DATA STRUCTURE AND ALGORITHM

Dr. Khine Thin Zar
Professor
Computer Engineering and Information Technology Dept.
Yangon Technological University

Lecture 8

File Processing and External Sorting

Outlines of Class (Lecture 8)

- Introduction
- Primary versus Secondary Storage
- Disk Drives
- Buffers and Buffer Pools
- External Sorting
- External Mergesort Algorithm
- Replacement Selection
- Multiway Merging

Introduction

- Large amounts of information be stored and processed
- It cannot all fit into main Memory
- Reside on disk and be brought into main memory selectively for processing
- Main memory access is much faster than access to data stored on disk or other storage devices
- Many programmers do a poor job when it comes to file processing applications
- The fundamental issues relating to the design of algorithms and data structures for disk-based applications

Primary versus Secondary Storage

- Classified into primary or main memory and secondary or peripheral storage
- Primary memory: Random Access Memory (RAM)
- Secondary storage: hard disk drives, solid state drives, removable "USB" drives, CDs, and DVDs
- Secondary storage devices have two other advantages over RAM memory:
  - ✓ persistent,
  - ✓ volatile

Disk Drives

- Logical file: a random access file stored on disk as a contiguous series of bytes
- Physical file: actually stored on disk is usually not a contiguous series of bytes
- File manager: responsible for taking requests for data from a logical file and mapping those requests to the physical location of the data on disk
- Direct access: it takes roughly equal time to access any record in the file
- Sequential access: require the tape reader to process data from the beginning of the tape until the desired position has been reached

Disk Drive Architecture

- Hard disk drive: composed of one or more round platters, stacked one on top of another and attached to a central spindle.
- Each usable surface of each platter is assigned a read/write head or I/O head
  - through which data are read or written
- A hard disk drive has several platters and several read/write heads
  - each head is attached to an arm
- Track: the data on a single platter that are accessible to any one position of the head for that platter
- Cylinder: the collection of all tracks that are a fixed distance from the spindle

Disk Drive Architecture (Cont.)

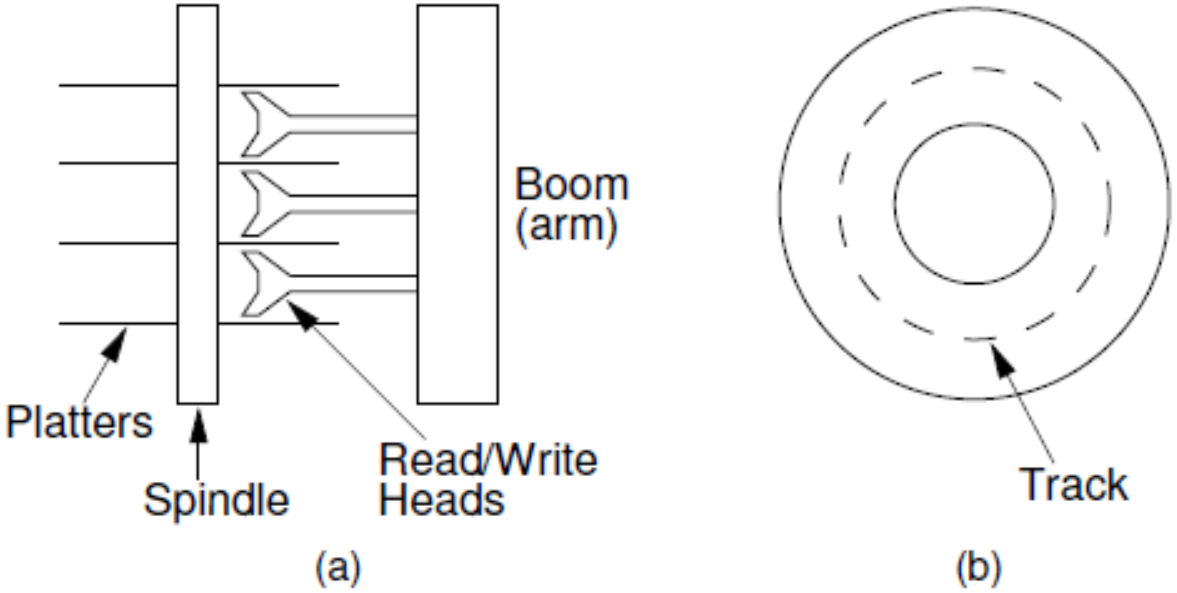

Figure: (a) A typical disk drive arranged as a stack of platters.
(b) One track on a disk drive platter

Buffers and Buffer Pools

- Once a sector is read, its information is stored in main memory (buffering or caching the information)
- Caching information in memory is extended to multiple buffers
- The process of using buffers as an intermediary between a user and a disk file is called buffering the file.
- The information stored in a buffer is called a page.
- The collection of buffers is called a buffer pool.

The Programmer's View of Files

- Random access processes records in an order independent of their logical order within the file.
- Sequential access processes records in order of their logical appearance within the file.
- Sequential processing requires less seek time if the physical layout of the disk file matches its logical layout

The Programmer's View of Files (Cont.)

