EXTERNAL SORTING WITH Polyphase Merging

Department of Computer Science

PRESENTATION BY

Adrita Chakraborty
Bony Gain
Debjit Dey
Ria Das

External Sorting

- **Performing sorting operations on amounts of data that are too large to fit into main memory. So this large set of data is stored in comparatively slow peripheral devices (like hard disks, magnetic tapes etc.) which are known as Secondary Memory or External Memory because CPU can not access these memories directly.**
- **Basic concept of external sorting is internal sorting method followed by external merging.**

e.g. Balanced Two Way Merge,
Multi way Merge,
Polyphase Merge etc.

Mechanism

1. Split the data into pieces that fit in the internal memory (main/primary memory).
2. Sort the pieces with conventional internal sorting algorithms (such as Quick sort, Merge sort etc.)
3. Merge those and build the completely sorted data-set.

Most of the external sorting algorithms are variations of this concept. Such as,

Balanced two-way merge
Multi-way merge
Poly-phase merge etc

We limit our discussion with Poly-phase merge.

Basic Terminologies

Before we begin with a brief discussion of the poly-phase merge, we introduce our terminology commonly referred in the algorithms.

- **Run:** A sorted segment of a file, which is termed as an ascending run or simply a run.
- **Tape:** It is an auxiliary storage space. Usually tapes are used to store huge data.
- **Access time:** Auxiliary storage devices are very slow compared to primary memory. These access is due to read (read time, T_r), write a record (write time, T_w), and rewind a tape to beginning of the tape (rewind time, T_f), etc.

Balanced Two Way Merge Sort

- A balanced two-way merge sort sorts a data stream using repeated merges. It distributes the input into two streams by repeatedly reading a block of input that fits in memory, a run, sorting it, then writing it to the next stream. It then repeatedly merges the two streams and puts each merged run into one of two output streams until there is a single sorted output.

- Balanced Two Way Merge sort requires four magnetic tape to sort the data which is sorted in one of them.

EXAMPLE

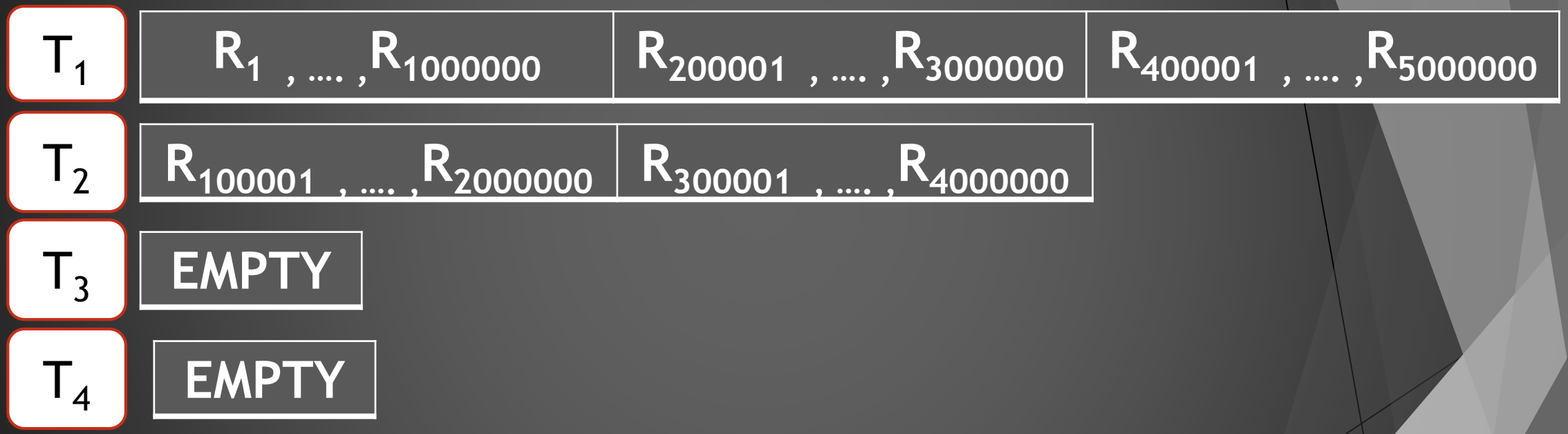
Let the four tapes T1,T2,T3 and T4.

Input File :

R_1 , …. , $R_{1000000}$	R_{100001} , …. , $R_{2000000}$	R_{200001} , …. , $R_{3000000}$	R_{300001} , …. , $R_{4000000}$	R_{400001} , …. , $R_{5000000}$

