

External Sorting

Why can't we simply do quicksort on a disk?

To get $a[i]$ on

☞ internal memory – $O(1)$

☞ hard disk



1. find the track;

2. find the sector;

3. find $a[i]$ and transmit.

device-dependent



Tool: Mergesort



To simplify –

☞ Store data on tapes (can only be accessed sequentially)

☞ Can use at least **3** tape drives

[[**Example**]] Suppose that the internal memory can handle $M = 3$ records at a time.

T_1 81 | 94 | 11 | 96 | 12 | 35 | 17 | 99 | 28 | 58 | 41 | 75 | 15

Internal memory 12 | 35 | 96

Run

{ T_2 11 | 81 | 94 | 17 | 28 | 99 | 15
 T_3 12 | 35 | 96 | 41 | 58 | 75 |

Number of passes = 1 + 3

$$1 + \lceil \log_2(N / M) \rceil$$

{ T_1 11 | 12 | 35 | 81 | 94 | 96 | 15
 T_4 17 | 28 | 41 | 58 | 75 | 99 |

{ T_2 11 | 12 | 17 | 28 | 35 | 41 | 58 | 75 | 81 | 94 | 96 | 99 |
 T_3 15

What are the concerns?

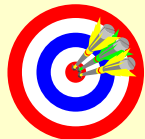
🕒 **Seek** time — $O(\text{number of passes})$

🕒 Time to **read or write** one **block** of records

🕒 Time to **internally sort** M records

🕒 Time to **merge** N records from input buffers to the output buffer

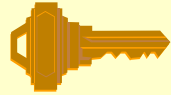
👉 Computer can carry out I/O and CPU processing in **parallel**



Targets:

- *Reduction of the number of passes*
- *Run merging*
- *Buffer handling for parallel operation*
- *Run generation*

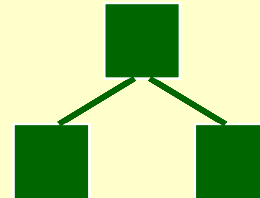
How to reduce the number of passes?



Use a *k-way* merge!

T_1 81 94 11 | 96 12 35 | 17 99 28 | 58 41 75 | 15

$\left\{ \begin{array}{l} T_2 \\ T_3 \\ T_4 \end{array} \right.$
 11 81 94 | 41 58 75 |
 12 35 96 | 15
 17 28 99 |



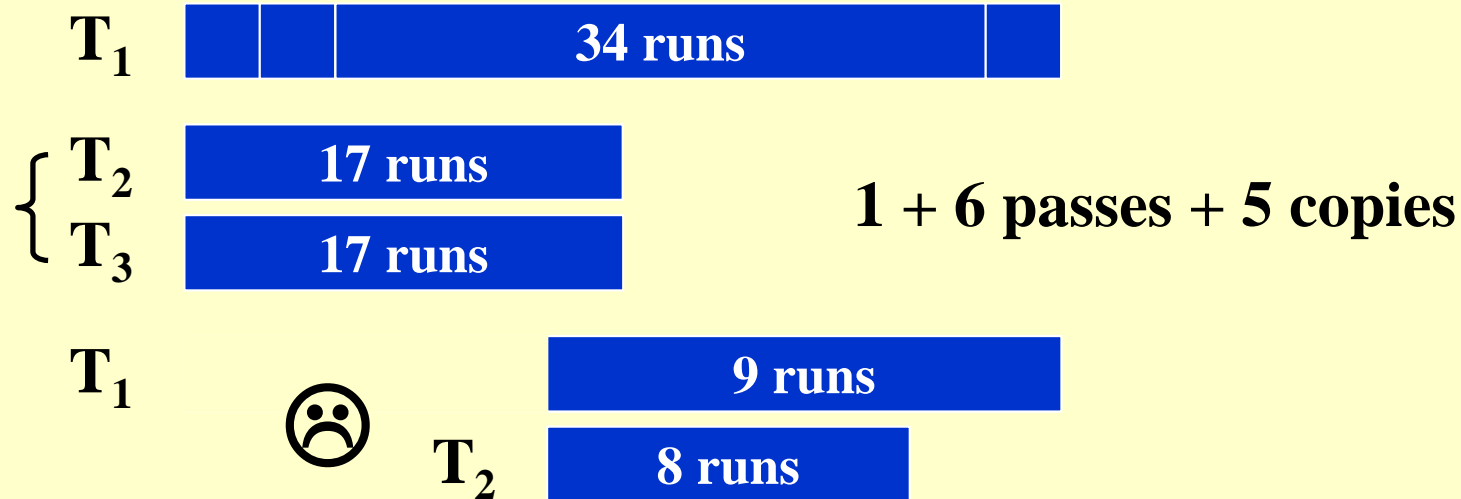
$\left\{ \begin{array}{l} T_1 \\ T_5 \\ T_6 \end{array} \right.$
 11 12 17 28 35 81 94 96 99 |
 15 41 58 75 |

