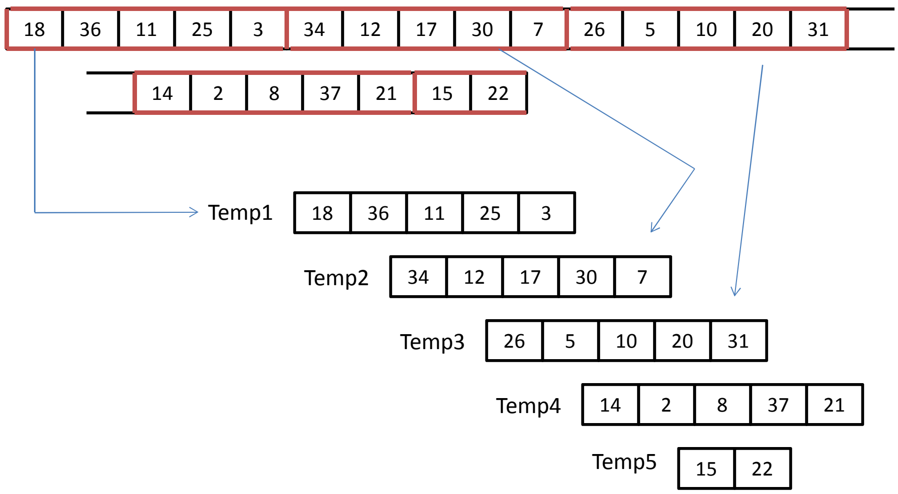
CSC 344 – Algorithms and Complexity

Lecture #4 – External Sorting

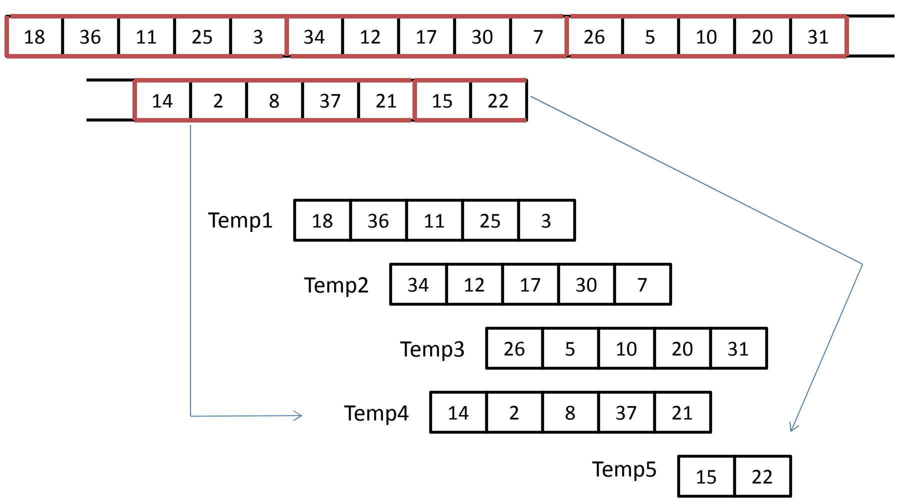
Why External Sorts?

- We can't fit the entire file in memory
- Therefore, we break up the file into fragments small enough to store in memory, sort them and then merge them back together.
- To keep the sort as time-efficient as possible we need to minimize the read and write operations.

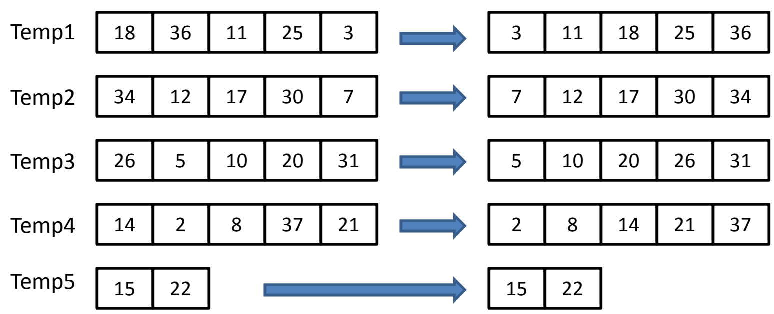
Creating The Temporary Files or “Runs”



