THE t-FIBONACCI NUMBERS AND POLYPHASE SORTING

W. C. LYNCH
Case Institute of Technology, Cleveland, Ohio

1. INTRODUCTION

This paper is divided into two parts that can be read almost independently. The first part defines a generalization of the Fibonacci numbers called the t-Fibonacci numbers, and investigates certain of their properties in detail and without reference to their applications. The second part describes merge sorting and particularly polyphase sorting. A new solution to the initial distribution problem is presented, which constitutes an important application of the theory developed in the first part of this paper.

The reader may start either with the Fibonacci theory in Part 1 or with the sorting application in Part 2. Results from Part 1 are used only in Section 7, the last section of Part 2. The material in Sections 2, 5, and 6 is an exposition of known results in a form designed to help introduce the new results which appear in the other sections.

PART I

2. THE t-FIBONACCI NUMBERS

Any sequence of numbers U_n which satisfies

$$U_n = U_{n-1} + \cdots + U_{n-t} \tag{1}$$

will be called a <u>t-Fibonacci</u> sequence. The <u>i^{th} t-Fibonacci sequence</u> is the special t-Fibonacci sequence with initial conditions $U_j = \delta_{ij}$ for $1 \le j \le t$. The Kronecker delta is represented by δ_{ij}

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

The <u>cumulative t-Fibonacci</u> sequence is the special t-Fibonacci sequence with the initial conditions $U_j = 1$ for $1 \le j \le t$. We will denote the n^{th} term of the i^{th} t-Fibonacci sequence by ${}_iU_n$ and the n^{th} term of the cumulative t-Fibonacci sequence by ${}_\Sigma U_n$. Clearly

$$ {}_{\Sigma}U_n = \sum_{i=1}^{t} {}_iU_n \,. $$

Mention of t in the notation is suppressed since t will remain fixed for any given discussion in this paper. Further, we will restrict t so that $t \geq 2$. Values of ${}_iU_n$ and ${}_{\Sigma}U_n$ for $t = 4$ are given in Fig. 1.

n	${}_1U_n$	${}_2U_n$	${}_3U_n$	${}_4U_n$	${}_{\Sigma}U_n$
0	-1	-1	-1	1	-2
1	1	0	0	0	1
2	0	1	0	0	1
3	0	0	1	0	1
4	0	0	0	1	1
5	1	1	1	1	4
6	1	2	2	2	7
7	2	3	4	4	13
8	4	6	7	8	25
9	8	12	14	15	49
10	15	23	27	29	94
11	29	44	52	56	181
12	56	85	100	108	349
13	108	164	193	208	673

Fig. 1 The 4-Fibonacci Numbers

Many interesting relations can be observed in Fig. 1. We may verify directly that

(2) $$ {}_iU_{t+1} = 1 \,. $$

Another central pair of relations is

(3) $$ {}_1U_n = {}_tU_{n-1} $$

(4) $$ {}_iU_n = {}_{i-1}U_{n-1} + {}_tU_{n-1}, \quad \text{for} \quad 2 \leq i \leq t \,. $$

By (1) and the initial conditions

(5) $$ {}_iU_n = \delta_{in} \qquad 1 \le n \le t $$

we may verify (2) and (3) directly for $2 \le m \le t$. Equation (1a) and the initial conditions allow us to verify (2) and (3) for $n = t + 1$. By summing each side of (2) and (3) over the previous t terms, we can, by induction on n, establish their validity for all n.

For the case $t = 2$, the sequence ${}_1U_n$ and ${}_tU_n = {}_2U_n$ are both the usual Fibonacci sequences. Further, for $t = 2$, ${}_\Sigma U_n = {}_1U_n + {}_2U_n = {}_2U_{n-1} + {}_2U_n = {}_2U_{n+1}$. Thus for $t = 2$, all the sequences are the familiar ones, differing only in the designation of the first element.