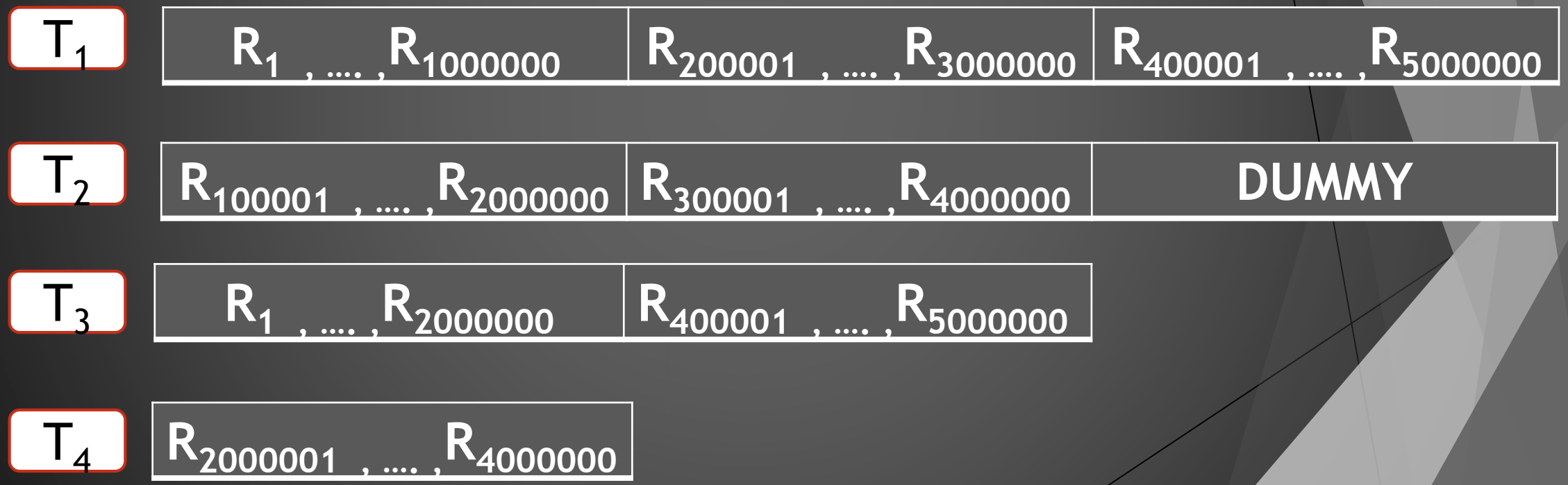
Step 1:

- Initial ascending runs by internal sort.
- This runs are placed alternately on T1 and T2, until the input is exhausted and T3 and T4 empty.



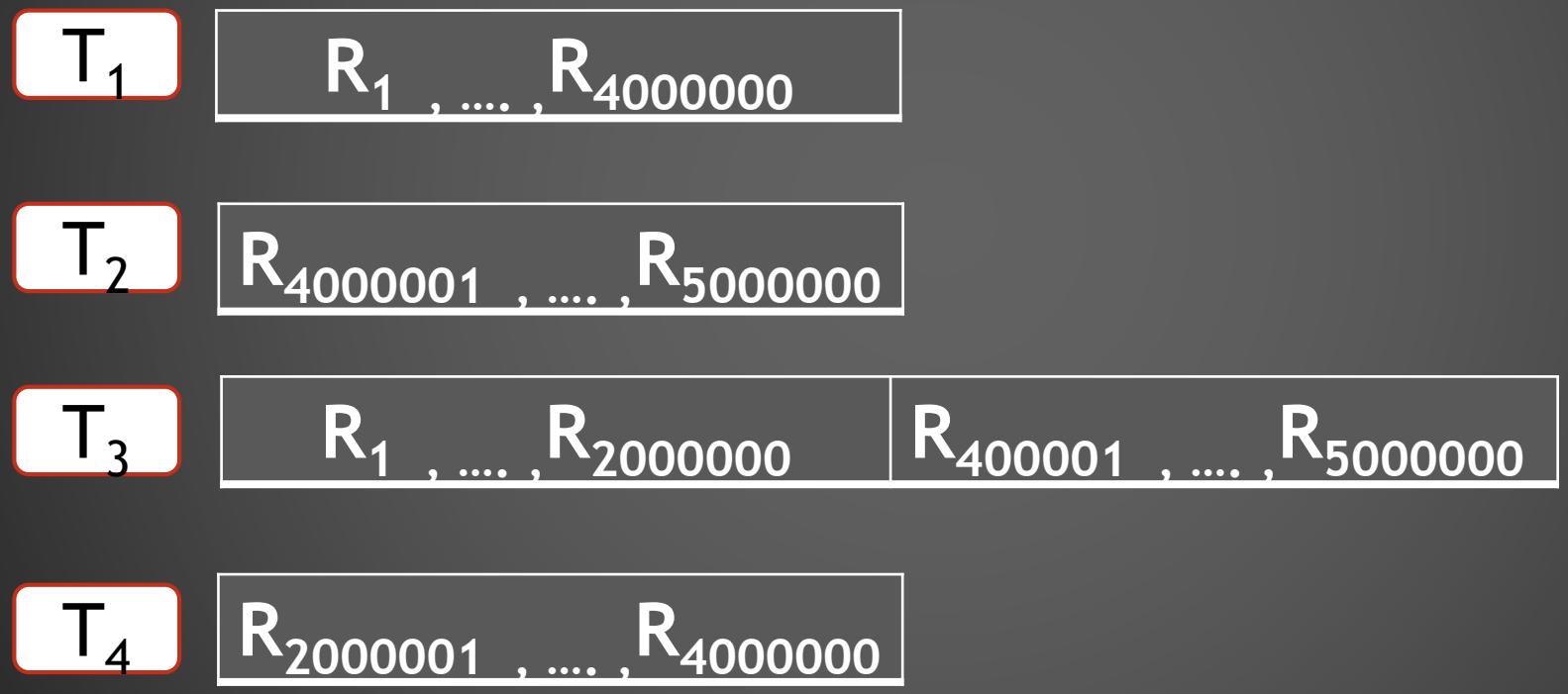
Step 2:

