OPTIMAL POLYPHASE SORTING

by

Derek A. Zave

STAN-CS-76-543
MARCH 1976

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Optimal Polyphase Sorting

by Derek A. Zave

Computer Science Department
Stanford University
Stanford, California 94305

Abstract

A read-forward polyphase merge algorithm is described which performs the polyphase merge starting from an arbitrary string distribution. This algorithm minimizes the volume of information moved. Since this volume is easily computed, it is possible to construct dispersion algorithms which anticipate the merge algorithm. Two such dispersion techniques are described. The first algorithm requires that the number of strings to be dispersed be known in advance; this algorithm is optimal. The second algorithm makes no such requirement, but is not always optimal. In addition, performance estimates are derived and both algorithms are shown to be asymptotically optimal.

Keywords and Phrases: Sorting, tape sorting, merge sorting, polyphase sorting, tape merging, optimal merging, optimal polyphase dispersion, blind dispersion, polyphase dispersion, Fibonacci numbers, generalized Fibonacci numbers, Zeckendorf Theorem, generalized Zeckendorf Theorem.

CR Categories: 5.31, 5.30.

This research was supported in part by National Science Foundation grant MCS 72-03752 A03, by the Office of Naval Research contract NR 044-402, and by IBM Corporation. Reproduction in whole or in part is permitted for any purpose of the United States Government.

1. Introduction.

This paper presents a mathematical analysis of the structure of the polyphase sort with special emphasis on those properties which are related to the performance of the sort. This analysis will enable us to construct a poly-phase sorting algorithm with optimal performance characteristics. We will also construct a near-optimal polyphase sort which is more suitable for applications. Finally, we will investigate the asymptotic performance of both of these algorithms.

Although the polyphase sort has been in use for over a decade, comparatively little work has been done in the direction of optimizing its performance. In an early unpublished paper [7], Sackman and Singer developed methods for predicting the performance of the polyphase merge and showed empirically that in certain cases that the performance of the usual method of implementing the polyphase sort could be greatly improved. Independently, Shell [8] developed similar techniques and used them along with some empirical observations to construct an optimal polyphase sorting algorithm. D. E. Knuth [5] has also investigated the optimal polyphase sort and several of his results have been incorporated into this paper.

2. The Polyphase Merge.

We will begin with a brief discussion of the polyphase merge which will serve primarily to introduce our terminology. Further details, as well as information on internal sorting and string merging, which we will not discuss, may be found in the books of Flores [2] and Knuth [5].

Let us suppose that we are given a collection of records containing various kinds of information and let us further suppose that some linear ordering has been defined on this collection. To <u>sort</u> the records is to arrange them into a sequence which is increasing with respect to the ordering relation. One method of accomplishing this is by means of merging. First, the collection of records is partitioned into a number of small groups of records which each sorted to form a "<u>string</u>" of records. Second, the sorted strings are merged to form larger sorted strings, and so on, until a single sorted string containing all of the records is formed.

In practice, merge sorts are employed when there are more records to be sorted than may be accommodated by a computer% main storage. Groups of records are sorted into strings using the available main storage. The strings are then "dispersed" to some secondary storage medium such as mass storage or magnetic tape. The string merging operations are performed as transfers of information from one part of secondary storage to another.

The poly-phase sort is a merge sort which is characterized by the manner in which the dispersed strings are merged. Let us suppose that there are $T > 3$ tape units which are numbered from zero to $t = T-l$. We define the <u>distribution numbers</u> S_i^n for $i = 1,\ldots,t$ and $n > 1$ by

(2.1)
$$s_i^1 = 1 \qquad \text{for } 1 \le i \le t$$
$$s_1^n = s_t^{n-1} \qquad \text{for } n > 1 \text{, and}$$
$$s_i^n = s_{i-1}^{n-1} + s_t^{n-1} \qquad \text{for } n > 1 \text{ and } 2 \le i \le t \text{ .}$$

From this definition it is easily show that for $n > 1$, we have

$$(2.2) \qquad s_1^n < s_2^n \le \ . \ . \ . \ \le s_t^n \ .$$

Suppose that for some $n \ge 1$ that $s_1^n + \ . \ . \ . + s_t^n$ strings have been *dispersed to the tapes in the following fashion:

tape:	0	1	2	. . .	t
strings:	0	s_1^n	s_2^n	. . .	s_t^n .

We will call this configuration the perfect stage n distribution and the sum

$$(2.3) \qquad s^n = s_1^n + \ . \ . \ . + s_t^n$$

will be called the stage n perfect number.

Example 2.1. The following table provides some values of the distribution numbers and perfect numbers when T = 5 (t = 4) :

n	s_1^n	s_2^n	s_3^n	s_4^n	s^n
1	1	1	1	1	4
2	1	2	2	2	7
3	2	3	4	4	13
4	4	6	7	8	25
5	8	12	14	15	49
6	15	23	27	29	94
7	29	44	52	56	181
8	56	85	100	108	349
9	108	164	193	208	673
10	208	316	372	401	1297

Suppose that we start with the perfect stage n distribution'. If we merge together one string from each of the tapes $1,\ldots,t$, then we will obtain a single string which may be written to unit zero. If $n = 1$, then this operation will merge all of the strings since each tape contains exactly one string. If $n > 1$, then, in view of (2.2), we may perform this operation S_1^n times after which we will arrive at the distribution.

tape:	0	1	2	...	t
strings:	S_1^n	0	$S_2^n - S_1^n$	...	$S_t^n - S_1^n$

From the formulas (2.1) we see that this distribution is the same as

tape:	0	1	2	. . .	t
strings:	S_t^{n-1}	0	S_1^{n-1}	...	S_{t-1}^{n-1}

so that if we renumber the tapes $t,0,1,2,\ldots,t-1$, then we obtain the perfect stage n-l distribution.

By repeating this process, we obtain the perfect distributions for stages n-2, n-3 , and so on, until we arrive at the perfect distribution for stage one. A single merge will then produce the final sorted string. This method of merging a perfect number of strings is called the <u>polyphase merge.</u>

In practice, the distribution routine rarely produces a perfect number of strings. In order to use the polyphase merge in this case it is necessary to include a number of "<u>dummy</u>" (empty) strings in order to fill out the-total number of strings to a perfect number. There are therefore two choices which have to be made before using the polyphase merge to sort x strings. First we must choose a starting stage number n ; any n for which $x < S^n$ is eligible. Second, we must decide how the $S^n - x$ dummy strings are to be distributed among the x strings. Although many methods have been proposed for distributing the dummy strings, most authors recommend starting with the smallest possible stage number n; we will refer to these approaches collectively as the standard polyphase sort.

Since the speed of a merge is usually limited by the transfer rate of the tape units and the speed of the merge algorithm, we see that the time required to perform the polyphase merge is approximately proportional to the total volume of information that moves through the merge. In order to make this idea precise, we assume that the dispersion routine produces strings of approximately the same size; this size will be our unit of information, the unit string. The size of a string formed by merging several strings is the sum of the sizes of the input strings and the size of a dummy string is zero. We say that a string is moved when that string or any string formed from it by a sequence of one or more merges becomes one of the inputs for a merge. The volume of information moved by the polyphase merge is then equal to the sum of the products of the size of each string of the starting distribution and the number of times that string is moved. In this paper we will show how this volume may be minimized.

3. The Movement Numbers.

In general, the polyphase merge does not move all of the S^n strings of the stage n perfect distribution the same number of times. It is for this reason that the polyphase sort is much more difficult to analyze than other merge sorting algorithms. However, much useful information is supplied by the set of movement numbers M?(j) which are defined for $n > 1$, $1 \le i \le t$, and all integers j by the relations

$$(3.1)\qquad \begin{aligned} M_i^1(1) &= 1 && \text{for } 1 \le i \le t\,, \\ M_i^1(j) &= 0 && \text{for } j \ne 1 \text{ and } 1 \le i \le t\,, \\ M_1^n(j) &= M_t^{n-1}(j-1) && \text{for } n > 1\,, \text{ and} \\ M_i^n(j) &= M_{i-1}^{n-1}(j) + M_t^{n-1}(j-1) && \text{for } n > 1 \text{ and } 2 < i < t\,. \end{aligned}$$

We claim that M?(j) is precisely the number of strings on tape unit i of-the stage n perfect distribution which will be moved exactly j times by the poly-phase merge. For this to make any sense it is necessary that $M_i^n(j)$ be nonzero only if $1 \le j \le n$ and that $S_i^n = M_i^n(1) + \ldots + M_i^n(n)$.

We will prove these assertions together by induction on n. When $n = 1$, everything is obvious since each of the tapes $1,\ldots,t$ of the perfect distribution contains exactly one string which will be moved by the poly-phase merge exactly once. Now suppose that $n > 1$ and that everything has been proved for stage $n-1$. For $M_i^n(j)$ to be nonzero we must have, by (3.1), $M_t^{n-1}(j-1) \ne 0$ or $i \ge 2$ and $M_{i-1}^{n-1}(j) \ne 0$. These inequalities imply that $1 \le j-1 \le n-1$ or $1 \le j \le n-1$ which both imply that $1 \le j \le n$. We may show that $S_i^n = M_i^n(1) + \ldots + M_i^n(n)$ by summing the last two formulas of (3.1) over j and by applying the corresponding equality for stage $n-1$ and the last two formulas of (2.1). We recall that the stage n polyphase merge is performed by merging S_1^n strings from each of the tapes and then by applying the stage n-1 polyphase merge. A string on unit one which will be moved j times will become part of a string on the output tape which will be moved j-1 times. Since every string on the output tape contains exactly one string from unit one and since the output tape becomes unit t for the stage n-1 merge, we see that unit one must contain exactly $M_t^{n-1}(j-1)$ strings that will be moved exactly j times. If $2 \leq i \leq t$, then a j movement string on unit i will either be moved to the output tape or will remain on the tape. From similar considerations, we see that unit i must contain exactly $M_t^{n-1}(j-1) + M_{i-1}^{n-1}(j)$ strings which will be moved exactly j times. This completes the proof.

<u>Example 3.1</u>. Table 3.1 lists some of the movement numbers in the case $t = 4$.

In this paper we will make use of quite a few sets of numbers which are defined using the movement numbers $M_i^n(j)$. We list the definitions:

$$M^n(j) = M_1^n(j) + \cdots + M_t^n(j) ,$$

$$S_i^n(j) = M_i^n(1) + \cdots + M_i^n(j) ,$$

(3.2) $$S^n(j) = M^n(1) + \ . \ . \ . + M^n(j) = S_1^n(j) + \ . \ . \ . + S_t^n(j) ,$$

$$G_i^n(j) = S_i^n(1) + \ . \ . \ . + S_i^n(j) ,$$

$$G^n(j) = S^n(1) + \ . \ . \ . + S^n(j) = G_1^n(j) + \ . \ . \ . + G_t^n(j) .$$

In addition, we have already defined

$$S_i^n = S_i^n(n)$$

$$S^n = S^n(n) = S_1^n + \ . \ . \ . + S_t^n .$$

In a number of the form $A_i^n(j)$ the superscript n is the associated stage number, the subscript i is the number of a tape unit, and j is some number of movements. $A^n(j)$ is formed from $A_i^n(j)$ by summing over $i = 1,\ldots,t$ and A_i^n is formed from $A_i^n(j)$ by setting $j = n$. In a similar fashion we may form A^n from A_i^n or $A^n(j)$.

Except for the numbers $G_i^n(j)$ and $G^n(j)$, which are used in connection with the volume function, the various sets of numbers which we have defined express some simple properties of the perfect stage n distribution:

$M_i^n(j)$ The number of strings on unit i which will be moved exactly j times.

$M^n(j)$ The number of strings which will be moved exactly j times.

$S_i^n(j)$ The number of strings on unit i which will be moved at most j times.

$S^n(j)$ The number of strings which will be moved at most j times.

S_i^n The number of strings on unit i.

S^n The total number of strings.

A set of numbers $A^n(j)$ is said to be a t-array if the following relation is satisfied for all integers n and j:

$$A^n(j) = A^{n-1}(j-1) + \ldots + A^{n-t}(j-1) . \tag{3.3}$$

We will call a sum of this form a t-sum. When a t-array is represented as a table of numbers, then we will let j index the rows and n index the columns. It is clear that the t-array $A^n(j)$ is completely determined by its values on the vertical strip $1-t \le n \le 0$ (or any other strip of width t). We will call this strip the initialization region.

Most of the sets of numbers which we have defined can be expressed as t-arrays. The t-array approach exposes many of the interesting properties of these numbers which are obscured by the original definitions. Since all of these numbers are defined in terms of the movement numbers, we will begin by showing that the movement numbers may be defined as t-arrays.

For each $i = 1,\ldots,t$ we define the t-array $A_i^n(j)$ by specifying that $A_i^{i-t}(0) = 1$ is the only nonzero element of the initialization region for $A_i^n(j)$. We will show that for all $n \geq 1$, $1 \leq i \leq t$ and all j that $A_i^n(j) =$ M?(j) . It is clear that the only nonzero values in the columns $n = -t$ are $A_t^{-t}(-1) = 1$ and $A_i^{-t}(0) = -1$ for $1 \leq i < t$. If we let δ_b^a denote the Kronecker Delta, then for $-t \leq n \leq 0$ we have $A_t^n(j) = \delta_0^n\delta_0^j + \delta_{-t}^n\delta_{-1}^j$ and $A_i^n(j) = \delta_{i-t}^n\delta_0^j - \delta_{-t}^n\delta_0^j$ for $1 \leq i < t$. Therefore, for $1-t \leq n \leq 0$, we have

$$A_t^{n-1}(j-1) = \delta_0^{n-1}\delta_1^j + \delta_{1-t}^n\delta_0^j = \delta_{1-t}^n\delta_0^j = A_1^n(j)$$

and for $2 \leq i \leq t$

$$\begin{aligned} A_{i-1}^{n-1}(j) + A_t^{n-1}(j-1) &= \delta_{i-t-1}^{n-1}\delta_0^j - \delta_{-t}^{n-1}\delta_0^j + \delta_0^{n-1}\delta_0^{j-1} + \delta_{-t}^{n-1}\delta_{-1}^{j-1} \\ &= \delta_{i-t}^n\delta_0^j = A_i^n(j) \quad . \end{aligned}$$

These relations correspond to the last two formulas of (3.1) and since they hold for n and j in the initialization region, they can be extended to all values of n and j by a simple induction argument using the recurrence relation (3.3). Since the only nonzero values in the columns $n = 1$ are $A_i^1(1) = 1$, we see that the numbers $A_i^n(j)$ also satisfy the first two relations of (3.1). We therefore conclude that $M_i^n(j) = A_i^n(j)$ for all $n > 1$.