3. NUMBER REPRESENTATIONS IN THE t-FIBONACCI NUMBER SYSTEM

We will say a sequence C_j, $j > 1$, is a representation of the integer K in the t-Fibonacci number system if and only if the following conditions are met:

(a) $$ C_j = 0 \quad \text{or} \quad C_j = 1 \quad , $$

(b) $$ K = \sum_{j=1}^{\infty} C_j\,{}_\Sigma U_j \quad , $$

(c) $j \le t$ and $C_j = 0$ implies that $C_i = 0$ for all $i \le j$,

(d) for all $i \ge 0$, if $C_j = 1$ for $i < j < i + t - 1$ then $C_{i+t} = 0$.

This rather technical definition describes a binary positional notation for representing integers. The coefficient digits are restricted to zero and one. The value of the j^{th} position in the notation is ${}_\Sigma U_j$. For example, when $t = 4$ (refer to Fig. 1 again),

$$ 39 = 25 + 13 + 1 $$

or in the positional notation

$$ 39 = 1100|1000 \ . $$

(It is convenient to insert a vertical line t digits from the right.)

By condition (c), $39 = 1100|0100$ would be incorrect, and by condition (d), $39 = 1011|1110$ would be incorrect.

Condition (c) says that the positions of value 1 that we use must be the left-most ones available. Condition (d) says that we may have no more than $t - 1$ ones in a row. It is not immediately clear whether or not all positive integers are representable, or whether such a representation is unique.

We now present the

COUNTING ALGORITHM:

Let C_j be a representation for K.

<u>Step (1)</u> Select the largest j such that $t \geq j \geq 1$ and $C_k = 0$ for all $k \leq j$. Change C_j from 0 to 1.

<u>Step (2)</u> If C_j through C_{j+t-1} are not all 1's then the algorithm terminates.

<u>Step (3)</u> If C_j through C_{j+t-1} are all 1's then change C_{j+t} from 0 to 1 (if C_{j+t} is not 0 then the original sequence has 1's in C_{j+1} through C_{j+t} and thus violated condition (d)). Increase j by t and go back to Step (2).

We observe the following about the counting algorithm when C_j is a representation of K:

1. Step (1) increases by one the value of the representation.

2. Step (3) does not change the value of the representation since U_n

3. Each application of Step (3) reduces the number of 1's in the representation so that the algorithm terminates.

4. At termination, the resulting sequence of C_j's satisfies (a) through (d) and is thus a representation of $K + 1$.

5. If C_{n-1} was the non-zero coefficient of maximum index in the original representation of K, then either

5a. C_{n-1} is still the non-zero coefficient of maximum index in the representation of $K + 1$ or

5b. The resulting representation of $K + 1$ contains the sole non-zero entry C_n $(K + 1 = {}_{\Sigma}U_n)$.

As step 3 was or was not executed with $j = n - t$, 5a or 5b applies.

<u>Lemma 1</u>. Each sequence C_j which satisfies (a) through (d) represents an integer K such that $K < {}_{\Sigma}U_n$, where C_{n-1} is the non-zero coefficient of maximum index.

Proof: Repeatedly apply the Counting Algorithm to C_j. Since an infinite number of integers can be represented, some application of the algorithm (say, the r^{th}) must terminate with condition 5b holding. Then $K + r = {}_{\Sigma}U_n$ so that $K < {}_{\Sigma}U_n$.

We now prove

Theorem 2: (Extended Zeckendorf theorem) Each non-negative integer has a unique representation in the t-Fibonacci number system.

Proof: We will show that the theorem holds for all integers less than U_n. The proof will proceed by induction on n.

Base Step: Take $n = t + 1$. Then ${}_{\Sigma}U_n = t$. For all non-negative integers less than t, representability is assured by having enough positions (t) of value 1 available. Uniqueness follows from condition (d). Condition (d) is satisfied trivially.

Induction Step: Let ${}_{\Sigma}U_{n-1} \leq K < {}_{\Sigma}U_n$ and assume the theorem for all non-negative integers $< {}_{\Sigma}U_{n-1}$. Using (3) as a clue, we observe that