- The first pass of merging produces longer runs on T3 and T4, as it reads T1 and T2.
- when a tape contains one block more than other then final merge operation.
- In this step will occur dummy block of size 0 as follows:



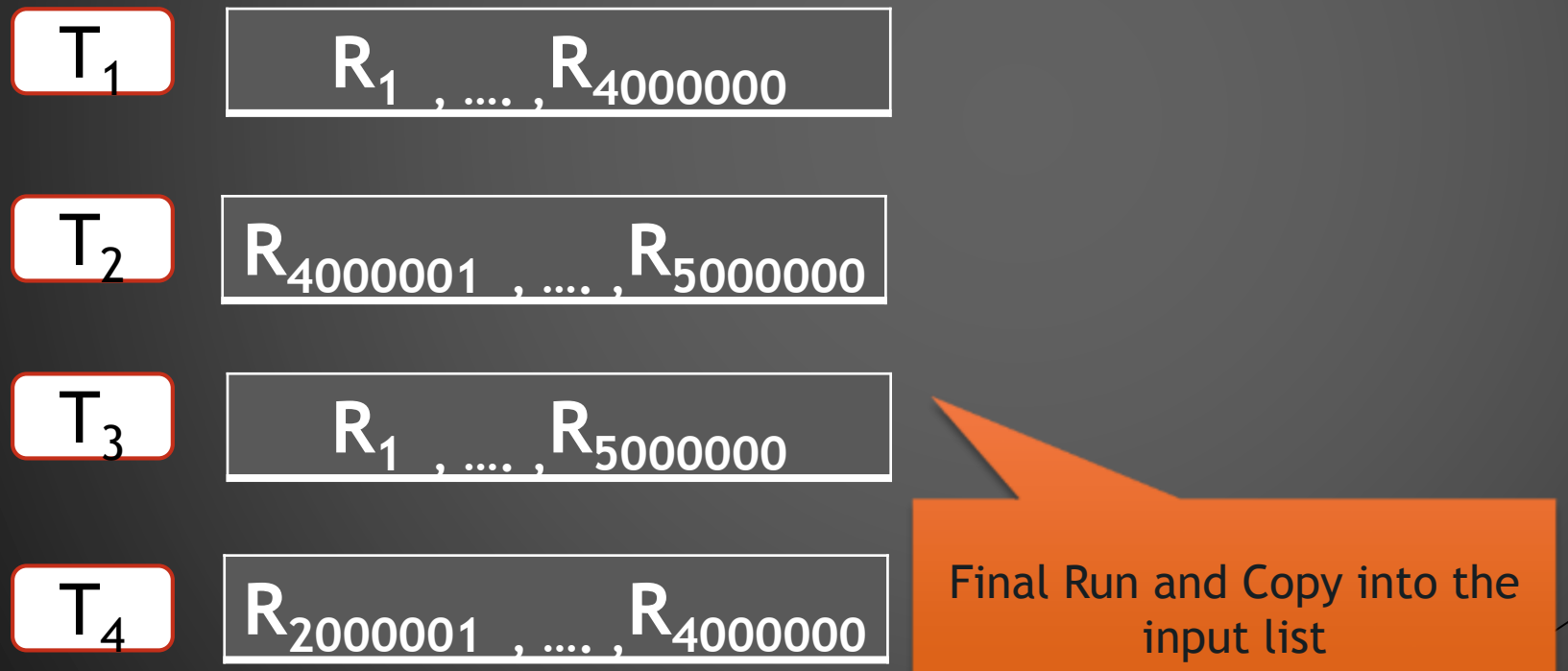
Step 3:

The second pass of merging produces ,same as Step2.Merging result stored alternatively into two tapes T1 and T2 And here just copy RUN5 into T2 tapes



Step 4:

- The third pass of merging produces ,same as Step2.
- Merging result stored into two tapes T3 and T4.
- here the final result for this example stored in tape T3.
- Copy the final sorted file into the orijinal input list.



Problem with Balancd Two-Way Merge

In Balance Two-way merge it can be noticed that when a tape contains one block more than other, then the final merge operation in this step will occurs with a dummy block of size 0.

Multi Way Merge Sort

- The balanced two way merge is based on combining two ascending runs into a single ascending run.This is why it id two way merging.
- In multi-way merge sorting procedure is extended to m-way merging, where m(m>2) runs are combine into a single run.

✓ Let's assume that we have been given P ascending runs , that is, sequence of records whose keys are is non-decreasing order.