Below we list the various t-arrays in which we will be interested and specify the nonzero values in their respective initialization regions:

$M_i^n(j)$	$M_i^{i-t}(0) = 1$,	
$M^n(j)$	$M^n(0) = 1$	for $1-t \le n \le 0$,
$S_i^n(j)$	$S_i^{i-t}(j) = 1$	for $j \ge 0$,
$S^n(j)$	$S^n(j) = 1$	for $1-t \le n \le 0$ and $j \ge 0$,
$G_i^n(j)$	$G_i^{i-t}(j) = j+1$	for $j \ge 0$,
$G^n(j)$	$G^n(j) = j+1$	for $1-t \le n \le 0$ and $j \ge 0$.

It is not difficult to show that these t-arrays satisfy the definitions given in (3.2).

<u>Example 3.2</u>. Table 3.2 shows a portion of the t-array $S_i^n(j)$ when $i = 2$ and $t = 4$. In this case, the only nonzero elements of the initialization region are $S_2^2(j) = 1$ for $j \ge 0$.

4. Optimal Merging.

In this section we will examine some of the properties of the poly-phase merge when it is implemented using read-forward tape units. (Read-forward tape units can be thought of as queues in which strings are written at the end of the tape and are read from the beginning.) Of particular importance is the close relationship with generalized Fibonacci numbers. These results will be used to construct an optimal polyphase merge algorithm which has a number of desirable characteristics.

From (2.1) it is easily shown that

$$S_t^n = S_t^{n-1} + \dots + S_t^1 + 1 \qquad \text{for } 2 \le n \le t \text{ , and}$$

$$S_t^n = S_t^{n-1} + \dots + S_t^{n-t} \qquad \text{for } n > t \text{ .}$$

If we define $F_n = 0$ for $n < 0$, $F_0 = 1$, and $F_n = S_t^n$ for $n > 1$, then, from the above relations, we have

$$F_n = F_{n-1} + \dots + F_{n-t} \tag{4.1}$$

for $n \geq 1$. Because of the similarity of (4.1) to the defining recurrence relation for the Fibonacci numbers, we will call these numbers F_n the t-Fibonacci numbers.

The t-Fibonacci numbers play a central role in the problem of analyzing the motion of the strings for the read-forward poly-phase merge. Indeed, suppose that the strings have been dispersed according to the perfect stage n distribution and that the string positions on each tape are numbered from zero starting at the front of the tape. If we perform the polyphase merge starting with stage n , then the number of times m that a string in position p on one of the tapes will be moved is computed by the following algorithm;

Algorithm 4.1 Simulate String Motion.

Step 1. Let $m = 1$, $k = n-1$, and $q = p$.

Step 2. If $k = 0$, then terminate.

Step 3. If $q < F_k$, then go to Step 5.

Step 4. Let $q = q-F_k$ and go to Step 6.

Step 5. Let $m = m+1$.

Step 6. Let $k = k-1$ and go to Step 2.

This algorithm simply follows the motion of the string as the polyphase merge is performed. In particular, $k+1$ is the stage number of the polyphase merge being performed. If $q < F_k = S_t^k = S_1^{k+1}$, then the string will be moved (and m incremented), but its position on the output tape will be the same as its position on the input tape. If $q \geq F_k$, then the string will not be moved but its position will be changed to $q-F_k$ since F_k strings will have been removed from the tape. Since we are simulating the poly-phase merge, we always have $q < F_{k+1} = S_t^{k+1}$ (this may also be shown by induction) so that $q = 0$ when the algorithm terminates.

Let us define the sequence $s_1, s_2, \ldots, s_{n-1}$ as follows: we let $s_j = 1$ if, when performing Algorithm 4.1, we perform Step 4 with $k = j$; otherwise, we let $s_j = 0$. Obviously, the number of times that the string in position p is moved is $n - s_1 - s_2 - \cdots - s_{n-1}$. From the mechanics of the algorithm and the fact that it terminates with $q = 0$, we find that

$$p = \sum_{j=1}^{n-1} s_j F_j \quad .$$

Since a string can not remain on a tape for t consecutive merges, we see that the sequence $s_1, \ldots, s_{n-1}$ cannot contain more than t-1 consecutive ones.

We have shown that p may be represented as a sum of distinct t-Fibonacci numbers in such a way that at most t-1 consecutive t-Fibonacci numbers appear in the sum. We will now study some properties of this type of representation.

We define a t-sequence to be a sequence $s_1, s_2, \ldots$ of zeros and ones with the properties that only finitely many ones appear and that no t consecutive ones appear. It will sometimes be convenient to assume that $s_m = 0$ for $m \leq 0$. The length $L(s)$ of a t-sequence s is defined to be the largest m for which $s_m = 1$ or zero if $s_m = 0$ for all m. If s and s' are t-sequences, then we say that $s < s'$ if for some m we have $s_m < s'_m$ (i.e., $s_m = 0$ and $s'_m = 1$) and $s_n = s'_n$ for all $n > m$. It is clear that this defines a linear ordering of the set of all t-sequences.

A t-sequence s represents a number $F(s)$ in the sense that

$$F(s) = \sum_{n \geq 1} s_n F_n .$$

We have the following theorem concerning such representations:

Theorem 4.1. For each $p \geq 0$, there exists a unique t-sequence $R(p)$ for which $p = F(R(p))$. Furthermore, if $p < p'$, then $R(p) < R(p')$.

First we require some lemmas:

Lemma 4.1. If s is a t-sequence for which $L(s) < n$, then $F(s) < F_n$.

Proof. If $L(s) = 0$, then $F(s) = 0 < F_n$ for all $n > 0$. Now suppose that s is a t-sequence of length $m > 0$ and that the result has been proved for all t-sequences of length less than m. Clearly there must be a $k \geq 0$ with $m-t+1 \leq k \leq m$ for which $s_k = 0$. We form the' t-sequence s' by letting $s'_j = s_j$ for $j < k$ and $s'_j = 0$ for $j \geq k$. If $k = 0$, then $F(s') = 0 < F_k$. If $k > 0$, then $L(s') < k < m$ so that by our induction hypothesis we have $F(s') < F_k$. Consequently, if $m < n$, then we have

$$F(s) = F(s') + \sum_{j>k} s_j F_j \leq F_k + F_{k+1} + \ . \ . \ . + F_m$$

$$= F_{m \ t+1} + \ldots + F_m = F_{m+1} \leq F_n \ . \qquad \square$$

Lemma 4.2. If s and s' are t-sequences for which $s < s'$, then $F(s) < F(s')$.

Proof. Let m be the largest integer for which $s_m < s'_m$. We then have $s_m = 0$ and $s_n = s'_n$ for $n > m$. From Lemma 4.1 it follows that

$$F(s) = \sum_{k \geq 1} s_k F_k = \sum_{k=1}^{m-1} s_k F_k + \sum_{k>m} s_k F_k$$

$$< F_m + \sum_{k>m} s_k F_k \leq \sum_{k>1} s'_k F_k = F(s') \ . \qquad \square$$

Lemma 4.3. There are precisely F_n t-sequences for which $L(s) < n$.

Proof. We will use induction on n. Clearly the result is true when $n = 1$. If $n > 1$, then we may partition the set of all t-sequences s for which $L(s) < n$ into t classes as follows: for each k with $1 \leq k \leq t$, we define the k-th class to be the set of all such t-sequences s which have the property that $s_j = 1$ for $n-k < j < n$ (this condition is vacuous when $k = 1$) and $s_{n-k} = 0$. Assuming that the lemma has been proved for all $n' < n$, we will show that for each k that the k-th class contains $F_{n \ k}$ elements. If $n-k < 0$, then we must have $s_0 = 1$ for any s in the k-th class and therefore the k-th class contains

$F_{n-k} = 0$ elements, If $n-k \geq 0$, then for any t-sequence s in the k-th class, we may construct a t-sequence s' by letting $s'_j = s_j$ for $j < n-k$ and $s'_j = 0$ for $j \geq n-k$. It is easily seen that this construction defines a bijection between the k-th class and the set of all t-sequences s' for which $L(s') < n-k$. Since the latter set contains F_{n-k} elements, so does the k-th class. Summing over k , we find that there are exactly $F_{n-1} + \ldots + F_{n-t} = F_n$ t-sequences s for which $L(s) < n$. □

Proof of Theorem 4.1. It is clear that the numbers F_n are unbounded. Therefore, if $p > 0$ is given, then we can find an n for which $p < F_n$. By Lemma 4.3, there are F_n t-sequences of length less than n which by Lemma 4.1 are mapped by F into the nonnegative integers less than F_n . By Lemma 4.2, this mapping is injective and therefore, by pigeonholing, is surjective. Consequently, we can find a t-sequence R(p) for which $p = F(R(p))$. Uniqueness and the strict monotony of the mapping R both follow from Lemma 4.2. □

Remarks. Theorem 4.1 is an extension of a well known theorem of Zeckendorf which concerns the representation of integers by sums of Fibonacci numbers. The extension given here is due to Knuth ([5], Exercise 5.4.2-10) although our proof is somewhat different. Lynch [6] has generalized this result and has shown how generalized Fibonacci numbers may be used to control dispersion and merging in the standard polyphase sort. There is another extension of Zeckendorf's theorem which contains the others as special cases. Let r(n) be a positive integer-valued function of $n > 1$ which has the property that $r(n) > 2$

for infinitely many values of n . We define the <u>r-Fibonacci numbers</u> f_n by $f_n = 0$ for $n<0$, $f_0 = 1$, and $f_n = f_{n-1} + \cdots + f_{n-r(n)}$ for $n > 1$. Every positive integer is uniquely represented by a sum of r-Fibonccci numbers f_n with distinct subscripts $n \geq 1$ which has the property that if $f_{m-1},\ldots,f_{m-r(m)}$ all appear in the sum, then so does f_m . A proof may be constructed along the lines of our proof of Theorem 4.1 although some care is required when $r(n) = 1$. When $r(n) = n$ for all $n \geq 1$ then the above result implies the existence and uniqueness of representations in the binary number system.

Let $D(p)$ be the number of ones in the t-sequence $R(p)$. In the discussion following Algorithm 4.1 we showed that if a string appears in position p on some tape of the perfect stage n distribution, then the polyphase merge will move the string exactly $n-D(p)$ times. Therefore, it is of some interest to determine those values of p for which $D(p)$ takes a given value.

Let j be a nonnegative integer. We define $E(j)$ to be the smallest nonnegative integer p for which $D(p) = j$. The following theorem and the corollary provide methods of computing $E(j)$:

<u>Theorem 4.2</u>. $E(0) = 0$. If $j > 0$, then $E(j) = E(j-1) + F_{j+k}$ where $k = \lfloor (j-1)/(t-1) \rfloor$.

<u>Proof</u>. We will prove the theorem together with the fact that $L(R(E(j))) = j+k$ for $j > 0$ by induction on j . Clearly $E(0) = 0$. Now suppose that $j > 0$ and define $s = R(E(j))$, $m = L(s)$, and $p = E(j) - F_m$. Clearly $D(p) = j-1$ so that $p \geq E(j-1)$. If we let $k = \lfloor (j-1)/(t-1) \rfloor$, then we must have $m \geq j+k$ for othrrwise s would contain t consecutive ones or would have less than j ones. It follows that $E(j) \geq E(j-1) + F_{j+k}$

and to prove equality, it is sufficient to show that $D(E(j-1)+F_{j+k}) = j$. We assume that everything has been proved for $j' < j$. If $k = 0$, then we clearly have

$$E(j-1) = F_1 + \ldots + F_{j-1}$$

(the sum being zero when $j = 1$) and since $j < t$ we have $D(E(j-1)+F_{j+k}) = j$. We also observe that $L(s) = j = j+k$. If $k > 0$, then let $j' = k(t-1)+1$. Clearly $j' \le j$ and we have $k = \lfloor (n-1)/(t-1) \rfloor$ for $j' \le n \le j$. From our induction hypothesis we obtain

$$E(j-1) + F_{j+k} = E(j'-1) + F_{j'+k} + \ldots + F_{j+k} .$$

However, if we let $k' = \lfloor (j'-2)/(t-1) \rfloor$, then $L(R(E(j'-1))) = j'+k'-1 = j'+k-2$. Since $j-j' < t-1$, it follows that the t-sequence $s' = R(E(j'-1))$ remains a t-sequence if we let $s'_n = 1$ for $j'+k \le n < j+k$. It follows at once that $D(E(j-1)+F_{j+k}) = j$ and that $L(s) = j+k$. This completes the proof. □

<u>Corollary 4.1</u>. For $j > 0$ and k defined as before we have

$$E(j) = \sum_{m=kt}^{j+k} F_m - 1$$

the sum having at most t terms.

<u>Proof</u>. The proof is by induction on j. When $j = 1$ we have $k = 0$ so the above expression is $F_0+F_1-1 = 1 = E(1)$. Now suppose that the corollary has been proved for all $j' < j$, in particular, for $j' = k(t-1)$. Since $\lfloor (n-1)/(t-1) \rfloor = k$ for $j' < n \le j$ we have from the theorem

$$E(j) = E(j') + F_{j'+k+1} + \ldots + F_{j+k} .$$

Applying the corollary with j' and $k' = \lfloor (j'-1)/(t-1) \rfloor = k-1$, we obtain

$$E(j') = \sum_{m=k't}^{j'+k'} F_m - 1 = \sum_{m=kt-t}^{kt-1} {}_mF - 1$$

$$= F_{kt} - 1 \ .$$

Since $j'+k+1 = kt+1$, it follows that

$$E(j) = F_{kt} + \ . \ . \ . + F_{j+k} - 1 \ .$$

Finally, we observe that $j+k-kt = 1+(j-1)-k(t-1) < 1+(t-1) = t$ so the sum contains at most t terms. □

If $j > 1$, then there are infinitely many positive integers p for which $D(p) = j$. We have just shown how to find the smallest such p so now we will show how to find the others. We will do this by constructing an algorithm which computes, given $p > 0$, the smallest $p' > p$ for which $D(p') = D(p)$.