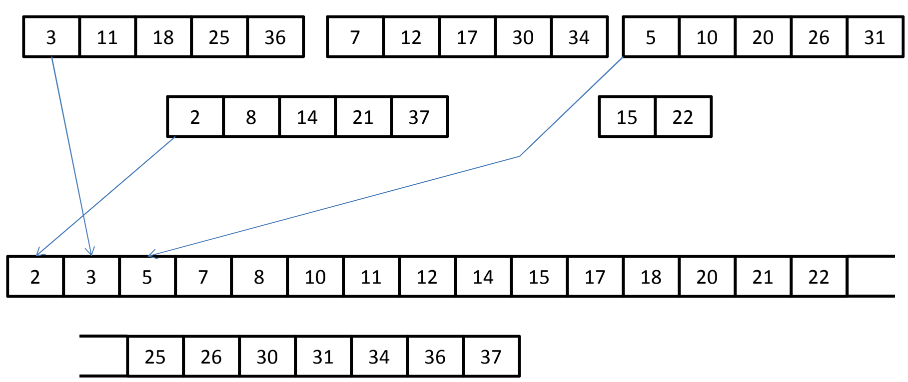
Creating The Runs (continued)



Sorting The Runs



Merging The Runs



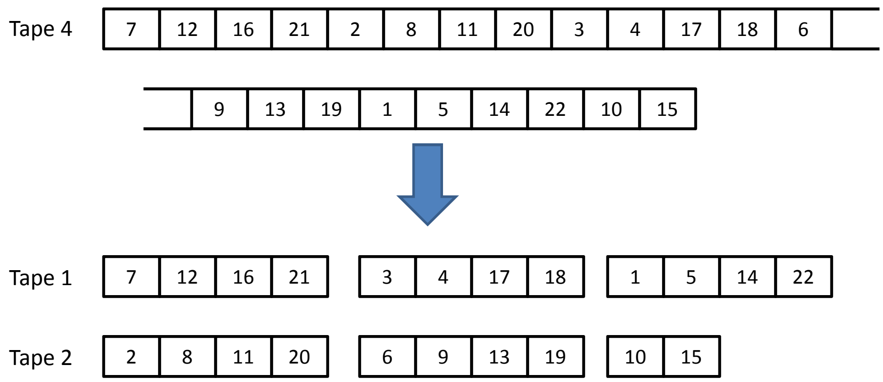
Cost of Merging

- If the source file has n records and memory can store m records, we need $\lceil n/m \rceil$ temporary files.
- We need to read and write each record twice: once during the sorting and once during the merging.
- Because this is the most time consuming task, the cost is $2n$.

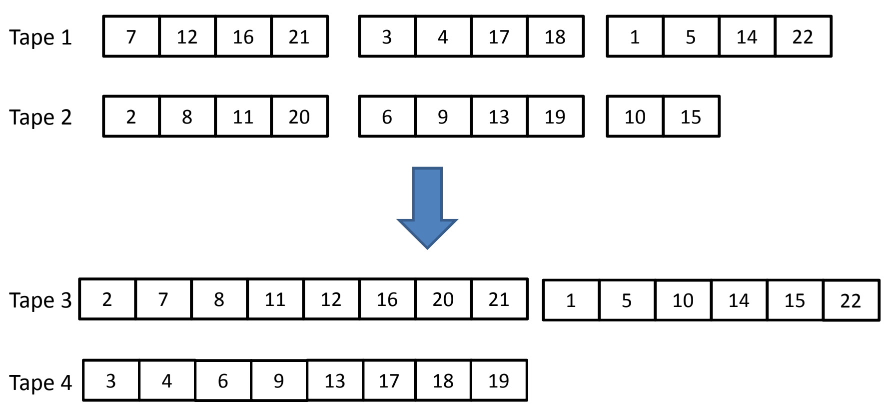
Balanced Merging

- While most disk drives can work with large numbers of temporary files, this won't work as well for tape storage.
- It is difficult to have multiple files on one tape and we have a limited number of tape drives.
- We can make do with 3 or 4 tapes but we can increase efficiency with more tape drives.