Steps :

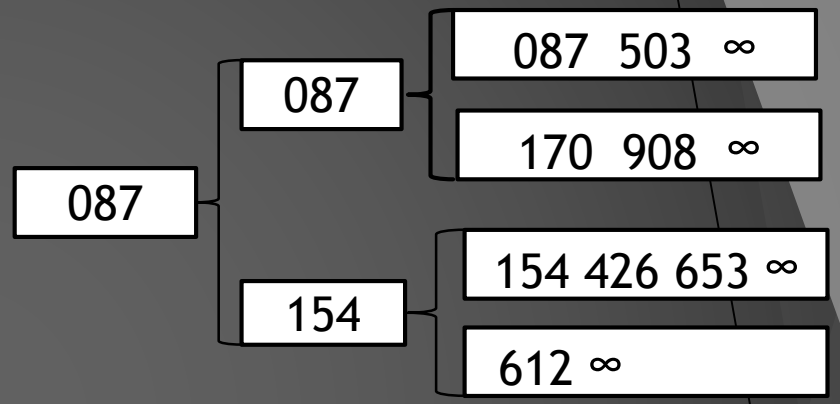
1. Find the record whose key value is minimum.
2. Transfer the record to the output and remove that from the input.
3. Repeat the process until the input tapes become empty.

{If two or more keys are smallest then an arbitrary one is selected}

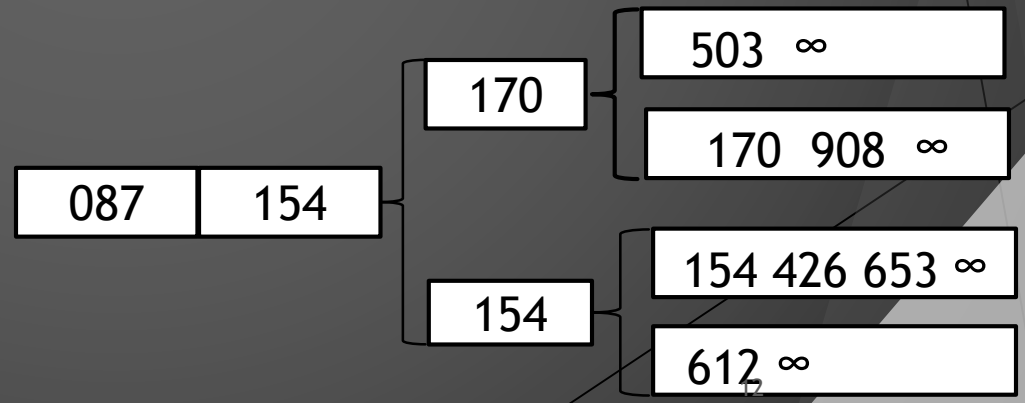
EXAMPLE *(POCESS1)*

We have taken four runs which are stored in four tapes.