Let $s = R(p)$ and $s' = R(p')$. We already know that $s < s'$ if and only if we can find an m for which $s_m = 0$, $s'_m = 1$, and $s'_k = s_k$ fork $>m$. Consequently, to find the smallest $p' > p$ for which $D(p') = D(p)$, we must first find a suitable value of m . Clearly the smaller the value of m that is chosen, the smaller the value of p' . There are three conditions that m must satisfy: First there is the condition $s_m = 0$ which was given above. Second, we must have $s_k = 1$ for some $k < m$ for otherwise we would have $D(p') > D(p)$. Third, we can not have $s_{m+1} = \ . \ . \ . = s_{m+t-1} = 1$ for otherwise any sequence s' with $s'_m = 1$ and $s'_k = s_k$ for $k > m$ will not be a t-sequence.

Therefore, let us choose m to be the smallest integer for which $s_m = 0$, $s_{m-1} = 1$, and $s_{m+1} + \ldots + s_{m+t-1} < t-1$. This choice can always be made since $m = L(s)+1$ satisfies the requirements. If we define p' by

$$p' = E(s_1 + \ldots + s_{m-2}) + F_m + \sum_{k>m} s_k F_k$$

then it is easily verified that $p' > p$ and that $D(p') = D(p)$ and that it is the smallest integer to have these properties.

In order to use the formula above, it is necessary to know the representation $R(p)$ of p. The following algorithm computes p' by combining the conversion of p to $R(p)$ (using a technique similar to Algorithm 4.1) and the search for m. The algorithm is easily implemented on digital computers since it is fully arithmetic and does not involve t-sequences.

Algorithm 4.2. Find the smallest $p' > p$ for which $D(p') = D(p)$.

Step 1. Let $q = p$ and $k = 0$ and choose some m for which $p < F_m$.

Step 2. If $F_m \leq q$, then go to Step 4.

Step 3. Let $m = m-1$. If $m = 0$, then go to Step 10; otherwise go to Step 2.

Step 4. Let $q' = q$, $m' = m$, and $k' = k$.

Step 5. If $m < t$, then go to Step 7.

Step 6. If $q < F_{m+1} - F_{m-t+1}$, then go to Step 7; otherwise, let $q = q - (F_{m+1} - F_{m-t+1})$, $m = m-t$, and $k = k+t-1$ and go to Step 8.

Step 7. Let $q = q-F_m$, $m=m-1$, and $k = k+1$.

Step 8. If $m = 0$, then go to Step 10.

Step 9. If $F_m \le q$, then go to Step 5; otherwise, *go to Step 3.

Step 10. Terminate with $p' = p - q' + F_{m'+1} + E(k-k'-1)$.

To understand this algorithm, let $s = R(p)$. If $F_m \le q$ in Step 2, then $s_m = 1$ and the values of q , m , and k are saved. The check that $q \ge F_{m+1} - F_{m-t+1} = F_m + \ldots + F_{m-t+2}$ determines whether or not $s_m = \ldots = s_{m-t+2} = 1$ and $s_{m-t+1} = 0$. Steps 6 and 7 decrement m in such a way as to bypass ineligible values of m , that is, those for which $s_{m+1} = 1$ or $s_{m+1} = 0$ and $s_{m+2} = \ldots = s_{m+t} = 1$. The variable k contains the number of nonzero values of s_m which have been encountered. At completion, the last values of q , m , and k saved by Step 4 enable us to compute p' .

Example 4.1. First we list some values of F_n and $E(n)$ for the case $t = 4$:

n	F_n	E(n)	n	F_n	E(n)
1	1	1	9	188	1339
2	2	3	10	361	3921
3	4	7	11	693	8897
4	8	22	12	1340	18488
5	15	51	13	2582	54126
6	29	97	14	4976	122820
7	46	285	15	9591	255232
8	98	646	16	18489	747209

If we let $p = 3913$ and let $s = R(p)$, then it is easily shown that

$$s = \{0,1,1,1,0,1,1,0,1,1,1,0,1,0,0,\ldots\}$$

so the representation of p' has the form

$$s' = \{1,1,0,0,1,1,1,0,1,1,1,0,1,0,0,\ldots\}$$

and it follows that $p' = 3917$.

We are now in a position to examine the problem of optimizing the polyphase merge for an arbitrary initial distribution. Suppose that the dispersion routine writes $x_1,\dots,x_t$ strings to units 1, . . ., t , respectively, and that the choice is made to perform the polyphase merge starting with stage n . The only requirement on n is that $x_i < S_i^n$ for each i . If this requirement is met, then it is only necessary to include $S_i^n - x_i$ dummy strings on each tape i in order to obtain the perfect stage n distribution. We have already observed that the number of times that a string is moved depends upon its tape position. Therefore, the manner of placement of the dummy strings has a direct influence on the volume of information moved.

It is quite obvious how to arrange the dispersed strings and the dummy strings so as to minimize the volume of information moved. On each unit i , we place $M_i^n(1)$ of the dispersed strings in the $M_i^n(1)$ string positions which will be moved once, $M_i^n(2)$ strings into the positions which will be moved twice, and so on, until we exhaust the x_i dispersed strings; we then place dummy strings in the remaining $S_i^n - x_i$ string positions. In this way we insure that the dummy strings are in the positions which will be moved the most.

One practical difficulty with the above approach is the problem of placing the dummy strings if the dispersed strings are already on the tapes. With read-forward tape units it is not permissable to write randomly on a tape. For this reason, we will transform the above approach into a practical algorithm in which dummy strings do not explicitly appear.

If $S_i^n(j-1) < x_i < S_i^n(j)$, then, with the above scheme, there will be some j movement string positions which contain dispersed strings and others which contain dummy strings. We have not said how they are to be arranged. We propose placing all of the j movement dispersed strings in front of all of the j movement dummy strings on each tape. It does however have the important property that the pattern is preserved as the polyphase merge is performed. It is not difficult to see that any time during the operation of the merge, any k movement strings of nonzero length will be in front of any k movement dummy strings on the same tape.

Another important consequence of this choice is that we are able to calculate the positions of the j movement dispersed strings. Since these, positions p have the property that $j = n-D(p)$, we see that the first of these positions is $E(n-j)$ and that the remaining positions are calculated by repeated application of Algorithm 4.2. Since the pattern is preserved, the same observation holds throughout the polyphase merge.

The algorithm which we will present is controlled by the two arrays $C[i,j]$ and $P[j]$ $(0 \le i \le t,\ 1 \le j \le n)$. $C[i,j]$ will contain the number of strings on tape i which will be moved j times and $P[j]$ contains the next j movement position on the input tapes. It is also convenient to have arrays for the numbers F_m and $E(m)$, but we will not mention these explicitly.

The inputs to the algorithm are the numbers $x_1,\ldots,x_t$ of dispersed strings on tape units $1,\ldots,t$ and the starting stage number n of the polyphase merge to be performed. (The next three sections of this paper are devoted to the proper choice of these numbers.) In order to facilitate implementation, we will explicitly mention the tape rewind operations required.

Algorithm 4.3 Optimal Read-Forward Polyphase Merge.

Step 1. [Initialization.] Let $C[i,j] = M_i^n(j)$ for $1 \le j \le n$ and $1 \le i \le t$. Let $C[0,j] = 0$ for $1 \le j \le n$. Let $m = n$ and $u = 0$. Rewind all of the tapes.

Step 2. '[Initialize C.] For each $i = 1,\ldots,t$ find the smallest j for which $x_i \le C[i,1]+\ldots+C[i,j]$; let $C[i,j] = x_i - C[i,1] - \ldots - C[i,j-1]$ and let $C[i,k] = 0$ for $j < k \le n$.

Step 3. [Test for termination.] If $m > 0$, then go to Step 4. Otherwise, the sort is finished. Rewind all of the tapes. The sorted records are on tape u'.

step 4. [Initialize for stage m .] For $j = 1,\ldots,m$ let $P[j] = E(m-j)$ if $C[i,j] > 0$ for some i; otherwise, let $P[j] = F_{m-1}$.

Step 5. [Test for the end of a merge,] Find the value of j which minimizes $P[j]$ $(1 \le j \le m)$. If $P[j] \ge F_{m-1}$, then go to Step 9.

Step 6. [Merge some strings.] Merge one string from each unit $i \ne u$ for which $C[i,j] > 0$ and write the resulting string to unit u.

Step 7. [Update C .] If $m > 1$, then increment $C[u,j-1]$ by one. For each $i \ne u$ for which $C[i,j] > 0$, decrement $C[i,j]$ by one. If each of these decrements results in a value of zero, then let $P[j] = F_{m-1}$ and go to Step 5.

Step 8. [Update Q .] Using Algorithm 4.2, find the smallest $p > P[j]$ for which $D(p) = D(P[j])$. Let $P[j] = p$ and go to Step 5.

Step 9. [End of a merge.] Let $m = m-1$, $u' = u$, and $u = u+1 \bmod T$. Rewind tapes u and u' and go to Step 3.

In view of the discussion, this algorithm is reasonably straightforward. However, we will comment on a few points. The computations required in Step 1 can be performed without any additional storage by careful use of the recurrence relations (3.1). Our use of F_{m-1} in Steps 4, 5, and 7 is accounted for by the fact that $F_{m-1} = S_t^{m-1} = S_1^m$ which is the number of strings produced by the Stage m merge; consequently F_{m-1} is the first position which will not be used for this merge.

Although the computations required by the algorithm are formidable, they do not really require much time. The bulk of the computation is performed in Steps 5, 7, and 8 which are performed once for each string that is output. Since a unit string will represent a large fraction of the storage utilized by the sort, it is clear the time required will be insignificant when compared with the time required for merging.

The storage requirements are not much larger than for other polyphase merge algorithms. The only extra storage which is not required by other algorithms is the storage for the arrays C and P and, possibly, the arrays containing the numbers $E(m)$ and F_m for a suitable range of m . We remark that the additional storage required for these arrays when merging 100000 strings, using ten tapes and the dispersion algorithm we will describe, should be less than four hundred locations.

Remarks. Shell [8] has described an optimum polyphase sort which is somewhat different from ours. He describes a method of generating the $D(0), D(1), D(2), \ldots$ directly and uses an array based on this sequence to control the placement of the strings and the assumed placement of the dummy strings. Unfortunately, this array becomes prohibitively large for large applications. An account of Shell's work also appears in [5] (Section 5.4.2).

5. The Volume Function.

Let us suppose that we have $x \leq S^n$ unit strings which we wish to merge with the stage n polyphase merge. Obviously, in order to minimize the volume, we should place the unit strings into the positions which will be moved the least and the dummy strings into the positions which will be moved the most. Thus, if $S^n(j) \leq x \leq S^n(j+1)$, then unit strings should be placed in all of the $S^n(j)$ positions which will be moved j or fewer times and in $x - S^n(j)$ of the $j+1$ movement positions When this is done, the volume of information which will be moved by the merge is found to be

$$\sum_{k=1}^{j} k M^n(k) + (j+1)(x - S^n(j)) \quad .$$

We will call the value of this expression the <u>volume function</u> and denote it by $V^n(x)$. The expression may be simplified by observing that

$$\begin{aligned}
(j+1)S^n(j) - \sum_{k=1}^{j} k M^n(k) &= \sum_{k=1}^{j} (j-k+1) M^n(k) \\
&= \sum_{k=1}^{j} \sum_{i=k}^{j} M^n(k) = \sum_{i=1}^{j} \sum_{k=1}^{i} M^n(k) \\
&= \sum_{i=1}^{j} S^n(i) = G^n(j) \quad .
\end{aligned}$$

We may now write

$$(5.1) \qquad V^n(x) = (j+1)x - G^n(j)$$

where $S^n(j) \leq x \leq S^n(j+1)$.

In Section 4, we looked at the similar problem of optimizing the stage n polyphase merge when it is known that tapes $1,\ldots,t$ contain

$x_1, \ldots$ [illegible] dispersed strings, respectively. By **similar reasoning,** the volume of information moved in this case is

$$V_1^n(x_1) + \ldots + V_t^n(x_t)$$

where each $V_i^n(x_i)$ represents the contribution of tape i to the volume. This contribution is given by

$$(5.2) \qquad V_i^n(x_i) = (j_i+1)x_i - G_i^n(j_i)$$

where j_i is chosen to satisfy $S_i^n(j_i) \leq x_i \leq S_i^n(j_i+1)$.

Obviously we must have

$$V^n(x_1 + \ldots + x_t) \leq V_1^n(x_1) + \text{[illegible]} .$$

We are interested in those distributions $x_1, \ldots, x_t$ for which we have equality. Such a distribution is said to be <u>optimal for stage n</u> .

<u>Theorem 5.1</u>. A distribution $x_1, \ldots, x_t$ is optimal for stage n if and only if we can find a j such that $S_i^n(j) \leq x_i \leq S_i^n(j+1)$ for each i .