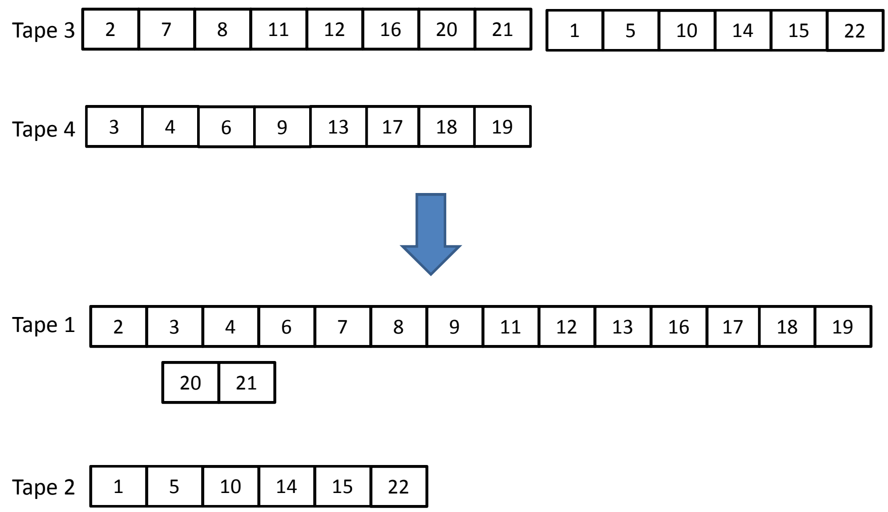
Creating The Runs



Merging The Runs



Merging The Runs



Balanced Merging

- Since the scratch tapes receive the same number of records, this is a ***balanced multiway merge***.
- If we have $2d$ drives, the total cost will be:

$$n \log_d (n/m)$$

Which Internal Sort?

- What sort do we use internally?
  - A quicksort won't work well if the data is already sorted.
  - A mergesort may tie up too much memory
  - A heapsort may offer the best compromise:
    - Efficiency is always $O(n \log n)$
    - It's done inplace.

What Wrong with Balanced Merging?

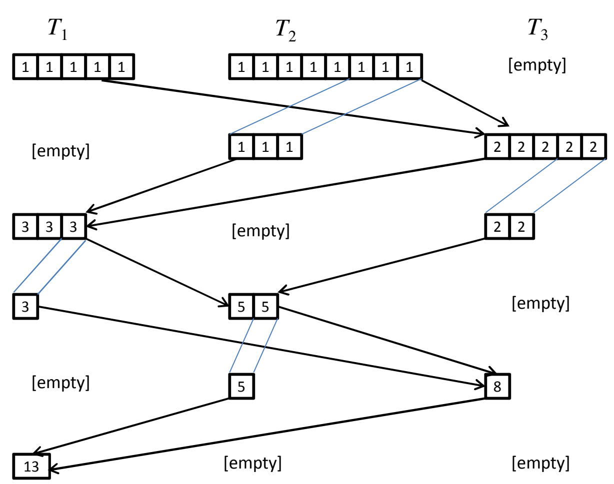
- Balanced merging uses many tapes.
- A p-way merge will need $2p$ tape in the ideal case.
- We could get away with p+1 tapes but we would need to keep distributing the output files onto the other p tapes.

Why Polyphase Merging?

- Let's assume that we have 3 tapes (T_1, T_2, T_3) and we merge in the following sequence:
  1. Sort and distribute the records onto T_1 and T_2.
  2. Merge T_1 and T_2 onto T_3 leaving some on T_2.
  3. Merge T_2 and T_3 onto T_1 leaving some on T_3.
  4. Merge T_3 and T_1 onto T_2 leaving some on T_1.
  5. Merge T_1 and T_2 onto T_3 leaving some on T_2.

  – and so on…
- We are always left us two source tapes and one tape on which to place the merged files.

Polyphase Merge On 13 Runs



Efficiency of Polyphase Merging

- The balanced merge required 4 passes but went through ALL the data, while the polyphase merge required 5 passes but went through only part of the data.
- The balanced merge went through $4 \times 13 = 52$ runs.
- The polyphase merge went through $10 + 9 + 10 + 8 + 13 = 50$ runs

2 Questions About The Polyphase Merge

- What if the source file is not exactly F_n runs long?
- What if we have more than 3 tapes?

What If We Don't have F_n Runs?

- We have the sort ad distribute step include dummy runs of length 0.

What if we have more than 3 tapes?

- We start with the desired result and work backwards

T_1	T_2	T_3	T_4	Sum
1	0	0	0	1
0	1	1	1	3
1	0	2	2	5
3	2	0	4	9
7	6	4	0	17
0	13	11	7	31
13	0	24	20	57

What if we have more than 3 tapes?

- We can permute the rows so the empty tape is always at the end.

- If each row contains

  a b c d 0

The next row contains

  a+b a+c a+d 0

1	0	0	0
1	1	1	0
2	2	1	0
4	3	2	0
7	6	4	0
13	11	7	0
24	20	13	0

Polyphase Merge On 13 Runs