the algorithms section in Applied Statistics, I am afraid I find it difficult to justify devoting so many pages to an algorithm which comes into its own computationally only when there are more than 1000 observations. Surely there cannot be many readers with such problems?

If I am misinterpreting the situation, please let me know.

Yours sincerely,

A handwritten signature in cursive script that reads 'D. T. Muxworthy'.

David T. Muxworthy.  
Joint Algorithms Editor.