<u>Proof</u>. If the condition is satisfied, then **optimality** for stage n follows at once from formulas **(5.1) and (5.2)** and **the** fact that $G^n(j) = G_1^n(j) +$ [illegible]

Conversely, suppose that $x_1, \ldots, x_t$ does not satisfy the condition. We can then find a j and two indices a and b such that $x_a < S_a^n(j)$ and $x_b > S_b^n(j)$. If we define the **distribution** $x_1', \ldots, x_t'$ by $x_a' = x_a+1$, $x_b' = x_b-1$, and $x_i' = x_i$ for $i \neq a,b$, then it is clear that

$$V_a^n(x_a') - V_a^n(x_a) \leq j \quad \text{and}$$

$$V_b^n(x_b) - V_b^n(x_b') \leq j+1 .$$

It follows that

$$V_1^n(x_1') + \dots V_t^n(x_t') < V_1^n(x_1) \dots V_t^n(x_t)$$

and therefore $x_1, \dots, x_t$ can not be optimal for stage n . $\square$

Example 5.1. We let t = 4 as in our other examples and x = 500 . From the table in Example 2.1, we see that the smallest value of n for which $x < S^n$ is 9 . Let us evaluate $V^n(x)$ for this value of n . Since $S^n(5) = 338 < x < 534 = S^n(6)$, we may apply formula (5.1) with j = 5 to obtain

$$V^n(x) = (j+1)x - G^n(j) = 6\cdot 500 - 478 = 2522 .$$

This volume is the best possible volume obtainable with the stage 9 merge no matter how the strings are dispersed. If we let n = 10 , then a similar calculation shows that $V^n(x) = 2448$ which illustrates how the choice of a larger stage number than the minimum may improve the performance of the polyphase sort. We will discuss this subject in Section 6.

We will conclude this section with two theorems concerning the volume function which will be required later.

Theorem 5.2. If $x < S^n$, then $V^{n+1}(x) - V^n(x) < x$.

Proof. We may assume that $x > 0$. Let j and k be the unique integers for which

$$S^n(j) < x \leq S^n(j+1) \quad \text{and} \quad S^{n+1}(k) < x < S^{n+1}(k+1)$$

From the recurrence relation for t-arrays, we see that $S^n(j+1) \leq S^{n+1}(j+2)$, so that

$$S^{n+1}(k) < x \le S^{n}(j+1) \le S^{n+1}(j+2)$$

which implies that $k < j+1$. From the recurrence relation, we also have $G^{n}(k-1) \le G^{n+1}(k)$. We may now write

$$\begin{aligned}
V^{n+1}(x) - V^{n}(x) &= (k+1)x - G^{n+1}(k) - (j+1)x + G^{n}(j) \\
&= (k-j)x + G^{n}(j) - G^{n+1}(k) \\
&\le (k-j)x + G^{n}(j) - G^{n}(k-1) \\
&< (k-j)x + (j-k+1)S^{n}(j)
\end{aligned}$$

<u>Theorem 5.3</u>. Suppose that $0 \le x_1 \le \ldots \le x_t$ and that $x_i \le S_i^n$ for each i . If $x_1', \ldots, x_t'$ is a permutation of $x_1, \ldots, x_t$ which has the property that $x_i' \le S_i^n$ for each i , then we have

$$V_1^n(x_1) + \ldots + V_t^n(x_t) \le V_1^n(x_1') + \ldots + V_t^n(x_t') \quad .$$

<u>Proof</u>. First we will prove the result for a simple interchange. Suppose that $1 < a < b < t$ and that $x_a \le S_b^n$, $x_b \le S_a^n$, and $0 \le x_a < x_b$. If $x_a \le y < x_b$, then let j and j' be the unique integers for which

$$S_a^n(j) \le y < S_a^n(j+1) \quad \text{and} \quad S_b^n(j') \le y < S_b^n(j'+1) \quad .$$

Since $S_a^n(k) \le S_b^n(k)$ for all k , it is clear that $j \ge j'$ and therefore

$$V_b^n(y+1) - V_b^n(y) = j'+1 \le j+1 = V_a^n(y+1) - V_a^n(y) \quad .$$

By summing over y , we obtain

$$V_b^n(x_b) - V_b^n(x_a) \le V_a^n(x_b) - V_a^n(x_a)$$

which may be rewritten as

$$V_a^n(x_a) + V_b^n(x_b) \le V_a^n(x_b) + V_b^n(x_a) \quad :$$

The general result is proved by permuting the numbers $x'_1,\ldots,x'_t$ into $x_1,\ldots,x_t$ by a series of interchanges which successively place the proper values into positions $1,\ldots,t$ and by applying the above result at each step. It is clear that we only change the numbers y_a and y_b in positions $a < b$ when $y_b < y_a$. Also, since $y_a \le S_a^n \le S_b^n$, we never place a number which exceeds S_i^n into any position i . □

6. Optimal Dispersion.

In much of the literature on polyphase sorting, it is assumed that the best starting stage number when merging x strings is the smallest n for which $x \le S^n$. This method generally gives nice looking results when the usual polyphase merge algorithms are used. However, when an algorithm such as Algorithm 4.3 or the optimum polyphase sort of Shell [8] is employed, it is found that better results may be obtained by choosing larger values of n . In this section we will investigate the problem of finding the value of n which minimizes $V^n(x)$.

A good starting point is the following lemma on t-arrays:

Lemma 6.1. Let A denote one of the t-arrays M , S , or G . Let j and d be positive integers and let $n(j,d)$ denote the smallest integer $n \ge 1$ for which $A^n(j) > A^{n+d}(j)$, then the following are true:

(a) If $n' \ge n(j,d)$, then $A^{n'}(j) \ge A^{n'+d}(j)$.

(b) If $j' > j$, then $n(j',d) > n(j,d)$.

Proof. It is easily verified that

$$(6.1) \qquad A^{1-t}(0) = \dots = A^0(0) > 0 = A^1(0) = A^2(0) = \dots$$

and that for $j \ge 1$,

$$(6.2) \qquad 0 < A^{-}(j) = \dots = A^0(j) \le A^1(j) \; .$$

It is clear that $n(j,d)$ always exists since $A^n(j)$ is zero for n sufficiently large. From (6.1) it follows that

$$A^1(1) > A^2(1) \ge A^3(1) \ge A^4(1) \ge \dots$$

so that $n(1,1) = 1$ and (a) is true for $n(1,1)$.

We will now show that if (a) is true for $n(j,1)$, then it is true for $n(j,d)$ for $j > 1$ and for $n(j+1,1)$. Let $d > 1$ be given and let $m > 1-t$ be the smallest such integer for which $A^m(j) > A^{m+d}(j)$. It is clear that $m+d > n(j,1)$. We will show that $A^n(j) \geq A^{n+d}(j)$ for $n > m$. This is certainly true if $n \geq n(j,1)$. Also, if $m \leq n < n(j,1)$, then we have

$$A^n(j) \geq A^m(j) > A^{m+d}(j) \geq A^{n+d}(j) .$$

Since $n(j,d) \geq m$, we see that (a) is true for $n(j,d)$. From the recurrence relation for t-arrays, we have

$$A^{n+1}(j+1) - A^n(j+1) = A^n(j) - A^{n-t}(j) .$$

Consequently, if we let $d = t$ in the above argument, we see that we may choose $n(j+1,1) = m+t$ and that (a) is true for this choice. The validity of (a) now follows by induction.

To prove (b), let $j \geq 1$ and let $n = n(j+1,d)$. From the recurrence relation for t-arrays, we have

$$0 > A^{n+d}(j+1) - A^n(j+1) = \sum_{k=1}^{t} (A^{n+d-k}(j) - A^{n-k}(j))$$

so that $A^{n-k}(j) > A^{n+d-k}(j)$ for some k with $1 \leq k \leq t$. If $n-k \geq 1$, then $n-k \geq n(j,d)$ so that $n > n(j,d)$. If $n-k \leq 0$, then we must have $n+d-k > n(j,1)$ so that

$$A^1(j) \geq A^{n-k}(j) > A^{n+d-k}(j) > A^{1+d}(j)$$

and therefore' $n(j,d) = 1 \leq n$. We have therefore shown that $n(j+1,d) \geq n(j,d)$ and (b) follows. □

The lemma is particularly useful in the following form:

Corollary 6.1. Let A denote one of the t-arrays M , S , or G , then the following are true:

(a) If $A^n(j) < A^{n'}(j)$ for some $1 \leq n < n'$ and $j \geq 1$, then $A^n(j') \leq A^{n'}(j')$ for all $j' \geq j$.

(b) If $A^n(j) > A^{n'}(j)$ for some $1 < n < n'$ and $j \geq 1$, then $A^n(j') \geq A^{n'}(j')$ for all j' with $1 \leq j' \leq j$.

Proof. To prove (a) let $d = n'-n$. Certainly $n < n(j,d)$ so it follows that $n < n(j',d)$ for all $j' > j$ and the result follows from the definition of $n(j',d)$. This also proves (b) since (b) is the contrapositive of (a). □

Theorem 6.1. If $n < n'$ and $V^n(x) > V^{n'}(x)$ for some $x \leq S^n$, then there exists a $j < n$ for which $G^n(j) < G^{n'}(j)$. Furthermore, if $x < y \leq S^n$, then $V^n(y) > V^{n'}(y)$.

Proof. Clearly $x > 0$. Let j and k be the unique integers for which

$$S^n(j) < x \leq S^n(j+1) \quad \text{and} \quad S^{n'}(k) < x \leq S^{n'}(k+1)$$

We observe that $j < n$. By assumption

$$(j+1)x - G^n(j) = V^n(x) > V^{n'}(x) = (k+1)x - G^{n'}(k)$$

which reduces to

$$G^n(j) < G^{n'}(k) + (j-k)x \ .$$

In order to prove that $G^n(j) < G^{n'}(j)$ we will show that $(j-k)x \leq G^n(j) - G^{n'}(k)$. If $j = k$, then there is nothing to prove. If $j > k$, then we have

$$(j-k)x \leq \sum_{i=k+1}^{j} S^{n'}(i) = G^{n'}(j) - G^{n'}(k) \ .$$

Similarly, if $j < k$, then

$$(j-k)x = -(k-j)x < -\sum_{i=j+1}^{k} S^{n'}(i) = G^{n'}(j) - G^{n'}(k) .$$

Now suppose that there is a smallest y with $x < y \leq S^n$ for which $V^n(y) \leq V^{n'}(y)$. Let j' and k' be the unique integers for which

$$S^n(j') < y \leq S^n(j'+1) \quad \text{and} \quad S^{n'}(k') < y \leq S^{n'}(k'+1)$$

Since $V^n(y-1) > V^{n'}(y-1)$, we find that

$$j'+1 = V^n(y) - V^n(y-1) < V^{n'}(y) - V^{n'}(y-1) = k'+1$$

from which it follows that $j' < k'$. On the other hand, since $G^n(j) < G^{n'}(j)$, we can find an $m \leq j$ for which $S^n(m) < S^{n'}(m)$. By (a) of Corollary 6.1, we see that $S^n(m') < S^{n'}(m')$ for all $m' \geq m$. Since $j'+1 > j \geq m$, it follows that

$$y \leq S^n(j'+1) \leq S^{n'}(j'+1) \leq S^{n'}(k') < y$$

which is impossible. This completes the proof. □

<u>corollary 6.2</u>. Let $N(x)$ be the smallest integer n which minimizes $V^n(x)$, then $N(x)$ is an increasing function of x .

<u>Proof</u>. Suppose that $N(x) > N(x+1)$ for some x and let $a = N(x)$ and $b = N(x+1)$. Since $b < a$, we must have $V^a(x) < V^b(x)$. Also, since $x+1 \leq S^b < S^a$, it follows from Theorem 6.1 that $V^a(x+1) < V^b(x+1)$ which implies that $N(x+1) \neq b$. □

<u>Remarks</u>. Most of-these results were first proved by Knuth ([5], Exercise 5.4.2-14), however, our proof of Theorem 6.1 is somewhat different. Shell [8] has observed Corollary 6.2 empirically.

In the remainder of this section, we will solve the problem of determining the range of values of x for which $N(x)$ takes a given value. We will begin by examining some of the more subtle properties of the numbers $G^n(j)$.

Lemma 6.2. For each $t \geq 2$, there exists a number n_t with the property that $G^n(j) < G^{n+1}(j)$ for some j with $1 \leq j < n$, if and only if $n \geq n_t$. Inparticular $n_2 = 8$, $n_3 = 5$, $n_4 = 4$, and $n_t = 3$ for $t \geq 5$.

Proof. If $G^n(j) < G^{n+1}(j)$ for some j with $1 \leq j < n$, then we can find a $j' \leq j$ for which $S^n(j') < S^{n+1}(j')$. By (a) of Corollary 6.1 we find that $S^n(k) \leq S^{n+1}(k)$ for $k \geq j \geq j'$ and consequently $G^n(n-1) < G^{n+1}(n-1)$. It follows at once that such a j exists if and only if $G^n(n-1) < G^{n+1}(n-1)$. Furthermore, if this inequality holds for n , it holds for n+1 since, by (a) of Lemma 6.1, we have $G^{n-k}(n-1) < G^{n-k+1}(n-1)$ for $k = 1,\ldots,t$ and it follows from the recurrence relation for t-arrays that

$$G^{n+2}(n) - G^{n+1}(n) = G^{n+1}(n-1) - G^{n-t+1}(n-1) > 0 \ .$$

The following table will serve to verify the values given for n_t :

t	$G^{n_t-1}(n_t-2)$	$G^{n_t}(n_t-2)$	$G^{n_t}(n_t-1)$	$G^{n_t+1}(n_t-1)$
2	58	56	109	114
3	20	20	48	56
4	11	11	32	40
≥ 5	t-1	t-2	4t-5	5t-9

Lemma 6.3 For each $n \geq n_t$, let j_n denote the smallest integer j for which $G^n(j) < G^{n+1}(j)$. We then have

$$j_n \leq j_{n+1} \leq j_n+1 \leq j_{n+t} \ .$$

Proof. First we will show that $j_n \leq j_{n+1}$. Assume that for some $k < j_n$ we have $G^{n+1}(k) < G^{n+2}(k)$. We may write

$$G^{n+1}(k) - G^n(k) = G^n(k-1) - G^{n-t}(k-1)$$

$$= (G^{n+1}(k-1) - G^{n-t+1}(k-1))$$

$$+ (G^n(k-1) - G^{n+1}(k-1))$$

$$+ (G^{n-t+1}(k-j) - G^{n-t}(k-1)) \ .$$

The first parenthesized term is equal to $G^{n+2}(k) - G^{n+1}(k)$ and is therefore positive. The second term is nonnegative since $k < j_n$. Since $G^{n+1}(k) < G^{n+2}(k)$ it follows from the recurrence relation for t-arrays that $G^{n+1-m}(k-1) < G^{n+2-m}(k-1)$ for some m with $1 \leq m \leq t$. From (a) of Lemma 6.1 it follows that $G^{n-t}(k-1) < G^{n-t+1}(k-1)$ so the last parenthesized term is nonnegative. We have therefore shown that $G^n(k) < G^{n+1}(k)$ which contradicts the minimality of j_n .