(6) $${}_{\Sigma}U_n = 2{}_{\Sigma}U_{n-1} - {}_{\Sigma}U_{n-t-1} .$$

Since we are on an induction step, n is greater than $t + 1$ so that

$${}_{\Sigma}U_{n-t-1} > 0 .$$

It then follows that

(7a) $${}_{\Sigma}U_n - {}_{\Sigma}U_{n-1} < {}_{\Sigma}U_{n-1} ,$$

and certainly

(7b) $$0 \leq K - {}_{\Sigma}U_{n-1} < {}_{\Sigma}U_{n-1} .$$

Since, by induction, we can represent $K - {}_{\Sigma}U_{n-1}$ (clearly with $C_{n-1} = 0$), we can represent K except for some uncertainty about condition (d). But the only place (d) could be violated would be for C_{n-1} down to C_{n-t} to be all 1's. If that is the case, then $K \geq {}_{\Sigma}U_{n-1} + \cdots + {}_{\Sigma}U_{n-t} = {}_{\Sigma}U_n$, contrary to assumption. Thus K is representable. By Lemma 2, every representation of K must have $C_{n-1} = 1$. Otherwise Lemma 1 asserts that $K < \Sigma U_{n-1}$, contrary to the induction assumption. If K has two representations, say C_j and C'_j, then $K - \Sigma U_{n-1}$ is represented by D_j and D'_j where D_j is obtained from C_j by changing C_{n-1} from 1 to 0. Similarly, D'_j is obtained from C'_j. Both D_j and D'_j are then distinct representations for $K - \Sigma U_{n-1}$. Since by (7b),

$$K - \Sigma U_{n-1} < \Sigma U_{n-1} ,$$

this is impossible by the induction hypothesis. The representation for K is thus unique. Q. E. D.

Theorem 2 for the case $t = 2$ is the familiar Zeckendorf Theorem [2].

In any representation, (d) implies that one of the t bottom positions must be zero. Select the left-most zero position from the bottom t positions and change it to a one. This increases the value of the representation by one and preserves condition (c). It may, however, cause a violation of condition (d). If this is so, the 1 we added must be the right-most in a string of t ones. Zero out that string of ones and change the next highest position (it must be zero or there would already have been a string of t ones contrary to (d)) to a one. Repeat this "carrying" step as often as necessary to assure condition (d). Figure 2 depicts counting in the 4-Fibonacci system.

4. t-FIBONACCI DISTRIBUTIONS

We define the nth t-Fibonacci distribution to be the t dimensional vector $({}_1U_n, {}_2U_n, \cdots, {}_tU_n) = V_n$. Using vector addition in the usual sense (add corresponding components), it is clear that

$$V_n = V_{n-1} + V_{n-2} + \cdots + V_{n-t} . \tag{8}$$

That is, the V_n's satisfy the t-Fibonacci equation. It is clear that the sum of the components of V_n is ΣU_n.

We will say that V is the t-distribution for the integer K if and only if

$$V = \sum_j C_j V_j ,$$

where C_j is the representation for K.

K	Representation	4-Distribution
0	$000 \mid 0000$	(0, 0, 0, 0)
1	$000 \mid 1000$	(0, 0, 0, 1)
2	$000 \mid 1100$	(0, 0, 1, 1)
3	$000 \mid 1110$	(0, 1, 1, 1)
4	$001 \mid 0000 \longleftarrow 000 \mid 1111$	(1, 1, 1, 1)
5	$001 \mid 1000$	(1, 1, 1, 2)
6	$001 \mid 1100$	(1, 1, 2, 2)
7	$010 \mid 0000 \longleftarrow 001 \mid 1110$	(1, 2, 2, 2)
8	$010 \mid 1000$	(1, 2, 2, 3)
9	$010 \mid 1100$	(1, 2, 3, 3)
10	$010 \mid 1110$	(1, 3, 3, 3)
11	$011 \mid 0000 \longleftarrow 010 \mid 1111$	(2, 3, 3, 3)
12	$011 \mid 1000$	(2, 3, 3, 4)
13	$100 \mid 0000 \longleftarrow 011 \mid 1100$	(2, 3, 4, 4)

Fig. 2 Representation and 4-Distribution of $0 \leq K \leq 13$

The Counting Algorithm makes it clear that the t-distribution for $K + 1$ is obtained from the t-distribution for K simply by adding 1 to the j^{th} component where j is the left-most zero position in the bottom t positions of the representation for K. See Fig. 2.