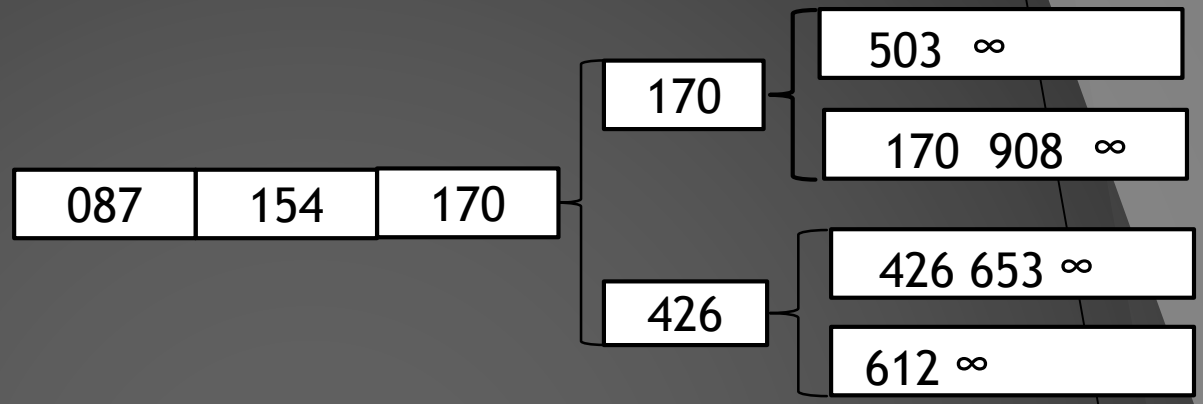
Step 1



Step 2

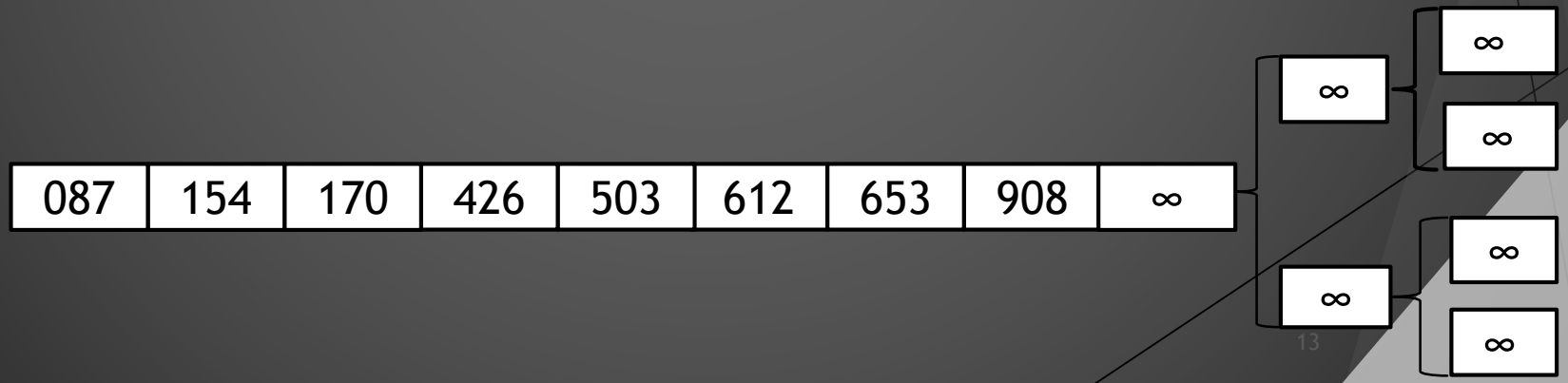


Step 3


087
154
170
170
426
503 ∞
170 908 ∞
426 653 ∞
612 ∞

-
-
-

Step 9


087
154
170
426
503
612
653
908
∞
∞
∞
∞
∞
∞
∞

(PROCESS 2)

- We also see the multi way merging in the binary tree in same procedure.
- It is shown in the belows figure.

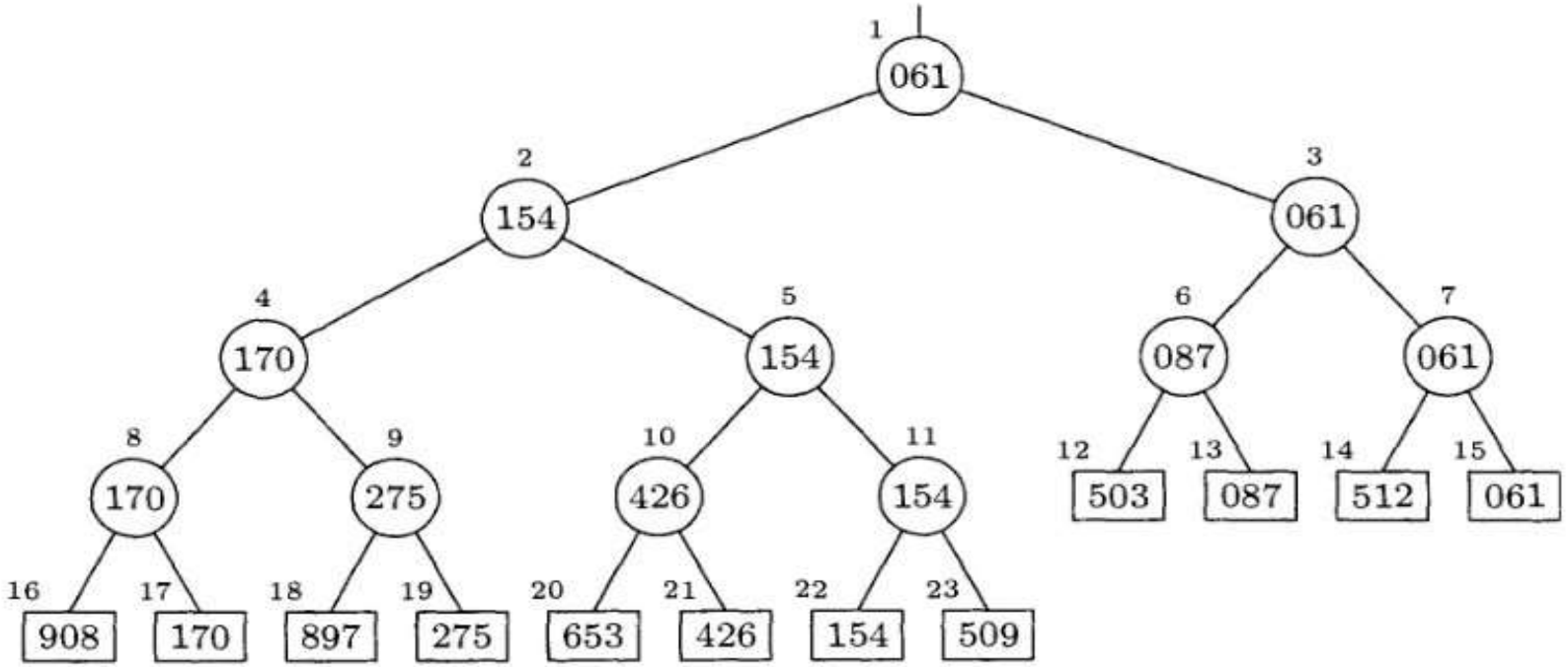


Fig. 62. A tournament to select the smallest key, using a complete binary tree whose nodes are numbered from 1 to 23.

Problem with Multi Way Merge

Here, m input runs are stored on m tapes, and internal memory should be large enough to hold m blocks so that one block from each input tape resides in the main memory because we merge them together, i.e., we look at the first element of each run and select the smallest element.