Since $G^n(j_n) < G^{n+1}(j_n)$, we may show as in the proof of Lemma 6.2 that $G^{n+1}(j_n+1) < G^{n+2}(j_n+1)$ and therefore $j_{n+1} \leq j_n+1$. Finally, since $G^{n+t}(j_{n+t}) < G^{n+t+1}(j_{n+t})$, it follows that $G^{n+t-k}(j_{n+t}-1) < G^{n+t+1-k}(j_{n+t}-1)$ for some k with $1 \leq k \leq t$. Consequently, $j_n \leq j_{n+t-k} \leq j_{n+t}-1$. This completes the proof. ❑

. Lemma 6.4. Define the numbers N_t by $N_2 = 19$, $N_3 = 6$, and $N_t = n_t$ for $t \geq 4$. If $n \geq N_t$ and $j \geq 0$, then

$$2G^n(j) \leq G^n(j+1) + G^{n+1}(j-1) \ .$$

Proof. We will show that the above inequality holds for all but finitely many values of $n \geq 1$ and $j \geq 0$. The condition on n is sufficient to exclude these exceptions. We define the t-array D by

$$D^n(j) = G^n(j+1) + G^{n+1}(j-1) - 2G^n(j) .$$

It is not difficult to verify that the nonzero elements of the initialization region for D are

$$D^0(j) = (t-1)j - t \quad \text{for } j \geq 1 \text{ and}$$

$$D^n(-1) = 1 \quad \text{for } 1-t \leq n \leq 0 .$$

We observe that $D^0(1) = -1$ is the only negative element for the initialization region. Tables 6.1(a), 6.1(b), and 6.1(c) each display a portion of the t-array D for $t \geq 4$, $t = 3$, and $t = 2$, respectively. By inspecting these tables, it is clear that there are no negative values of $D^n(j)$ with $n \geq 0$ other than those displayed. Since the negative entries only appear in the columns for which $n < N_t$, it follows that $D^n(j) \geq 0$ when $n \geq N_t$. □

Theorem 6.2. If $n \geq N_t$ and if we define

$$c_n = G^n(j_n) - G^{n+1}(j_n-1) ,$$

then the following are true:

(a) $S^n(j_n) \leq c_n \leq S^n(j_n+1)$,

(b) $S^{n+1}(j_n-1) \leq c_n < S^{n+1}(j_n)$,

(c) $V^n(c_n) = V^{n+1}(c_n)$,

(d) $V^n(c_n+1) > V^{n+1}(c_n+1)$ if $c_n < S^n$,

(e) $c_n < c_{n+1}$.

Proof. From the definition of j_n we know that $G^n(j_n-1) \geq G^{n+1}(j_n-1)$. We therefore have

$$\begin{aligned} S^n(j_n) &= G^n(j_n) - G^n(j_n-1) \leq G^n(j_n) - G^{n+1}(j_n-1) \\ &= c_n < G^{n+1}(j_n) - G^{n+1}(j_n-1) = S^{n+1}(j_n) \ . \end{aligned}$$

From Lemma 6.4

$$c_n = G^n(j_n) - G^{n+1}(j_n-1) \leq G^n(j_n+1) - G^n(j_n) = S^n(j_n+1) \ .$$

Also from Lemma 6.4,

$$S^{n+1}(j_n-1) = G^{n+1}(j_n-1) - G^{n+1}(j_n-2) \leq G^n(j_n) - G^{n+1}(j_n-1) = c_n \, .$$

This completes the proof of (a) and (b).

From (a) and (b) we have

$$\begin{aligned} V^n(c_n) &= (j_n+1)c_n - G^n(j_n) \\ &= j_n G^n(j_n) - (j_n+1)G^{n+1}(j_n-1) \\ &= j_n c_n - G^{n+1}(j_n-1) = V^{n+1}(c_n) \end{aligned}$$

which is (c). To prove (d) we first observe that from (a) and (b) we have $V^{n+1}(c_n+1) - V^{n+1}(c_n) = j_n$ and $V^n(c_n+1) - V^n(c_n) \geq j_n+1$ if $c_n < S^n$. From (c) it follows that

$$V^n(c_n+1) - V^{n+1}(c_n+1) > 1 + V^n(c_n) - V^{n+1}(c_n) = 1 \ .$$

By Lemma 6.3 we have $j_n \leq j_{n+1}$ so by (a) and (b)

$$c_n < S^{n+1}(j_n) \leq S^{n+1}(j_{n+1}) \leq c_{n+1}$$

which is (e). This completes the proof. ❑

Corollary 6.2. $c_n \to \infty$ as $n \to \infty$.

Proof. This follows from (e) and the fact that c_n is an integer. ❑

For each $t \geq 2$, we define the sequence $L_1, L_2, \ldots$ as follows: If $t \geq 3$, then we let $L_n = S^n$ for $n < N_t$ and $L_n = c_n$ for $n \geq N_t$. If $t = 2$, then we let $L_n = S^n$ for $n \leq 15$, $L_{16} = 2573$, $L_{17} = 3954$, $L_{18} = 6527$, and $L_n = c_n$ for $n \geq N_2 = 19$.

Theorem 6.3. The sequence $L_1, L_2, \ldots$ is strictly increasing and has the property that $V^{n+1}(x) \geq V^n(x)$ if and only if $x \leq L_n$.

Proof. First we will show that the sequence is strictly increasing. We already know that $S^n < S^{n+1}$ for all $n \geq 1$ and that $c_n < c_{n+1}$ for all $n \geq N_t$. These observations leave us with only a few special cases to consider.

When $t \geq 3$, we must show that when $n = N_t - 1$, we have $S^n = L_n < L_{n+1} = c_{n+1}$. When $t \geq 5$ we may show from the appropriate t-arrays that $S^2 = 2t-1$ and $c_3 = 3t-2$ so that $L_n < L_{n+1}$ since $N_t = 3$. When $t = 4$, we have $N_t = 4$ and $L_3 = S^3 = 13 < 22 = c_4 = L_4$. For $t = 3$, we have $N_t = 6$ and $L_5 = S^5 = 31 < 32 = c_6 = L_6$. For the remaining special case $t = 2$, we have $L_{15} = S^{15} = 1597 < 2573 = L_{16}$, $L_{16} < L_{17} < L_{18}$, and $L_{18} = 6527 < 10488 = c_{19} = L_{19}$.

To prove the second part of the theorem, it is sufficient, in view of Theorem 6.1, to show that $V^{n+1}(L_n) > V^n(L_n)$ for all $n \geq 1$ and that $V^{n+1}(L_n+1) < V^n(L_n+1)$ whenever $L_n < S^n$. If $n < n_t$, then $G^n(j) \geq G^{n+1}(j)$ for all j with $0 \leq j < n$, so by Theorem 6.1, we have $V^{n+1}(L_n) \geq V^n(L_n)$. We also note that $L_n = S^n$ for $n < n_t$. When $n \geq N_t$, then everything follows from Theorem 6.2. Since $n_t = N_t$ for $t \geq 4$, this proves the result for $t \geq 4$. To extend the result to the case $t = 3$, we observe that in this case we have $L_5 = S^5$ and $V^5(L_5) = 107 < 108 = V^6(L_5)$.

When $t = 2$, there are a number of special cases to consider. First we note that $L_n = S^n$ for $8 \leq n \leq 15$. By direct computation, we may verify that

$$V^{8}(L_8) = 331 < 343 = V^{9}(L_8),$$

$$V^{9}(L_9) = 600 < 614 = V^{10}(L_9) ,$$

$$V^{10}(L_{10}) = 1075 < 1092 = V^{11}(L_{10}) ,$$

$$V^{11}(L_{11}) = 1908 < 1935 = V^{12}(L_{11}) ,$$

$$V^{12}(L_{12}) = 3360 < 3396 = V^{13}(L_{12}) ,$$

$$V^{13}(L_{13}) = 5878 < 5901 = V^{14}(L_{13}) ,$$

$$V^{14}(L_{14}) = 10225 < 10240 = V^{15}(L_{14}) ,$$

$$V^{15}(L_{15}) = 17700 < 17726 = V^{16}(L_{15}) ,$$

$$V^{16}(L_{16}) = 30342 < 30343 = V^{17}(L_{16}) ,$$

$$V^{17}(L_{17}) = 48950 = V^{18}(L_{17}) ,$$

$$V^{18}(L_{18}) = 85819 < 85820 = V^{19}(L_{18}) .$$

We also have

$$V^{16}(L_{16}+1) = 30357 > 30356 = V^{17}(L_{16}+1) ,$$

$$V^{17}(L_{17}+1) = 48965 > 48963 = V^{18}(L_{17}+1) ,$$

$$V^{18}(L_{18}+1) = 85835 > 85834 = V^{19}(L_{18}+1) ,$$

which completes the proof of the theorem. $\square$

Two consequences of this theorem are easily proved.

Corollary 6.3. $N(x)$ is the smallest integer n for which $x \leq L_n$.

Coronary 6.4. If $V^n(x) \leq V^{n+1}(x)$, then $V^{n'}(x) < V^{n'+1}(x)$ for all $n' \geq n$.

Remark. Corollary 6.4 answers in the affirmative a conjecture of Knuth ([5], Exercise 5.4.2-15).

Table 6.2 provides the values of L_n for $t = 2,\ldots,7$ and $n = 1,\ldots,19$. Since such a table is easily prepared, we are able to provide a very simple dispersion algorithm.

Algorithm 6.1. Optimal Polyphase Sort for x Strings.

Step 1. Find the smallest n for which $x \leq L_n$.

Step 2. Choose a j for which $S^n(j) \leq x \leq S^n(j+1)$.

Step 3. Find integers $x_1,\ldots,x_t$ for which $x = x_1 + \ldots + x_t$ and $S_i^n(j) \leq x_i \leq S_i^n(j+1)$ for $i = 1,\ldots,t$.

Step 4. For each $i = 1,\ldots,t$ write x_i strings to tape i .

Step 5. Use Algorithm 4.3 to perform the polyphase merge on the distribution $x_1,\ldots,x_t$ starting at stage n .

Remarks. Since Steps 2 and 3 of the above algorithm and Steps 1 and 2 of Algorithm 4.3 both require tables of the numbers $M_i^n(j)$, some of the operations of these steps can be combined. The above algorithm should be compared with Shell's optimum dispersion algorithm [8] which is directed by a table of numbers closely related to the numbers L_n . The functions $V_i^n(x)$ share many of the properties of the function $V^n(x)$ and most of the results of this section can be carried over to these functions. Unfortunately, the analogues of the numbers j_n are in general different for each i ; otherwise the next section would not have to have been written.

7. Blind Dispersion.

In practice, it is very difficult to predict the number of strings that a dispersion routine will provide. However, Algorithm 6.1 requires that this number be known before the strings are written to the tapes. This brings us to the problem of blind dispersion, that is, dispersion without knowing the number of strings in advance.

We begin by observing that no solution to the blind dispersion problem will in general be optimal. Indeed, solutions which require rearranging the contents of the tapes will require additional string motion which will result in a solution which is at best optimal. Therefore, let us consider a solution in which the strings stay on the tapes once they are written. Let us suppose that $t = 2$ and that we have dispersed $S^{10} = 144$ strings optimally. Since $N(144) = 10$ and $V^{10}(144) = 1075 < 1088 = V^{11}(144)$, it is clear that the only optimal distribution is for stage 10 when there are $S_1^{10} = 55$ strings on tape one and $S_2^{10} = 89$ strings on tape two. Let us see what happens when we add another string. Since $N(145) = 11$ and $V^{11}(145) = 1100 < 1143 = V^{12}(145)$, the best distribution of 145 strings is one which is optimal for stage 11. However, since $S_1^{10} > 52 = S_1^{11}(8)+1$ and $S_2^{10} < 96 = S_2^{11}(8)-1$, we see that there is no way of arriving at a distribution which is optimal for stage 11 by adding one string to our original distribution. This pathology was first discovered by D. E. Knuth.

It is not difficult to see that any blind dispersion technique which rearranges the contents of the tapes can be transformed into an equivalent (or perhaps better) method in which the rearranging is performed after all of the strings have been dispersed. The effectiveness of such a technique depends on how close the distribution, prior to rearranging, is to an optimal distribution. We remark that one kind of rearrangement which incurs no extra cost is that of renumbering the tape units. Theorem 5.3 shows that a monotone distribution provides the best renumbering possible. However, since the distributions which we will consider will be monotone or can be made monotone, we will have no use for this technique.

In the remainder of this section, we will construct a nearly optimal blind dispersion technique which requires no tape rearrangement. This dispersion technique can be used by itself or in conjunction with some rearrangement algorithm.

Suppose that $n \geq N_t$. We define m(n) to be the largest integer m for which $j_m = j_n$. From Lemma 6.3 we see that $m(n) < n+t$ and that $j_m = j_n$ for $n \leq m \leq m(n)$. For $i = 1,\ldots,t$ we define

$$B_i^n = \min\{S_i^m(j_n) \mid n \leq m \leq m(n)+1\}$$

and

$$B^n = B_1^n + \ldots + B_t^n .$$

<u>Theorem 7.1</u>. For $n \geq N_t$ we have

(a) $B_i^n \leq B_i^{n+1}$ for $1 \leq i \leq t$;

(b) $B_1^n \leq B_2^n < \ldots \leq B_t^n$;

(c) $S_i^n(j_n-1) \leq B_i^n \leq S_i^n(j_n)$ for $1 \leq i \leq t$;

(a) $S_i^{n+1}(j_n-1) \leq B_i^n \leq S_i^{n+1}(j_n)$ for $1 \leq i \leq t$;

(e) $B^n \leq c_n < B^{n+t}$.

<u>Remark</u>. Statements (c) and (d) imply that the distribution $B_1^n,\ldots,B_t^n$ is optimal for both stage n and stage n+1 .