Corresponding to each non-negative integer K, we now have two quantities, the representation of K in the t-Fibonacci number system and the t-distribution for K. Given the (representation) (t-distribution) for K we can calculate the (representation) (t-distribution) for $K + 1$. If K is ${}_{\Sigma}U_n$ for some n then the representation for K has but one 1 bit and the t-distribution has ${}_iU_n$ for its i^{th} component (the t-distribution is a "row" in a tabulation like that in Fig. 1). Given the t-distribution for some ${}_{\Sigma}U_n$ we can easily calculate the t-distribution for ${}_{\Sigma}U_{n+1}$ by means of Eqs. (3) and (4).

We now ask what is the effect of applying Eqs. (3) and (4) to an arbitrary t-distribution. In particular, is the result a t-distribution, and, if so, for which integer? Let X be the t-distribution for K and let the i^{th} component of $f(X)$ be

$$f(X)_i = \begin{cases} X_t & \text{if } i = 1 \\ X_{i-1} + X_t & \text{if } 1 < i \leq t \end{cases}$$

(the subscripts refer to the component positions of the t-dimensional vectors). Notice that f is a linear function on the t-dimensional vector space. Now,

$$X = \sum_{j=1}^{\infty} C_j V_j ,$$

where the C_j's are the coefficients in the representation of K.

$$f(x) = f\left(\sum_{j=1}^{\infty} C_j V_j\right) = \sum_{j=1}^{\infty} C_j f(V_j) = \sum_{j=1}^{\infty} C_j V_{j+1} = \sum_{j=2}^{\infty} C_{j-1} V_j .$$

Let $D_1 = 0$ and $D_{j+1} = C_j$ so we have

$$f(X) = \sum_{j=1}^{\infty} D_j V_j .$$

Since D_j satisfies (a) through (d), f(X) is the t-distribution for the integer represented by the D_j's. In other words, applying f to the t-distribution for K yields the t-distribution for an integer whose representation in the t-Fibonacci number system is the representation of K shifted left one place. Observe, in Fig. 2, the entries for 5 and 11. Applying f to the 4-distribution of 5 yields the 4-distribution of 11. The representation of 5 shifted left one place yields the representation of 11.

Since it is a non-singular linear transformation, f is invertible. Considering g(X) where

$$g(X)_j = \begin{cases} X_1 & \text{if } i = t \\ X_{i+1} - X_1 & \text{if } 1 \leq i < t \end{cases} .$$

The inverse of f is clearly g. Applying g to the t-distribution of the t-distribution of the integer whose representation is the representation of K shifted right one place provided condition (c) is not violated by the right shift operation.

Since f is not onto, the domain of g must be restricted to coincide with the range of f. Condition (c) will not be violated under precisely this condition. Thus the composition gf is the identity function.

PART II

5. SORTING BY MERGING

In data-processing by digital computers, it very often proves to be convenient or essential to arrange large volumes of data into a linear sequence. The unit of data is called a <u>record</u>. Each record has associated with it a number called its <u>key</u>. The process of arranging the records into a sequence (or <u>file</u>) so that the key values are non-decreasing is called <u>sorting</u>.

A utility company will keep its customer file in ascending order by customer number. The incoming utility bill payments will be sorted into ascending order by customer number. The customer file (particularly the customer's balance) can now be adjusted to account for the payment. With both files in order by customer number, this can be accomplished without undue searching through either file.

The sorting of large files is not particularly simple. Usually the files are much too large to be held in the memory of the computer, and they must be recorded on some linear medium such as magnetic tape. The primary method of sorting such external files depends on the technique of merging. If we have two files already sorted into order, we can combine, or merge, these into one file. We do this as follows. Call the two files to be merged A and B and the resulting file C. File C is initially empty. We look at the first records of A and B and select the one with the smallest key. We transfer this record from the input file to C, removing it from the input file. We repeat this process until both A and B are exhausted. File C will now contain the records from A and B and will be in ascending order. See Fig. 3.

We will say that a file in ascending order by its key is a <u>sorted file.</u> The idea behind merge sorting is to begin with many small sorted files; perhaps

A	B	C
2	4	2
3	7	3
5	8	4
6	10	5
9		6
11		7
		8
		9
		10
		11

Fig. 3 Merger of A and B to form C

each contains but one record. By repeated mergers, the average length of the sorted files grows, and, more importantly, the number of files decreases after each merger. Since the number of files cannot increase indefinitely, the process must terminate with one file in sorted order.