15

Polyphase Merge

- A polyphase merge sort decreases the number of runs at every iteration of the main loop by merging runs into larger runs.
- Used for external sorting.
- Combines both the Balanced Two-way and Multi-way merging techniques.
- Use of the *empty input file* by turning that file into the *output file*.

Balance Merging :

Assuming three tape units, T1, T2, and T3 is described below:

Steps:

1.Distribute initial runs alternatively on Tapes T1 and T2.
2.Merge runs from T1 and T2 onto T3; then stop if T3 contains only one run.
3.Copy the runs of T3 alternatively onto T1 & T2,then return to step 2.

If 17≤S(no. of runs)≤32,then we have 1 distribution,5 merge and 4 copy passes. In general, if S>1 the total number of passes is *2[logS].*

To avoid half of the copying, we use ***two-phase procedure*** given below:

Steps:

1. Distributed initial runs alternately on the tape T1 and T2.
2. Merge runs from T1 and T2 onto T3.
3. If T3 contains only one run then go to step 9.
4. Copy half of the runs form T3 into T1
5. Merge form T1 and T3 into T2.
6. If T2 contains only one run then go to step 9.
7. Copy half of the runs from T2 into T1.
8. Return to 2.
9. Stop.

The number of passes over the data has been reduced to $3/2[\log_2 S] + 1/2$, since steps 4 and 7 do only "half pass"; about 25 percent of the time has therefore been saved.

The copying can actually be eliminated entirely, if we start with initially F_n runs into T1 and F_{n-1} runs into T2 where F_n and F_{n-1} are consecutive *Fibonacci number (proposed by B.K.Betz)*.

Example : let total number of runs(S)= $F_n + F_{n-1}$ =13+8 =21

Phase	Contents of T1	Contents of T2	Contents of T3	Remarks
1	1,1,1,1,1,1,1,1,1,1,1,1,1	1,1,1,1,1,1,1,1		Initial distribution
2	1,1,1,1,1	-	2,2,2,2,2,2,2,2	Merge 8 runs to T3
3	-	3,3,3,3,3	2,2,2	Merge 5 runs to T2
4	5,5,5	3,3	-	Merge 3 runs to T1
5	5	-	8,8	Merge 2 runs to T3
6	-	13	8	Merge 1 runs to T2
7	21	-	-	Merge 1 runs to T1

Considering each initial run to be of relative length 1. The total no. of "passes" comes to (21+16+15+15+16+13+21)/21=5.57≈6.

The *perfect Fibonacci distributions :*

For example, when T=6 we have the following distribution of runs:

Level	T1	T2	T3	T4	T5	Total	Final output will be on
0	1	0	0	0	0	1	T1
1	1	1	1	1	1	5	T6
2	2	2	2	2	1	9	T5
3	4	4	4	3	2	17	T4
4	8	8	7	6	4	33	T3
5	16	15	14	12	8	65	T2
6	31	30	28	24	16	129	T1
7	61	59	55	47	31	253	T6
8	120	116	108	92	61	497	T5
.....	.....	.....	.....	.....	.....	.....	.....
n	a_n	b_n	c_n	d_n	e_n	t_n	T(k)
n+1	a_n+b_n	a_n+c_n	a_n+d_n	a_n+e_n	a_n	t_n+4a_n	T(k-1)
.....	.....	.....	.....	.....	.....	.....	.....

(1)

19

The rule for going from level n to level n+1 shows that the condition

$$a_n \geq b_n \geq c_n \geq d_n \geq e_n \qquad (2)$$

Will hold in every level. In fact, it is easy to see from(1) that

$e_{n=} a_{n-1}$

$d_{n=} a_{n-1} + e_{n-1} = a_{n-1} + a_{n-2}$

$c_n = a_{n-1} + d_{n-1} = a_{n-1} + a_{n-2} + a_{n-3}$

$b_n = a_{n-1} + c_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4}$

$a_n = a_{n-1} + b_{n-1} = a_{n-1} + a_{n-2} + a_{n-3} + a_{n-4} + a_{n-5}$

Where $a_0 = 1$ and where we let $a_n = 0$ for n = -1 , -2 , -3, -4

20

Using the previous idea with T (T≥3)tapes ,using (T-1) way merging. The generalized pattern involves *Generalized Fibonacci numbers*.
Consider the following with *six tape* example:

Phase	T1	T2	T3	T4	T5	T6	Initial runs processed
1	1^{31}	1^{30}	1^{28}	1^{24}	1^{16}	-	31+30+28+24+16=129
2	1^{15}	1^{14}	1^{12}	1^{8}	-	5^{16}	16*5=80
3	1^{7}	1^{6}	1^{4}	-	9^{8}	5^{8}	8*9=72
4	1^{3}	1^{2}	-	17^{4}	9^{4}	5^{4}	4*17=68
5	1^{1}	-	33^{2}	17^{2}	9^{2}	5^{2}	2*33=66
6	-	65^{1}	33^{1}	17^{1}	9^{1}	5^{1}	1*65=65
7	129^{1}	-	-	-	-	-	1*129=129