Before we prove the theorem we require a lemma:

Lemma 7.1. If $n \geq N_t$, then for $i = 1,\ldots,t$ we have

$$S_i^{n+1}(j_n-1) \leq S_i^n(j_n) \quad .$$

Proof. We will begin by showing that for $n > 1$ we have

$$(7.1) \qquad S_i^n(j) - S_i^{n+1}(j-1) > G^n(j-1) - G^{n+1}(j-1)$$

with only finitely many exceptions. We define the t-arrays A_i for $i = 1,\ldots,t$ and D by

$$A_i^n(j) = S_i^n(j) - S_i^{n+1}(j-1)$$

$$D^n(j) = G^n(j-1) - G^{n+1}(j-1) \quad .$$

It is not difficult to verify that the nonzero elements of the initialization regions for the t-arrays $A_1,\ldots,A_t$ are

$$A_i^{i-t}(j) = 1 \qquad \text{for } 1 < i < t \text{ and } j \geq 0 ,$$

$$A^{i-t-1}(j) = -1 \qquad \text{for } 1 < i < t \text{ and } j \geq 1 ,$$

$$A_i^0(j) = -1 \qquad \text{for } 1 < i < t \text{ and } j \geq 2 , \text{ and}$$

$$A_t^0(0) = A;(1) = 1 \quad .$$

Also, the nonzero elements of the initialization region for the t-array D are

$$D^0(j) = t - (t-1)j \qquad \text{for } j \geq 1 .$$

Tables 7.1(a) to 7.1(g) each display portions of the t-arrays A_i-D for various ranges of t and i . By inspection, we see that the only negative entries outside of the initialization region are those displayed. Except in the case $t = i = 2$, we see that (7.1) holds for all $n \geq N_t$

and $i = 1,\ldots,t$. From the definition of j_n , we see that $G^n(j_n-1) \geq G^{n+1}(j_n-1)$ and therefore by (7.1) we have $S_i^n(j_n) \geq S_i^{n+1}(j_n-1)$. In the exceptional case we have $n = N_t = 19$ and $j_{19} = 15$ so we may verify directly that $S_2^{19}(15) = 6050 > 5270 = S_2^{20}(14)$. This completes the proof. □

Proof. of Theorem 7.1. If $j_n = j_{n+1}$, then $m(n) = m(n+1)$ so that (a) is obvious. If this is not the case, then by Lemma 6.3, we must have $j_{n+1} = j_n+1$ so that $S_i^{n+1}(j_n) \leq S_i^{n+1}(j_{n+1})$. Also, since $m(n+1) \leq n+t$, we see that $S_i^{n+1}(j_n) \leq S_i^k(j_{n+1})$ for $n+2 < k < m(n+1)+1$; this follows from the fact that $S_i^{n+1}(j_n)$ is a term of the t-sum which computes $S_i^k(j_{n+1})$. We have therefore shown that $B_i^n \leq S_i^{n+1}(j_n) \leq B_i^{n+1}$, which is (a). Statement (b) follows at once from the fact that $S_1^n(j) \leq \ldots \leq S_t^n(j)$ for all $n \geq 1$ and $j \geq 1$.

To prove (c) and (d) we first observe that the definition of B_i^n implies that $B_i^n \leq S_i^n(j_n)$ and $B_i^n \leq S_i^{n+1}(j_n)$. It is also clear that $S_i^n(j_n-1) \leq S_i^n(j_n)$ and $S_i^{n+1}(j_n-1) \leq S_i^{n+1}(j_n)$. From Lemma 7.1, we have $S_i^{n+1}(j_n-1) \leq S_i^n(j_n)$. Finally, by reasoning similar to that used in the above paragraph, we have $S_i^n(j_n-1) \leq S_i^k(j_n)$ for $n+1 < k \leq m(n+1)$ and $S_i^{n+1}(j_n-1) \leq S_i^k(j_n)$ for $n+2 < k \leq m(n)+1$ if $m(n) > n$. From these inequalities, it follows at once that $S_i^n(j_n-1) \leq B_i^n$ and that $S_i^{n+1}(j_n-1) \leq B_i^n$ which completes the proof of (c) and (d).

By (c) we have $B^n \leq S^n(j_n) \leq c_n$. If we let $n' = m(n)+1$, then it is clear that $j_{n'} = j_n+1$, $j_{n'-1} = j_n$, and $n' < n+t$. Therefore, by

(a) and (c) we have

$$c_n \le c_{n'-1} < S^{n'}(j_{n'-1}) = S^{n'}(j_{n'}-1) < B^{n'} \le B^{n+t}$$

which establishes (e) and completes the proof of the theorem. □

From the theorem, two important properties of the distributions $B_1^n, \ldots, B_t^n$ are apparent. First, we may arrive at the distribution $B_1^{n+1}, \ldots, B_t^{n+1}$ by simply adding strings to the distribution $B_1^n, \ldots, B_t^n$. Second, if we are dispersing for stage n and we reach the distribution $B_1^n, \ldots, B_t^n$, then we may begin dispersing for stage n+1 since the distribution is optimal for both stages. Clearly we can base a blind dispersion algorithm on these properties of the numbers B_i^n. However, since we will be making several refinements, it is of value to examine the general structure of such an algorithm.

We define a <u>quota scheme</u> for polyphase dispersion to be a family of nonnegative integers $Q^n, Q_1^n, \ldots, Q_t^n$, $n = 1,2,\ldots$ which have the following properties fom ≥ 1 and $1 \le i \le t$:

$$Q_i^n \le S_i^n \ , \quad Q_i^n \le Q_i^{n+1} \ , \quad Q^n \le Q^{n+1} \quad \text{I}$$

$$Q^n \le Q_1^n + \ldots + Q_t^n \ , \text{ and } Q^n \to \infty \text{ as } n \to \infty \ .$$

Following is the dispersion algorithm which is directed by the quota scheme. The counters $x_1, \ldots, x_t$ contain the numbers of strings which have been'written to tapes $1,\ldots,t$. Upon completion, the values of $x_1, \ldots, x_t$ and n are the parameters for initializing Algorithm 4.3.

Algorithm 7.1 Quota-Directed Polyphase Dispersion,

Step 1. Let $n = 1$ and $x_i = y_i = 0$ for $i = 1,\ldots,t$.

Step 2. If there are no more strings to disperse, then terminate the algorithm.

Step 3. If $x_1 + \ldots + x_t = Q^n$, then let $n = n+1$ and $y_1 = \cdots = y_t = 0$ and repeat this step.

Step 4. Choose some i for which $x_i < y_i$. If this choice can not be made, then go to Step 6.

Step 5. Write a string to tape unit i , let $x_i = x_i+1$, and go to Step 2.

Step 6. Find the smallest j for which $x_i < S_i^n(j)$ for some i . Let $y_i = \min(Q_i^n, S_i^n(j))$ for $i = 1, \ldots, t$. Go to Step 4.

Informally, this algorithm disperses for stage n keeping $x_i \leq Q_i^n$ for each i until $x_1 + \ldots + x_t = Q^n$ and then begins dispersing for stage $n+1$. When the algorithm is dispersing for stage n , the strings are written in such a way as to minimize the growth of $V_1^n(x_1) + \ldots + V_t^n(x_t)$.

Since the choice of i made in Step 4 is arbitrary, the distribution $x_1,\ldots,x_t$ may be uncertain when the algorithm switches from stage n to stage $n+1$. For this reason, the first value of j chosen for stage $n+1$ by Step 6 may vary thereby causing the volume of the sort to vary. This uncertainty disappears if $Q^n = Q_1^n + \ldots + Q_t^n$ or if it is known that when we switch from stage n to stage $n+1$, then the distribution is optimal for stage $n+1$. Indeed, in the first case the distribution is completely known and in the second case we know that j is the smallest integer for which $x_1 + \ldots + x_t < S^{n+1}(j)$. The quota scheme which we will consider has one or the other of these properties for each n .

When the quota scheme has these properties, then Algorithm 7.1 may be transformed into a simpler table-directed algorithm. The tables have the entries $n^k, q^k, q_1^k, \ldots, q_t^k$ for $k > 1$ and are constructed as follows: We initialize the counter k to zero and perform Algorithm 7.1 with an unlimited supply of strings; after each time that Step 6 is performed, we increment k by one and let $n^k = n$, $q^k = Q^n$, and $q_i^k = y_i$ for each i . The simplified algorithm follows:

<u>Algorithm 7.2</u> Simplified Quota-Directed Polyphase Dispersion.

Step 1. Let $k = 1$ and $x_1 = \ldots = x_t = 0$.

Step 2. If there are no more strings to disperse, then terminate the algorithm.

Step 3. If $x_1 + \ldots + x_t = q^k$, then let $k = k+1$.

Step 4. Choose some i for which $x_i < q_i^k$. If this choice can not be made, then let $k = k+1$ and go to Step 3.

Step 5. Write a string to unit i , let $x_i = x_i+1$, and go to Step 2.

At termination, the parameters for the polyphase merge algorithm are n^k and $x_1, \ldots, x_t$. Since the required tables may be prepared in advance, this algorithm provides a very compact method of dispersing for the polyphase sort. For most applications, the maximum value of k should never exceed forty.

We will now present the rules for constructing the quota scheme for the-blind polyphase dispersion algorithm.

1. If $n \geq N_t$ and if $B_i^n < S_i^n(j_n)$ for some i , then we let

$$Q^n = B^n \quad \text{and} \quad Q_i^n = B_i^n \quad \text{for } i = 1,\ldots,t \ .$$

2. If $n \geq N_t$ and if $B_i^n = S_i^n(j_n)$ for each i , then we let

$$Q_i^n = \min(S_i^n(j_n+1) \ , \ S_i^{n+1}(j_n) \ , \ B_i^{n+1})$$

for $i = 1,\ldots,t$ and we let

$$Q^n = \min(c_n , Q_1^n + \ldots + Q_t^n) \ .$$

3. If $t \geq 3$ and $1 \leq n < N_t$, then we let

$$Q^n = S^n \quad \text{and} \quad Q_i^n = S_i^n \quad \text{for } i = 1,\ldots,t \ .$$

4. If $t = 2$ and $n < N_t = 19$ then we let

$$Q^n = S^n \quad \text{and} \quad Q_i^n = S_i^n \quad \text{for } n \leq 15 \text{ and } i = 1,\ldots,t$$

and, in addition, we let

$$Q^{16} = L_{16} = 2573 \ , \ Q^{17} = 3845 \ , \ Q^{18} = L_{18} = 6527 \ ,$$

$$Q_1^{16} = S_1^{16}(15) = 986 \ , \ Q_2^{16} = S_2^{16}(15) = 1596 \ ,$$

$$Q_1^{17} = S_1^{18}(13) = 1383 \ , \ Q_2^{17} = S_2^{17}(14) = 2462 \ ,$$

$$Q_1^{18} = S_1^{18}(16) = 2567 \ , \ Q_2^{18} = S_2^{18}(16) = 4163 \ .$$

. To show that these rules define a quota scheme, we will being by showing that for $n \geq N_t$, we have

$$B^n \leq Q^n \leq B^{n+1} \quad \text{and} \quad B_i^n \leq Q_i^n \leq B_i^{n+1} \quad \text{for } i = 1,\ldots,t.$$

These relations are obvious when Rule 1 is applied. If Rule 2 is applied instead, we have, from Theorems 6.2 and 7.1,

$$B_i^n = S_i^n(j_n) \le Q_i^n \le B_i^{n+1} \qquad \text{for } i = 1,\dots,t \text{ and}$$

$$B^n = S^n(j_n) \le Q^n \le Q_1^n + \dots + Q_t^n < B^{n+1} .$$

For $t \ge 3$, we must show that when $n = N_t - 1$, that we have $S^n \le Q^{n+1}$ and $S_i^n \le Q_i^{n+1}$ for $i = 1,\dots,t$. Clearly, it is sufficient to show that $S_i^n \le B_i^n$ for each i. For $t = 3$, we have $N_t = 6$ and

$$S_1^5 = 7 < 12 = B_1^6 , \; S_2^5 = 11 < 19 = B_2^6 , \; S_3^5 = 13 < 27 = B_3^6 .$$

Similarly, for $t = 4$ we have $N_t = 4$ and

$$S_1^3 = 2 < 3 = B_1^4 , \quad S_2^3 = 3 < 5 = Q_2^4 ,$$

$$S_3^3 = 4 < 6 = B_3^4 , \quad S_4^3 = 4 < 7 = Q_4^4 .$$

For $t \ge 5$, we have $N_t = 3$ and using the t-array representation, it may be shown that $G^n(2) = 2t + (n-1)(t-1-n/2)$ for $1 \le n \le t$ from which it follows that $G^3(2) < \dots < G^{t-1}(2) = G^t(2)$ so that $m(3) = t-2$ since $j_3 = 2$. Using t-arrays, we may also show that $S_i^2(2) < \dots \le S_i^t(2)$ for each i so that $S_i^2(2) \le S_i^3(2) = B_i^3$ for each i. The proof that Rule 4 also contributes to a quota scheme is straightforward once we observe that when $t = 2$ we have

$$S_1^{15} = 610 , \; S_2^{15} = 987 , \; B_1^{19} = 3588 , \; B_2^{19} = 6050 .$$

We have already seen how the numbers B^n and $B_1^n,\dots,B_t^n$ can be used to describe blind polyphase dispersion so Rule 1 requires no explanation. Rule 2 represents a refinement in which Q^n is pushed to the largest value not exceeding c_n for which we can switch from stage n to stage $n+1$ with a distribution which is optimal for both stages. Rule 2 will be used for each n for which $j_{n+1} = j_n + 1$. Rules 3 and 4 simply fill out the quota scheme for small values of n.

The special assignments in Rule 4 were chosen subjectively to insure reasonably good performance.

If we are dispersing using the quota scheme just described, it is clear that when the number of strings x is large, then we will switch stages with a distribution which is optimal for both stages. Consequently, the volume of the sort will be $V^{N'(x)}(x)$ for some integer $N'(x)$ when x is sufficiently large. Since $Q^n \leq c_n$ for $n \geq N_t$, it is clear that $N'(x) \geq$ [illegible]. On the other hand, since $c_n < B^{n+t} \leq Q^{n+t}$, we see that $N'(x) < N(x)+t$.