$$\text{Number of passes} = 1 + \lceil \log_k (N / M) \rceil$$

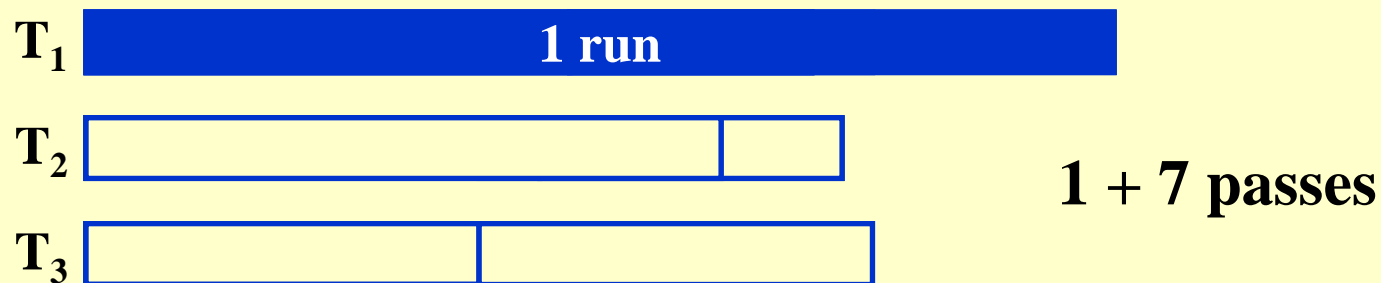


Require *2k* tapes!

Can we use 3 tapes for a 2-way merge?



👉 A smarter way – split *unevenly*



Discussion 20:

What will happen if 22 runs are placed on T_2 , with 12 on T_3 ?

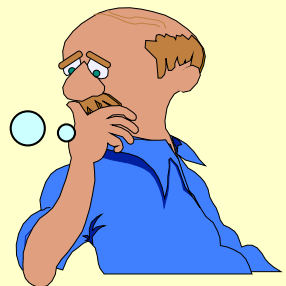
Claim: If the number of runs is a Fibonacci number F_N , then the best way to distribute them is to split them into F_{N-1} and F_{N-2} .

Claim: For a k -way merge, $F_N^{(k)} = F_{N-1}^{(k)} + \dots + F_{N-k}^{(k)}$
 where $F_N^{(k)} = 0$ ($0 \leq N \leq k-2$), $F_{k-1}^{(k)} = 1$