Optimization Of Polyphase Merging:

- The pth order Fibonacci numbers $F_n^{(p)}$ are defined by the rules,

$$F_n^{(p)} = F_{n-1}^{(p)} + F_{n-2}^{(p)} + \ldots\ldots + F_{n-p}^{(p)}, \text{ for } n \geq p;$$

$$F_n^{(p)} = 0, \text{ for } 0 \leq n \leq p-2;\ F_{p-1}^{(p)} = 1$$

i.e. we start with p-1 0s,then 1,and then each number is the sum of the preceding p values. For larger value of p the sequence studied and derived the generating function by V.Schlegel is,

$$\sum F_n^{(p)} z^n = \frac{z^{p-1}}{1 - z - z^2 - \ldots\ldots - z^p} = \frac{z^{p-1} - z^p}{1 - 2z + z^{p+1}}$$

If we set P=T-1,the polyphase merge distributions for T tapes will correspond to Pth order Fibonacci numbers. In the same way kth tape gets

$$F_{n+p-2}^{(p)} + F_{n+p-3}^{(p)} + \ldots\ldots + F_{n+k-2}^{(p)}$$

Initial runs in the perfect nth level distribution,for 1<=k<=P,and the total number of initial runs on all tapes is therefore

$$t_n = PF_{n+P-2}^{(P)} + (P-1)F_{n+P-3}^{(P)} + \ldots\ldots + F_{n-1}^{(P)}$$

This settle the issue of "perfect Fibonacci distributions".

- When S(no. of runs) is not exactly equal to t_n (perfect Fibonacci no.) for any n, we can add artificial "dummy runs" so that we can pretend S is perfect after all.

Polyphase merge sorting with horizontal distribution :

Algorithm :

This algorithm takes initial runs and disperses them to tapes, one run at a time, until the supply of initial runs is exhausted. Then it specifies how the tapes are to be merged, assuming that there are T= P+1 ≥ 3 available tape units, using P way merging. Tape T may be used to hold the input, since it does not receive any initial runs.

- A[j] , 1 ≤ j ≤ T: the perfect Fibonacci distribution we are striving for.
- D[j], 1 ≤ j ≤ T: number of dummy runs assumed to be present at the beginning of logical tape unit number j
- TAPE[j], 1 ≤ j ≤ T: Number of physical tape unit corresponding to the logical tape unit number j

D1. [Initialize.] Set A *[j]* <— D *[j]*- 1 and TAPE [k] <— j, for 1 ≤ *j* < *T*. Set A [T] <— D [T] <— 0 and TAPE [T] <— *T*. Then set *l* <- *1*, *j* <- **1**.

D2. [Input to tape *j.]* Write one run on tape number j, and decrease D *[j]* by 1. Then if the input is exhausted, rewind all the tapes and go to step D5.

D3. [Advance *j.]* If **D [*j]* < D *[j]* ±** 1] , increase *j* by 1 and return to D2. Otherwise if D *[j]* = 0, go on to D4. Otherwise set *j* <— 1 and return to D2.

D4. [Up a level.] Set *l* ← *l* + 1, a <— A [1] , and then for *j* = 1, 2, ... , *P* (in this order) set D [*j]* <— a + A *[j* + 1] — A *[j]* and A *[j]* <— a + A *[j* + 1] . Now set *j* <— 1 and return to D2.

D5. [Merge.] If *l* = 0, sorting is complete and the output is on **TAPE [1] .** Otherwise, merge runs from **TAPE [1] , . . . ,TAPE** [P] onto **TAPE** [7] until **TAPE [P]** is empty and **D** *[P]* = 0. The merging process should operate as follows, for each run merged: If D *[j]* > 0 for all *j*, 1 ≤ j ≤ P, then increase D[T] by 1 and decrease each D *[j]* by 1 for 1 ≤ j ≤ P; otherwise merge one run from each **TAPE** *[j]* such that **D *[j]* =** 0, and decrease **D** *[j]* by 1 for each other *j*. (Thus the dummy runs are imagined to be at the *beginning* of the tape instead of at the ending.)

24

D6. [Down a level.] Set *l* <- *1* -1. Rewind **TAPE** [P] and **TAPE [T] .** (Actually the rewinding of **TAPE *[P]*** could have been initiated during step D5, just after its last block was input.) Then set **(TAPE [1] ,TAPE [2] , . . . ,TAPE [T]) <- (TAPE [7] ,TAPE [1] , . . . ,TAPE [7' - 1]), (D [1] , D [2]** $_1$ **. . . , D [T]) (D [T]** $_1$ **D [1]** $_1$ **. . .** $_1$ **D *[T* 1])**, and return to step D5.