The blind polyphase sort which we have described is almost as good as the optimal polyphase sort of Algorithm 6.1, when the number of strings is in the range of the size of most applications (say, less than a thousand), the two sorts are almost always equivalent. When the number of strings is large, it can be shown that the two algorithms are equivalent infintely often. Indeed, this happens for $S^n(j_n)$ strings every time that $j_{n+1} = j_n+1$. In the next section we will show that the two algorithms are also asymptotically equivalent.

Example 7.1. Table 7.2 displays a portion of the simplified quota scheme for the case $t = 4$.

8. Asymptotic Performance.

In this section, we will study the performance of the algorithms which we have described when the number of strings is large. There are two volumes which we are interested in estimating. First there is the volume of the optimal polyphase sort of Section 6,

$$V(x) = V^{N(x)}(x) ,$$

and, second, there is the volume of the blind polyphase sort, when x is large, this is

$$V'(x) = V^{N'(x)}(x) .$$

We will show that when x is large that both of these volumes are asymptotically equal to

$$x \log_t x + \frac{1}{2} x \log_t \log_t x + o(x) .$$

The reader who is not familiar with asymptotic methods may find [1] or the first chapter of [4] to be helpful.

Our startingpoint is an interesting connection between the movement numbers and the theory of probability. Let $y_1, y_2, \ldots$ be independent random variables which each take on the values $1,2,\ldots,t$ with equal probability t^{-1}. Simple calculations will show that each y_i has an expectation $\mu = (t+1)/2$ and a variance $\sigma^2 = (t^2-1)/12$. For positive integers m and k we define

$$(8.1) \qquad p(m,k) = \operatorname{prob}(y_1 + \ldots + y_k = m) .$$

<u>Lemma 8.1</u>. For $n \geq 1$ and $j > 1$, we have $M_t^n(j) = t^j p(n,j)$.

<u>Proof</u>. Let $q(z) = z + z^2 + \ldots + z^t$ for the real variable z. We will begin by showing that

$$(8.2) \qquad \sum_{n \geq 1} M_t^n(j) z^n = q(z)^j \ .$$

Since the only **nonzero** values of $M_t^n(1)$ **are** $M_t^n(1) = 1$ when $1 < n < t$, we see that (8.2) is true when $j = 1$. **Furthermore,** given (8.2) and the fact that $M_t^{n-k}(j) = 0$ when $n \leq k$, we may write

$$\begin{aligned}
\sum_{n > 1} M_t^n(j+1) z^n &= \sum_{n \geq 1} \sum_{k=1}^{t} M_t^{n-k}(j) z^n \\
&= \sum_{k=1}^{t} z^k \sum_{n \geq 1} M_t^{n-k}(j) z^{n-k} \\
&= \sum_{k=1}^{t} z^k q(z)^j = q(z)^{j+1}
\end{aligned}$$

so that (8.2) follows by induction. From the form of $q(z)$, we see that the coefficient of z^n in $q(z)^j$ is precisely the number of ways that n may be written as the ordered sum of j integers, not necessarily distinct, chosen from the set $\{1,2,\ldots,t\}$. Since this number is precisely $t^j p(n,j)$, the proof is complete.

Techniques for estimating probabilities of the form (8.1) are well known. For our purposes, the best such approximation follows from a theorem of C. G. Esseen which is given on page 241 of [3]:

$$p(m,k) = \frac{e^{-s^2/2}}{\sigma\sqrt{2\pi}} \left(\frac{1}{k^{1/2}} + \frac{Q_1(s)}{k} + \frac{Q_2(s)}{k^{3/2}} + \frac{Q_3(s)}{k^2} + \frac{Q_4(s)}{k^{5/2}} \right) + o\left(\frac{1}{k^{5/2}} \right)$$

where we have written $s = (m - \mu k)/\sigma\sqrt{k}$ **and where** $Q_1(s)$, $Q_2(s)$, $Q_3(s)$, $Q_4(s)$ are polynomials in which the coefficients depend only on the moments of y_i which, in turn, depend only on t . It turns out that all of the centralized moments of y_i of odd order are zero. This leads to some simplifications; in **particular** $Q_1(s) = Q_3(s) = 0$ and $Q_2(s)$ takes on the simplified form $c(s^4 - 6s^2 + 3)$ in which c depends only on t . Using the estimates

$$e^{-s^2/2} = 1 - \frac{1}{2} s^2 + O(s^4) \; ,$$

$$Q_2(s) = 3c + O(s^2 + s^4) \; ,$$

$$s^2/2 \; Q_4(s) = O(1) \; ,$$

we obtain the approximation

$$p(m,k) = \frac{1}{\sigma\sqrt{2\pi}} \left(\frac{1}{k^{1/2}} (1 - s^2/2) + \frac{3c}{k^{3/2}} \right) + O\left(\frac{1}{k^{5/2}} + \frac{s^4}{k^{1/2}} + \frac{s^8}{k^{3/2}} \right)$$

which may be written in the form

$$p(m,k) = \frac{1}{\sigma\sqrt{2\pi}} \left(\frac{1}{k^{1/2}} + \frac{1}{k^{3/2}} \left(3c - \frac{(m - \mu k)^2}{2\sigma^2} \right) \right) + O\left(\frac{1 + (m - \mu k)^8}{k^{5/2}} \right) .$$

To simplify subsequent calculations, we will use the symbol $\mathcal{P}_n(z)$ to represent generically an n-th degree polynomial in z in which the coefficient of z^n is positive and in which all of the coefficients are functions only of t . Two distinct appearances of the symbol in the text need not represent the same polynomial. With this convention, we now have

$$(8.3) \qquad p(m,k) = \frac{1}{\sigma\sqrt{2\pi k}} - \frac{1}{k^{3/2}} \mathcal{P}_2(m - \mu k) + O\left(\frac{1 + (m - \mu k)^8}{k^{5/2}} \right) .$$

<u>Lemma 8.2</u>. We have

$$(8.4) \qquad \sqrt{j}\, t^{-j} M^n(j) = \frac{\mu}{\sigma\sqrt{2\pi}} - \frac{1}{j} \mathcal{P}_2(n - \mu j) + O\left(\frac{1 + (n - \mu j)^8}{j^2} \right) .$$

Proof. From the last, two formulas of (3 .1), we have

$$M_i^n(j) = M_t^{n-1}(j-1) + \dots + M_t^{n-i}(j-1)$$

so that

$$M^n(j) = tM_t^{n-1}(j-1) + (t-1)M_t^{n-2}(j-1) + \dots + M_t^{n-t}(j-1)$$

Therefore, by Lemma 8.1, (8.3), and the facts that

$$\frac{1}{(j-1)^{1/2}} = \frac{1}{j^{1/2}} + \frac{1}{2j^{3/2}} + O\left(\frac{1}{j^{5/2}}\right) ,$$

$$\frac{1}{(j-1)^{3/2}} = \frac{1}{j^{3/2}} + O\left(\frac{1}{j^{5/2}}\right) ,$$

we have

$$\begin{aligned} t^{-j}M^n(j) &= \sum_{i=1}^{t} (t-i+1)t^{-j} M_t^{n-i}(j-1) \\ &= t^{-1} \sum_{i=1}^{t} (t-i+1)p(n-i,j-1) \\ &= \frac{t^{-1}}{\sigma\sqrt{2\pi j}} \sum_{i=1}^{t} (t-i+1) - \sum_{i=1}^{t} \frac{(t-i+1)}{j^{3/2}} \mathcal{P}_2(n-\mu j+i+\mu) \\ &\qquad + O\left(\frac{1+(n-\mu j)^8}{j^{5/2}}\right) \end{aligned}$$

which is easily reduced to the form (8.4). □

Lemma 8.3. We have

$$(8.5) \qquad \sqrt{j}\ t^{-j}G^n(j) = \frac{\mu}{\sigma\sqrt{2\pi}} \frac{t^2}{(t-1)^2} - \frac{\mathcal{P}_2(n-\mu j)}{j} + O\left(\frac{1+(n-\mu j)^8}{j^2}\right) .$$

Proof. From our definitions we have

$$G^n(j) = \sum_{k=1}^{j} S^n(j) = \sum_{k=1}^{j} \sum_{i=1}^{k} M^n(i)$$

$$= \sum_{k=1}^{j} (j-k+1)M^n(k) = \sum_{k=0}^{j-1} (k+1)M^n(j-k) \quad .$$

Using this formula and Lemma 8.2, we obtain

$$t^{-j}G^n(j) = \sum_{k=0}^{j-1} t^{-j}(k+1)M^n(j-k)$$

$$= \sum_{k=0}^{j-1} t^{-k}(k+1)\left(\frac{\mu}{\sigma\sqrt{2\pi}} \quad \frac{1}{(j-k)^{1/2}} \quad \frac{P_2(n-\mu j+\mu k)}{(j-k)^{3/2}}\right)$$

$$+ O\left(\sum_{k=0}^{j-1} t^{-k}(k+1) \frac{1+(n-\mu j+\mu k)^8}{(j-k)^{5/2}}\right) \quad .$$

In order to simplify the above approximation, we need to be able to estimate sums of the form

$$S(a,b) = \sum_{k=0}^{j-1} \frac{k^a}{(j-k)^b} t^{-k}$$

for. $a = 0,1,\ldots,9$ and $b = 1/2, 3/2, 5/2$. Let us write $m = \lfloor\sqrt{j}\rfloor - 1$. If j is sufficiently large, then we will have $k^a < (3/2)^k$ for all $k > m$ and $a = 0,1,\ldots,11$. From the binomial expansion, it is clear that for $k < m$, we have

$$\frac{1}{(j-k)^b} = \frac{1}{j^b} + \frac{bk}{j^{b+1}} + O\left(\frac{k^2}{j^{b+2}}\right)$$

It is also clear that

$$\sum_{k=0}^{m} k^a t^{-k} = s(a) + O((3/2t)^m)$$

where

$$s(a) = \sum_{k=0}^{\infty} k^a t^{-k} \quad .$$

We therefore have

$$s(a,b) = \sum_{k=0}^{m} \left(\frac{k^a}{j^b} + \frac{b\ k^{a+1}}{j^{b+1}} \right) t^{-k} + \sum_{k=m+1}^{j-1} \frac{k^a}{(j-k)^b} t^{-k}$$

$$+ O\left(\sum_{k=0}^{m} \frac{k^{a+2}}{j^{b+2}} t^{-k} \right)$$

$$= \frac{s(a)}{j^b} + \frac{b\ s(a+1)}{j^{b+1}} + O\left((3/2t)^m + \frac{1}{j^{b+1}} \right)$$

since the second sum is $O((3/2t)^m)$ and the last sum is $O(1/j^{b+2})$. Since $(3/2t)^m$ is $O(1/j^{b+2})$ for each b , we may replace the above error estimate by $O(1/j^{b+2})$. The conclusion of this lemma now follows from the facts that

$$s(0) = t/(t-1) \quad \text{and} \quad s(1) = t/(t-1)^2 \quad . \qquad \square$$

<u>Corollary 8.1</u>. If $n - \mu j = O(1)$, then

$$\sqrt{j}\ \ t^{-j} G^n(j) = \frac{\mu}{\sigma\sqrt{2\pi}} \frac{t^3}{(t-1)^2} + O\left(\frac{1}{\ } \right) \quad .$$

<u>Proof</u>. This is a simple consequence of the lemma. $\square$

<u>Corollary 8.2</u>. Let j_n be defined as in Section 6, we then have $n - \mu j_n = O(1)$.

<u>Proof</u>. We recall that j_n is the smallest integer j for which $G^n(j) < G^{n+1}(j)$. From the estimate (8.5), we obtain

$$\sqrt{j}\ t^{-j}(G^{n+1}(j) - G''(j)) = \frac{\rho_1(n - \mu j)}{j} + O\left(\frac{1 + (n - \mu j)^8}{j^2} \right) \quad .$$

With ρ_1 understood to be the ρ_1 appearing above, let a be the real number which satisfies $\rho_1(n-\mu a) = 0$. If $j \geq \lceil a \rceil + 1$, then clearly $\rho_1(n-\mu j)$ is less than some negative quantity which is independent of n and if $j \leq \lfloor a \rfloor - 1$, then $\rho_1(n-\mu j)$ is larger than some positive quantity which is independent of n. For j in the range $\lfloor a \rfloor - 1 \leq j \leq \lceil a \rceil + 1$, we have $n - \mu j = o(1)$ and the error estimate above becomes $O(j^{-2})$ so the first term of the estimate dominates. It follows that j_n lies within this range for large n and the proof is complete. □

<u>Theorem 8.1</u>. $V(x) = x \log_t x + \frac{1}{2} x \log_t \log_t x + O(x)$.