We propose to look into the details of this process, particularly a merge sorting technique called polyphase sorting. Let us begin by describing a simpler technique called "balanced symmetric sorting." Suppose initially that we have twenty-seven files arranged on six magnetic tapes as follows:

tapes	1	2	3	4	5	6
files	9	9	9	0	0	0

That is, tapes 1, 2, and 3 contain nine files each (one after the other), while tapes 4, 5, and 6 are empty. By an obvious extension of the described merging process, we can merge together three files at once. Suppose we merge the files that appear first on tapes 1, 2, and 3. This new file is placed on tape 4 so that we have twenty-five files.

1	2	3	4	5	6
8	8	8	1	0	0

By their nature, tapes 1, 2, and 3 are in position to read their second files and tape 4 is positioned to write a file after the one just constructed. Tape 4 is not in a convenient position to re-read the file just written. Files from 1, 2, and 3 are merged again, and the resulting file is written on tape 5. One more merger (the result to tape 6) gives us the twenty-one files.

1	2	3	4	5	6
6	6	6	1	1	1

We make six more mergers, writing the resulting files cyclically on tapes 4, 5, and 6, yielding nine files.

1	2	3	4	5	6
0	0	0	3	3	3

Each file is three times as long as the originals. We rewind all six tapes and make three mergers yielding

1	2	3	4	5	6
1	1	1	0	0	0

We rewind again and one merger gives us

1	2	3	4	5	6
0	0	0	1	0	0

All the records appear on tape 4 as one sorted file. It should be clear that the number of passes (i.e., the number of times each record is processed) is

$$\log_{\frac{(t+1)}{2}} (f)$$

where $t + 1$ is the number of available tapes and f is the number of files. The more tapes, the faster it goes. Figure 4 tabulates a sort of forty-nine files on five tapes. Usually, the number of files and the number of tapes don't come out even as in the example above, and some "fudging" is needed. For a more complete discussion of merge sorting, see [4] and [5].

Tapes	1	2	3	4	5
	16	16	17	0	0
	0	0	0	8	9
	5	6	6	0	0
	0	0	0	3	3
	1	1	1	0	0
	0	0	0	0	1

Fig. 4 A Sort of 49 Files on Five Tapes

6. POLYPHASE SORTING

Let us try to construct a four-way merge process with just five tapes. The natural thing to do is work the problem backwards. Suppose we wish to end with just one file on tape 5.

	tapes				
pass	1	2	3	4	5
	0	0	0	0	1

We must have gotten here by merging four files, one each on tapes 1, 2, 3, and 4.

	tapes				
pass	1	2	3	4	5
5	1	1	1	1	0

one each from 1, 2, 3, and 4 to 5. How did we get here? We could get the one file on tape 4 by a four-way merger of one file each from tapes 1, 2, 3, and 5.

	tapes				
pass	1	2	3	4	5
6	2	2	2	0	1

one each from 1, 2, 3, and 5 to 4. Notice that the sorting process leaves files on tapes 1, 2, and 3 and rewinds 5. The key idea of this process, called polyphase sorting [5] is not to exhaust all of the input files available. The next few steps backward are:

	tapes					
pass	1	2	3	4	5	
7	4	4	0	2	3	2 each from 1, 2, 4, 5 to 3
8	8	0	4	6	7	4 each from 1, 3, 4, 5 to 2
9	0	8	12	14	15	8 each from 2, 3, 4, 5 to 1
10	15	23	27	29	0	15 each from 1, 2, 3, 4 to 5
11	44	52	56	0	29	29 each from 1, 2, 3, 5 to 4

At each step, we four-way merge as many files as are available on the tape with the smallest number of files, then rewind this tape. We leave the residue on the other tapes for subsequent passes.