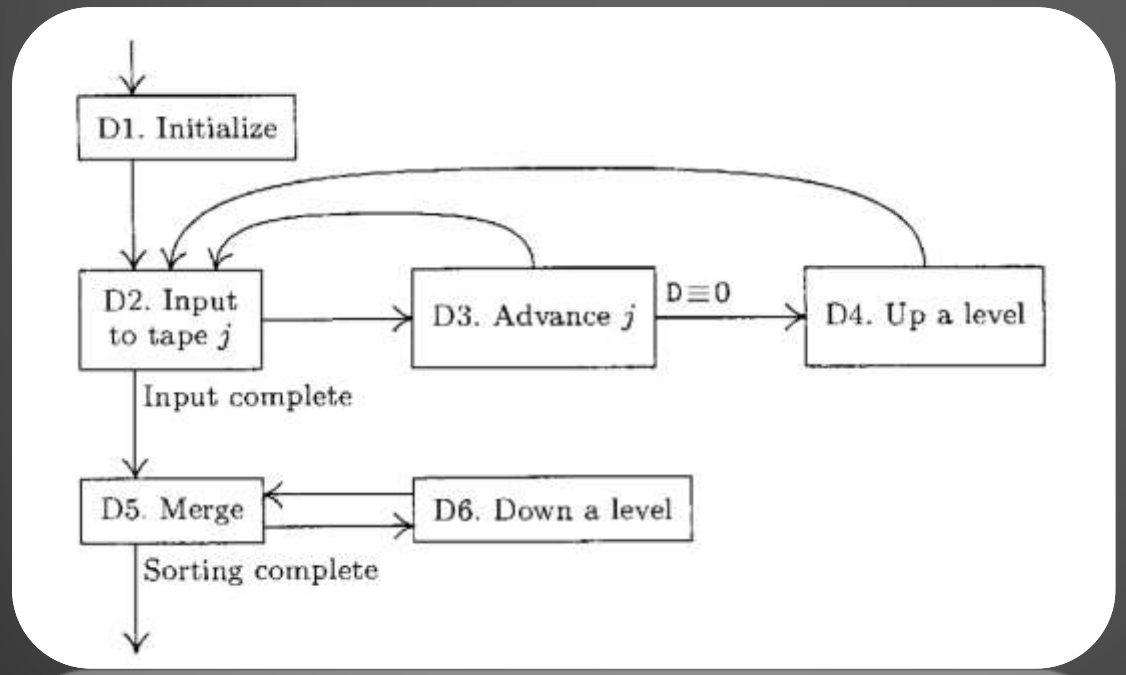


fig : polyphase merge sorting

Before we begin our analysis:

If we have N records in our original file, and we can store S records at one time, this means that after CreateRuns we must have $R = \lceil N/S \rceil$ runs split between the two files. Each of the PolyphaseMerge passes joins pairs of runs, so it must cut the number of runs in half. After one pass there will be $\lceil R/2 \rceil$ runs, after two passes there will be $\lceil R/4 \rceil$ runs, and, in general, after j passes there will be $\lceil R/2^j \rceil$ runs. Because we stop when we get down to one run, this will be when $\lceil \lg R \rceil$ is equal to 1.

Analysis of Polyphase Merge Sort

Number of comparisons in Run Construction:

- Assume that an O(N log N) sort is used.
- S elements in each run, each will take O(S log S) to construct.
- R runs, give O(R * S * log S) = O(N log S) comparisons for construction of all runs.
- Thus, run construction phase is O(N log S).

Number of comparisons in Run Merge:

- Number of elements in List1 = A
- Number of elements in List2 = B

So, internal Merge sort does A + B - 1 comparisons in the worst case.

- On the first pass, we have R runs of size S that get merged, so there are R / 2 merges, each of which will take at most 2S - 1 comparisons, or R / 2 * (2S - 1) = R * S - R / 2 comparisons.
- On the second pass, we have R / 2 runs of size 2S, so there are R / 4 merges, each of which will take at most 2(2S) - 1 comparisons, or R / 4 * (4S - 1) = R * S - R / 4 comparisons.
- On the third pass, we have R / 4 runs of size 4S, so there are R / 8 merges, each of which will take at most 2(4S) - 1 comparisons, or R / 8 * (8S - 1) = R * S - R / 8 comparisons.

If we recall that there will be log R merge passes, the total number of comparisons in the merge phase will be,

$$\sum_{i=1}^{\lg R}(R*S-R/2^i)=\sum_{i=1}^{\lg R}(R*S)-\sum_{i=1}^{\lg R}(R/2^i)$$

$$=(R*S)*\lg R-R*\sum_{i=1}^{\lg R}1/2^i$$

$$\approx(R*S)*\lg R-R$$

$$=N\lg R-R$$

The run merge phase is O(N log R). This makes the entire algorithm

O(N log S + N log R) = O[N * (log S + log R)]

= O[N * log(S * R)]

=O[N * log(S * N/S)]

=O(N log N)

28

Modification of Polyphase Merging

There might be N files, but the polyphase merge will read from N-1 files throughout and write only one output file at a time. The writing to that output file continues until an input file is exhausted, and then that input file becomes the new output file. The number of runs in each file is related to Fibonacci numbers and Fibonacci numbers of higher order, while the cascade merge uses (T-1) way, (T-2)way, (T-3)way, etc.

References:

- http://en.wikipedia.org/wiki/Polyphase_merge_sort
- Donald.E.Knuth, The Art of Computer Programming Volume-3 (searching and sorting).
- Jeffrey J. McConnell, Analysis of Algorithms(An Active Learning Approach)

30


Any
Questions