<u>Proof</u>. Let $c_n = G^n(j_n) - G^{n+1}(j_n - 1)$ be defined as in Section 6. From Corollarys 8.1 and 8.2 it is easily shown that

$$(8.6) \qquad \sqrt{j_n}\, t^{-j_n} c_n = \frac{\mu}{\sigma\sqrt{2\pi}} \frac{t}{t-1} + O(j_n^{-1}) .$$

If x is sufficiently large, then for $n = N(x)$ we have $c_{n-1} < x \leq c_n$. Let j be the unique integer for which $S^n(j) < x \leq S^n(j+1)$. From Theorem 6.2, it is clear that $j_{n-1} - 1 \leq j \leq j_n$. Using the formula

$$V(x) = (j+1)x - G^n(j)$$

we may write

$$(8.7) \qquad V(x) - x \log_t x = x(j+1-\log_t x) - G^n(j) .$$

From (8.6) we have

$$\log_t c_n = j_n - \frac{1}{2}\log_t j_n + O(1)$$

and therefore, since $0 \leq j_n - j_{n-1} \leq 1$,

$$j+1-\log_t x < j+1-\log_t c_{n-1}$$

$$= j+1-j_n + \frac{1}{2}\log_t j_{n-1} + O(1)$$

$$\leq j_n+1-j_n + \frac{1}{2}\log_t j_n + O(1)$$

$$= \frac{1}{2}\log_t j_n + O(1) \; .$$

Similarly

$$j+1-\log_t x \geq j+1-\log_t c_n = \frac{1}{2}\log_t j_n + O(1)$$

and we have shown that

$$j+1-\log_t x = \frac{1}{2}\log_t j_n + O(1) \tag{8.8}$$

comparing Corollary 8.1 and (8.6) we see that

$$G^n(j) = O(c_{n-1}) = O(x) \; .$$

From (8.8) and the fact that $j_n - 2 < j \leq j_n$, we have

$$j_n^{-1}\log_t x = 1 + O(j_n^{-1}\log_t j_n)$$

so that

$$\log_t j_n - \log_t \log_t x = O(j_n^{-1}\log_t j_n) = O(1) \; .$$

Putting everything together, (8.7) becomes

$$V(x) - x\log_t x = \frac{-1}{2}x\log_t j_n + O(x)$$

$$= \frac{1}{2}x\log_t \log_t x + o(x)$$

and the proof is complete. $\square$

<u>Corollary 8.3</u>. $V'(x) = x \log_t x + \frac{1}{2} x \log_t \log_t x + o(x)$

<u>Proof</u>. In Section 7 we showed that $N(x) < N'(x) < N(x)+t$. From Theorem 5.3, it follows that

$$0 \leq V'(x) - V(x) = V^{N'(x)}(x) - V^{N(x)}(x) \leq (N'(x) - N(x))x \leq (t-1)x$$

so that $V'(x) - V(x) = O(x)$ and the result follows from the theorem. □

It is well known (see Section 5.4.4 of [5]) that the best possible volume for a merge sort which performs p-way merges is $x \log_p x + O(x)$. For this reason, a tape sort with T tape units has an optimum volue of $x \log_{T-1} x + o(x)$ since such a sort can perform at most T-1 -way merges. A tape sort with T tape units which has a volume asymptotic to $x \log_{T-1} x$ is said to be <u>asymptotically optimal</u>. Theorem 8.1 and Corollary 8.3 imply that both the optimal polyphase sort and the blind polyphase sort are asymptotically optimal.

<u>Remarks</u>. The optimal polyphase sort appears to be the first known example of an asymptotically optimal read forward tape sort. Other examples will appear in [9]. Several asymptotically optimal read backward sorts are known (see, for example, Section 5.4.4 of [5]) but these sorts have volumes of the form $x \log_{T-1} x + O(x)$ which is smaller than the volume we have derived for the optimal polyphase sort. One wonders if the volume $x \log_t x + \frac{1}{2} x \log_t \log_t x + o(x)$ can be improved upon for read forward sorts or whether it represents some theoretical minimum. A simplified self-contained analysis of the optimal polyphase sort, which is probably suitable for students, appears in [10].

9. Concluding Remarks.

Two questions concerning the optimal polyphase sort remain open for investigation. First there is the problem of estimating the amount of time the algorithm spends waiting for tapes to rewind and second there is the problem of optimizing the read backward polyphase sort.

The rewind time is significant since both the blind and the optimal polyphase sorts perform large numbers of tape rewind operations. Of course we may suppose that the total amount of rewinding corresponds to the volume 'of information moved. However, the polyphase merge rewinds two tapes simultaneously so it is conceivable that a highly unbalanced situation may arise in which one of the two tapes being rewound would be considerably longer than the other. This might cause the total rewind wait time to vary from the volume of the merge to twice that volume.

In the read backward polyphase sort, the tape units act as stacks so the direction in which a string is written is reversed when the string is moved. Therefore, strings which will be moved an odd number of times must be written in the opposite direction from strings which will. be written an even number of times. For this reason, strings are no longer interchangable so the dispersion routine must concern itself with the details of placing the dummy strings.

In this paper, we have limited the discussion to the traditional polyphase merge in which the appointment of the output tapes is cyclic. The polyphase merge, however, is just a special case of the class of single-output read-forward merge algorithms. Some information about these techniques can be found in the exercises for Sections 5.4.2 and 5.4.4 of [5]. It is known for example that in certain special cases, the optimal polyphase sort can be beaten by other methods of merging. In [9] it is shown that a large class of single-output read-forward merge algorithms also give rise to asymptotically optimal sorting algorithms.

Acknowledgments

This paper would have been a pale shadow of itself had I not encountered a preprint of the first half of [5]. Professor Knuth also read an early draft of the paper and made several valuable suggestions. This research was conducted while I was employed at Sperry UNIVAC in Roseville, Minnesota, and began as a study of possible sorting procedures for the DMS 1100 Schema Description (DDL) translator which I was then implementing. I am indebted to Mr. E. H. Moulton for many valuable conversations and to Mr. A. G. Reiter, Dr. H. C. Gyllstrom, and M. D. Thompson, for allowing me to pursue this subject when I should have been doing something else.

References

The references given below consist of only those books and papers which are referenced in the text. An extensive bibliography on computer sorting has been prepared by R. L. Rivest and D. E. Knuth which appears in Computing Reviews, vol. 13, no. 6 (June 1972), pp. 283-289.

[1] de Bruijn, N. G., Asymptotic Methods in Analysis, North-Holland, Amsterdam, 1970, 200 pp.

[2] Flores, I., Computer Sorting, Prentice-Hall, Englewood Cliffs, N. J., 1969, 237 pp.

[3] Gnedenko, B. V., and Kolmogorov, A. N., Limit Distributions for Sums of Independent Random Variables, Addison-Wesley, Reading, Mass., 1954, 264 pp.

[4] Knuth, D. Em, Fundamental Algorithms, The Art of Computer Programming 1, Addison-Wesley, Reading, Mass., 1968, 634 pp.

[5] Knuth, D. Em, Sorting and Searching, The Art of Computer Programming 3, Addison-Wesley, Reading, Mass., 1973, 722 pp.

[6] Lynch, W. Cm, "The t-Fibonacci Numbers and Polyphase Sorting," Fibonacci Quarterly, 8 (1970), pp. 6-22.

[7] Sackman, B. S., and Singer, T., "A Vector Model for Merge Sort Analysis, Fart I, Polyphase Merge Sort," unpublished paper presented at the ACM Sort Symposium (November 1962), 28 pp.

[8] Shell, D. L., "Optimizing the Polyphase Sort," Comm. ACM, 14, 11 (November 1971), pp. 713-719, Corrigendum, Comm. ACM 15, 1 (January 1972), p. 28.

[9] Zave, D. A., "A Note on Merge Sort Analysis," (in preparation).

[10] Zave, D. A., "A Simplified Analysis of the Optimal Polyphase Sort," (in preparation).

	j	$M_1^n(j)$	$M_2^n(j)$	$M_3^n(j)$	$M_4^n(j)$
$n = 1$	1	1	1	1	1
$n = 2$	1	0	1	1	1
	2.	1	1	1	1
$n = 3$	1	0	0	1	1
	2	1	2	2	2
	3	1	1	1	1
$n = 4$	1	0	0	0	1
	2	1	2	3	3
	3	2	3	3	3
	4	1	1	1	1
$n = 5$	1	0	0	0	0
	2	1	2	3	4
	3	3	5	6	6
	4	3	4	4	4
	5	1	1	1	1

Table 3.1. Movement Numbers for $t = 4$.

j	.n=	-3	-2	-1	0	1	2	3	4	5	6	7
0		0	1	0	0	0	0	0	0	0	0	0
1		0	1	0	0	1	1	0	0	0	0	0
2		0	1	0	0	1	2	2	2	2	1	0
3		0	1	0	0	1	2	3	5	7	8	7
4		0	1	0	0	1	2	3	6	11	17	23
5		0	1	0	0	1	2	3	6	12	22	37
6		0	1	0	0	1	2	3	6	12	23	43
7		0	1	0	0	1	2	3	6	12	23	44

Table 3.2. $S_2^n(j)$ for $t = 4$.

n =	0	1	2	3
j = 0	0	t		
1	-1	0	t	
2	t-2	-1	-1	t-1

Table 6.1(a). Proof of Lemma 6.4 ($t \geq 4$) .

n =	0	1	2	3	4	5	6
j = 0	0	3	2				
1	-1	0	3	5			
2	1	-1	-1	2	8		
3	3	1	0	-1	0	y	
4				4	o	-1	8

Table 6.1(b). Proof of Lemma 6.4 ($t = 3$) .

n =	0	1	2	3	4	5	6	7	8	y	10	11	12	13	14	15	16	17	18	19
j = 0	0	2	1	0	0	0														
1	-1	0	2	3	10		0													
2	0	-1	-1	2	5	4	1	0												
3	1	0	-1	-2	1	7	9	5	1											
4	2	1	1	-1	-3	-1	8	16	14	6										
5	3	2	3	2	0	-4	-4	7	24	30	20									
6		3	5	5	5	2	-4	-8	3	31	54	50								
7				8	10	10	7	-2	-12	-5	34	85	104							
8						18	20	17	5	-14	-17	29	119	189						
9								38	37	22	-9	-31	12	148	308					
10										75	59	13	-40	-19	160	456				
11												134	72	-27	-59	141	616			
12														206	45	-86	82	757		
13																251	-41	-4	839	
14																		210	-45	835

Table 6.1(c). Proof of Lemma 6.4 (t = 2) .

n	t = 2	t = 3	t = 4	t = 5	t = 6	t = 7
1	2	3	4	5	6	7
2	3	5	7	9	11	13
3	5	9	13	13	16	19
4	8	17	22	28	19	23
5	13	31	34	42	52	26
6	21	54	75	60	72	87
7	34	95	108	153	97	114
8	55	172	243	215	282	147
9	89	279	358	268	385	167
10	144	534	455	778	480	639
11	233	819	1196	1033	554	791
12	377	1634	1562	1248	1995	921
13	610	2400	4033	3909	2485	1016
14	987	4958	5378	4969	2900	4396
15	1597	7028	6455	5840	10577	5250
16	2573	14952	18560	19408	13096	5978
17	3954	20582	22875	23917	15335	6498
18	6527	44898	64188	27556	17028	30163
19	10488	60297	80858	95802	69843	35027

Table 6.2. L_n for $2 \leq t \leq 7$ and $1 \leq n \leq 19$.

n =	-1	0	1	2	3
j = 0	0	1			
1	-1	0	1	1	
2	-1	t-2	-1	0	1

Table 7.1(a). Proof of Lemma 7.1 ($t \geq 4$, $i=t$).

n =	1-t	. . .	0	1
j = 0	1	. . .	0	
1	1	. . .	-1	1

Table 7.1(b). Proof of Lemma 7.1 ($t \geq 3$, $i = 1$).

n =	i-t-1	i-t			. . .	0	1	2	3
j = 0			1	0	. . .	0			
1	-1	1		0	. . .	-1	1	1	
2		-1	1	0	. . .	t-3	-1	≥ 0	≥ 1

Table 7.1(c). Proof of Lemma 7.1 ($t \geq 4$, $1 < i < t$).

n =	-2	-1	0	1	2
j = 0	0	1	0'		
1	-1	1	-1	1	
2	-1	1	0	-1	1

Table 7.1(d). ProofofLema7.1 (t = 3 , i = 2) .

n =	-1	0	1	2	3	4	5	6
j = 0	0	1						
1	-1	0	1	1	1			
2	-1	1	-1	0	2	3		
3	-1	3	0	-1	0	1	5	
4	-1	5	2	2	2	-1	0	6

Table 7.1(e). ProofofLemma7.1 (t = 3 , i = 3) .

n =	-1	o	1	2	3	4	5	6	7	8	y	10	11	12	13	14	15	16	17	18	19
j= 0	1	0	0	0	0	0'	0														
1	1	-1	1	0	0	0	0	0													
2	1	-1	0	0	1	0	0	0	0												
3	1	0	0	-1	0	1	1	0	0	0											
4	1	1	1	0	-1	-1	1	2	1	0	0										
5'	1	2	2	2	1	-1	-2	0	3	3	1	0									
6			3	4	4	3	o	-3	-2	3	6	4	1								
7					7	8	7	3	-3	-5	1	9	10	5							
8							15	15	10	0	-8	-4	10	19	15						
9									30	25	10	-8	-12	6	29	34					
10											55	35	2	-20	-6	35	63				
11													90	37	-3.8	-26	29	98			
12															127	19	-44	3	127		
13																	146	-25	-41	130	
14																			121	-66	89

Table 7.1(f). Proof of Lemma 7.1 (t = 2 , i = 1) .

n =	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
j= 0	0	1	0	0	0	0	0															
1	-1	0	1	1	0	0	0	0														
2	-1	0	-1	1	2	1	0	0	0													
3	-1	1	-1	-l	0	3	3	10		0												
4'	-1	2	0	0	-2		-1	3		641	0											
5	-1	3	1	2	0	-2	-3	2	9	10	5	1										
6		4	2	4	3	2	-2	-5	-1	11	19	15	6									
7				6		67		5o	-7	-6	10	30	34	21								
8						12	13	12	5	-7	-13	4	40	64	55							
9								25	25	17	-2	-20	-9	44	104	119						
10										50	42	15	-22	-29	35	148	223					
11												92	57	-7	-51	6	183	371				
12														149	50	-58	-45	189	556			
13																199	-8	-103	144	745		
14																		191	-111	41	899	
15																				80	-70	930

Table 7.1(g). ProofofLema7.1 (t = 2 , i = 2) .

k	n^k	q^k	q_1^k	q_2^k	q_3^k	q_4^k
1	1	4	1	1	1	1
2	2	7	1	2	2	2
3	3	9	1	2	2	2
4	3	13	2	3	4	4
5	4	21	3	5	6	7
6	4	22	4	6	7	8
7	5	30	4	7	9	10
8	5	34	4	8	11	13
9	6	36	4	8	11	13
10	6	71	10	17	21	23
11	6	75	13	22	26	28
12	7	100	13	23	30	34
13	7	108	14	27	37	44
14	8	322	14	27	37	44
15	8	241	34	57	71	79
16	8	243	44	77	92	100
17	9	338	44	78	101	115
18	9	358	50	94	128	151
19	10	423	50	94	128	151
20	10	455	50	100	144	178
21	11	472	50	100	144	178
22	11	1156	151	266	345	394

Table 7.2. Simplified Quota Scheme for $t = 4$.