To simplify the analysis of this process, we can give the tapes "logical" names. We will select the names W and L_1, L_2, L_3, L_4. The tape we will write on is designated as W. The tape with the smallest number of files is L_1 and L_2 is the tape with the next smallest number up to L_4, the tape with the largest number of files. The correspondence of logical names to tape unit numbers was as follows for the previous example.

	tapes				
pass	1	2	3	4	5
4	W	L_1	L_2	L_3	L_4
5	L_1	L_2	L_3	L_4	W
6	L_2	L_3	L_4	W	L_1
7	L_3	L_4	W	L_1	L_2
8	L_4	W	L_1	L_2	L_3
9	W	L_1	L_2	L_3	L_4
10	L_1	L_2	L_3	L_4	W
11	L_2	L_3	L_4	W	L_1

At the end of each pass, the logical labels shift right one with respect to the physical numbers.

It will be much more convenient to organize the tableau containing the number of files per tape per pass by logical names rather than physical tape numbers. The example is given in Fig. 5.

Usually the W column is dropped, since it is always zero. Compare Fig. 5 with Fig. 1 in Part 1.

pass	L_1	L_2	L_3	L_4	W
0	-1	-1	-1	1	0
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	1	1	1	1	0
6	1	2	2	2	0
7	2	3	4	4	0
8	4	6	7	8	0
9	8	12	14	15	0
10	15	23	27	29	0
11	29	44	52	56	0

Fig. 5 A Polyphase Sort of 181 Files

The sorting rule for each pass is clear. We four-way merge n times, where n is the number of files on L_1, n files are written on W, which becomes L_4 in the next pass. We rewind L_1 and it becomes W (since it is empty) on the next pass. Now L_1 has lost n files and becomes L_{i-1} on the next pass. Hence if $\#L_i$ is the number of files on L_i at pass n we have the sorting rule

$$\#_{n-1}L_4 = \#_nL_1$$
$$\#_{n-1}L_{i-1} = \#_nL_1 - \#_nL_1 \qquad 1 < i \leq 4 .$$

Applying these relations, we can fill in the tableau for $0 \leq n < 4$.

To construct the tableau in the opposite direction, it is clear that

$$\#_nL_1 = \#_{n-1}L_4$$
$$\#_nL_i = \#_{n-1}L_{i-1} + \#_{n-1}L_4 \qquad 1 < i \leq t$$

for

$$1 \leq n \leq t \qquad \#_nL_i = \delta_{in} .$$

7. A DISTRIBUTION AND CONTROL ALGORITHM FOR POLYPHASE SORTING

It is at once clear that $\#_n L_i$ is ${}_iU_n$ and that row n of the tableau placed in vector form is the t-distribution for $\sum U_n$. Note that we have $\sum U_n$ files at pass n.

We now raise two related problems. How do we adjust if the number of files to be sorted is not some $\sum U_n$? And, in general, how do we determine the initial distribution of the files over t tapes? The answer to this second question is complicated by the fact that in general, we will not know how many files are to be sorted. The files will initially be contained on some other tape and we will read them one by one and distribute them over the input tapes. We must distribute them in such a manner that at any time we will be prepared to carry out the polyphase algorithm. This is because we won't know which file is last until we come upon it. This distribution problem is discussed in 8 .

The following algorithm is proposed for calculating the initial distribution. We will number the files in sequence and represent the number in the t-Fibonacci number system. As we distribute a file we will obtain its number by counting up by one the number representation of the previous file. This involves inserting a 1 in one of the lower t positions (say position j) of the representation of the number of the previous file and adjusting for carries. Refer back to the Counting Algorithm. The new file to be distributed is then copied to tape j. At each step, then, the distribution of files is the t-distribution for the number of files thus far distributed.

If we distribute 11 files on four tapes, Fig. 2 indicates that the distribution would be (2, 3, 3, 3). If we take one pass on the polyphase sort algorithm we obtain 5 files distributed (1, 1, 1, 2). (Two files each are merged from the four tapes, leaving three tapes with one file and creating a new tape with two files.) We can keep track of this by shifting the representation right one place. The representation for 11 is 011|0000 and the representation for 5 is 01|1000. Another pass of the polyphase algorithm gives us two files with the 4-distribution (0, 0, 1, 1) and the representation 0|1100.