Polyphase Merge

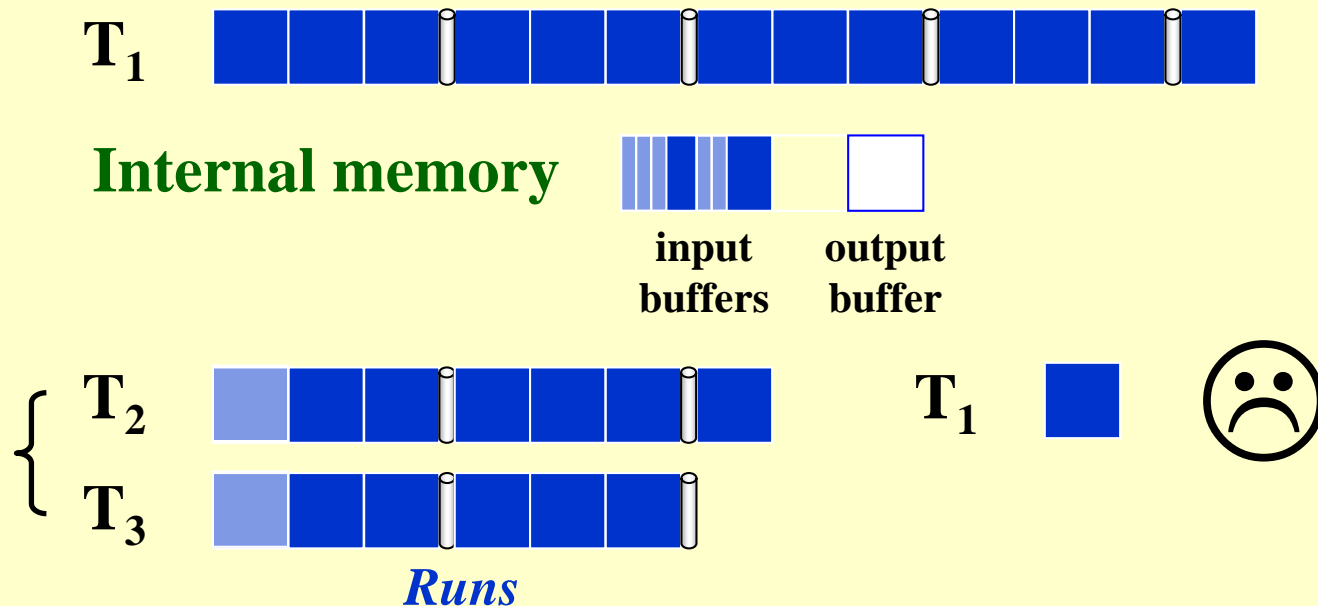
$k + 1$ tapes only

What if the initial
 number of runs is NOT
 a Fibonacci number?



How to handle the buffers for parallel operation?

【**Example**】 Sort a file containing **3250** records, using a computer with an internal memory capable of sorting at most **750** records. The input file has a block length of **250** records.



2-way merge

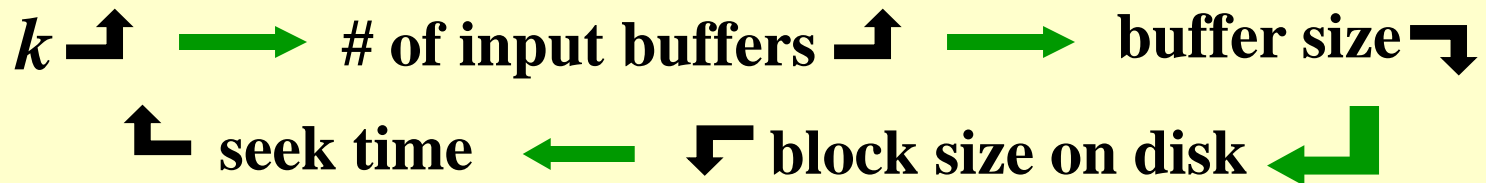


input buffers



output buffer

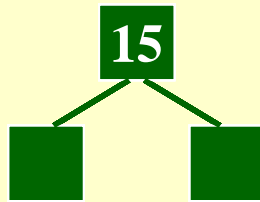
In general, for a k -way merge we need $2k$ input buffers and 2 output buffers for parallel operations.



Beyond a certain k value, the I/O time would actually **increase** despite the decrease in the number of passes being made. The optimal value for k clearly depends on disk parameters and the amount of internal memory available for buffers.

Can we generate a longer run?

81	94	11	96	12	35	17	99	28	58	41	75	15
----	----	----	----	----	----	----	----	----	----	----	----	----



*Replacement
selection*

11	81	94	96
----	----	----	----

12	17	28	35	41	58	75	99
----	----	----	----	----	----	----	----

$$L_{avg} = 2M$$

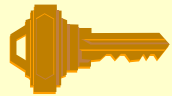
15



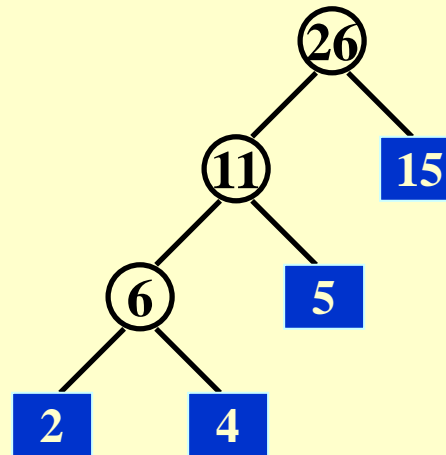
Powerful when input is often *nearly sorted* for external sorting.

How to minimize the merge time?

【Example】 Suppose we have 4 runs of length 2, 4, 5, and 15, respectively. How can we arrange the merging to obtain **minimum merge times**?



Huffman Tree!



$$2 \times 3 + 4 \times 3 + 5 \times 2 + 15 \times 1 = 43$$

Total merge time = O (*the weighted external path length*)

Reference:

Data Structure and Algorithm Analysis in C (2nd Edition):
Ch.6, p.222-227; *M.A.Weiss 著、陈越改编，人民邮电出版社，2005*

The Fibonacci Numbers and Polyphase Sorting; W.
C. LYNCH, Case Institute of Technology, Cleveland, Ohio