If we try another pass, nothing happens since L_i contains no files. Also, the shift would give the unacceptable representation |0110 (this violates (c)). We instead should add dummy files to tapes L_1 and L_2. Dummy files are files with no records in them. By (d), this promotes the representation to

$1|0000$ and the distribution to (1, 1, 1, 1). One more pass produces the representation $0|1000$ and the distribution (0, 0, 0, 1), and we are done. Actually, this last merger was only two-way, but we pretended it was four-way!

The rule then is that when a shift of the representation to the right is not allowed because a zero would shift into position t, dummy files should be added to the indicated tapes. With this slight adjustment, any number of files may be sorted. Figure 6 gives a blow-by-blow account of a five-tape, four-way merger of nine files.

Operation	Representation	Distribution	Comment
distribute	$000 \mid 1000$	(0, 0, 0, 1)	to tape 4
distribute	$000 \mid 1100$	(0, 0, 1, 1)	to tape 3
"	$000 \mid 1110$	(0, 1, 1, 1)	to tape 2
"	$001 \mid 0000$	(1, 1, 1, 1)	to tape 1
"	$001 \mid 1000$	(1, 1, 1, 2)	to tape 4
"	$001 \mid 1100$	(1, 1, 2, 2)	to tape 3
"	$010 \mid 0000$	(1, 2, 2, 2)	to tape 2
"	$010 \mid 1000$	(1, 2, 2, 3)	to tape 4
"	$010 \mid 1100$	(1, 2, 3, 3)	to tape 3
sort	$001 \mid 1000$	(1, 1, 1, 2)	dummy files added to 1, 2
sort	$000 \mid 1100$	(0, 0, 1, 1)	
sort	$000 \mid 1000$	(0, 0, 0, 1)	dummy files added to 1, 2

Fig. 6

A little thought will produce a slightly more economical method of adding dummy files, but we leave this to the reader's imagination.

8. CONCLUSION

We have presented a generalization of the Fibonacci numbers and developed some of their salient properties. In particular, we proved an extension of Zeckendorf's theorem, and used this to develop the t-Fibonacci positional number system. We investigated the processes of counting and shifting in this number system.

In Part 2, we reviewed the basics of merge sorting and polyphase sorting and went on to use the theory of Part 1 to develop a new initial distribution and merge-control algorithm.

It seems clear that this sort of analysis can be carried out for many other merge sorting schemes (e.g., oscillating [7] or cascade sort [5]). With each sorting scheme, we should be able to associate a number system. This number system should be such that one pass of the merger corresponds to shifting the representation of the number of files right, one place. The initial distribution of files is controlled by the mechanics of counting and imperfect distributions are adjusted to prevent shifting digits off the right end of the count.

ACKNOWLEDGEMENT

The author is indebted to D. E. Knuth for many valuable suggestions.

REFERENCES

1. J. L. Brown, Jr., "Zeckendorf's Theorem and Some Applications," Fibonacci Quarterly, Vol. 2, No. 3, pp. 163-168.
2. W. C. Carter, "Mathematical Analysis of Merge-Sorting Techniques," Proc. IFIP Congress 62, Munich 1962.
3. D. E. Daykin, Journal London Math. Soc., 35 (1960), pp. 143-160.
4. E. H. Friend, "Sorting on Electronic Computer Systems," J. Assoc. for Computing Machinery, Vol. 3 (1956), p. 134.
5. C. C. Gotlieb, "Sorting on Computers," Comm. Association for Computing Machinery, Vol. 6, No. 5, pp. 194-201.
6. R. L. Gilstad, "Polyphase Merge Sorting — An Advanced Technique," Proc. Eastern Joint Computer Conference (1960).
7. D. E. Knuth, CACM 6 (1963), pp. 555-563.
8. W. D. Malcolm, "String Distribution for the Polyphase Sort," Comm. Association for Computing Machinery, Vol. 6, No. 5, pp. 217-220.
9. A. G. Mendoza, CACM 5 (1962), pp. 502-504.
10. E. P. Miles, Jr., American Math. Monthly, 67 (1960), pp. 745-752.
11. V. Schlegel, El Progreso Mat. 4 (1894), pp. 171-174.
12. S. Sobel, "Oscillating Sort-A New Merge Sorting Technique," J. Assoc. for Computing Machinery, Vol. 9, No. 3, pp. 371-374.

★ ★ ★ ★ ★