- The following methods can be used to manipulate information in the file.
- **open(char *name, openmode flags)**: Open file name **name** for processing. **Flags** control various details such as whether the file permits reading, writing, or both; and whether its pre-existing contents should be deleted.
- **read(char *buff, int count)**: Read **count** bytes from the current position in the file. The current file position moves forward as the bytes are read. The bytes are read into array **buff**, which must be at least **count** bytes long.
- **write(char *buff, int count)**: Write **count** bytes at the current position in the file (overwriting the bytes already at that position). The current file position moves forward as these bytes are written. The bytes to be written come from array **buff**.

The Programmer's View of Files (Cont.)

- **seekg(int pos)** and **seekp(int pos)**: Move the current position in the file to **pos**. This allows bytes at arbitrary places within the file to be read or written. There are actually two "current" positions: one for reading and one for writing. Function **seekg** changes the "get" or read position, while function **seekp** changes the "put" or write position.
- **close()**: Close a file at the end of processing.

External Sorting

- Consider the problem of sorting collections of records too large to fit in main memory.
- Because the records must reside in peripheral or external memory (external sorts)
- The internal sorts sorted the records stored in main memory.
- When a collection of records is too large to fit in main memory, the only practical way to sort it is
  - ✓ to read some records from disk,
  - ✓ do some rearranging, then
  - ✓ write them back to disk.
  - ✓ This process is repeated until the file is sorted.

The Requirement of External Sorting

- Consists of two phases
- Sorting phase: in which a large amount of data is sorted in an intermediate file.
- Merge phase: the sorted files are combined into a single larger file.

Simple Approaches to External Sorting

- Read the entire file into virtual memory and run an internal sorting method
- Allows the virtual memory manager to use its normal buffer pool mechanism to control disk accesses
- Drawback: the size of virtual memory is usually limited to something much smaller than the disk space available.

External Mergesort Algorithm

1. Split the original file into two equal-sized run files.
2. Read one block from each run file into input buffers.
3. Take the first record from each input buffer, and write a run of length two to an output buffer in sorted order.
4. Take the next record from each input buffer, and write a run of length two to a second output buffer in sorted order.
5. Repeat until finished, alternating output between the two output run buffers. Whenever the end of an input block is reached, read the next block from the appropriate input file. When an output buffer is full, write it to the appropriate output file.

External Mergesort algorithm (Cont.)

6. Repeat steps 2 through 5, using the original output files as input files. On the second pass, the first two records of each input run file are already in sorted order. Thus, these two runs may be merged and output as a single run of four elements.

7. Each pass through the run files provides larger and larger runs until only one run remains.

External Mergesort Algorithm (Cont.)

- Input records are divided equally between two input files. The first runs from each input file are merged and placed into the first output file.
- The second runs from each input file are merged and placed in the second output file. Merging alternates between the two output files until the input files are empty.
- The roles of input and output files are then reversed, allowing the runlength to be doubled with each pass.

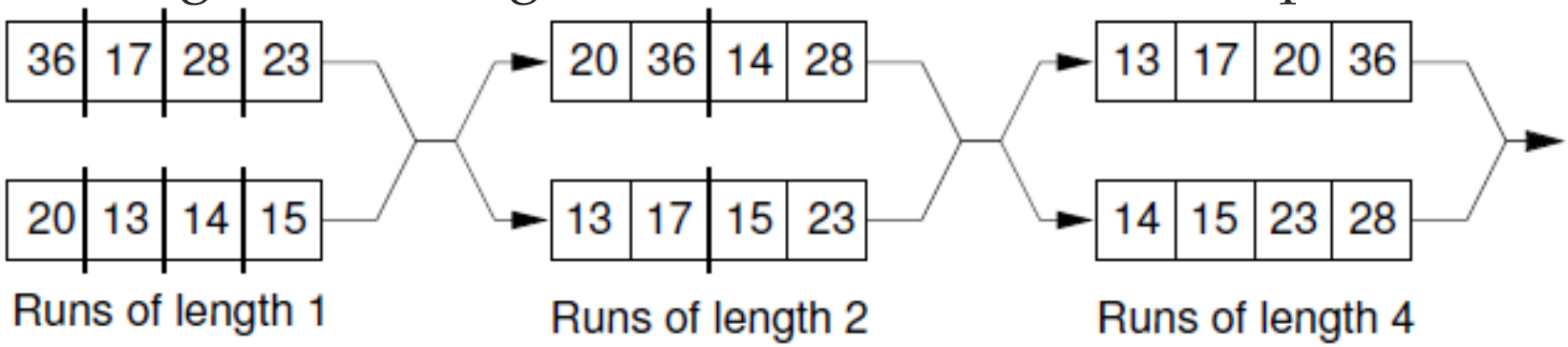

Figure: A simple external Mergesort algorithm.

Replacement Selection

- If the size of memory available for the array is M records, then the input file can be broken into initial runs of length M.
- Replacement selection creates runs of 2M records in length.
- Views RAM as consisting of an array of size M in addition to an input buffer and an output buffer.
- Takes the next record in sequential order from the input stream when needed, and outputs runs one record at a time to the output stream.
- Removes records from the input buffer one at a time until the buffer is empty.

Replacement Selection (Cont.)

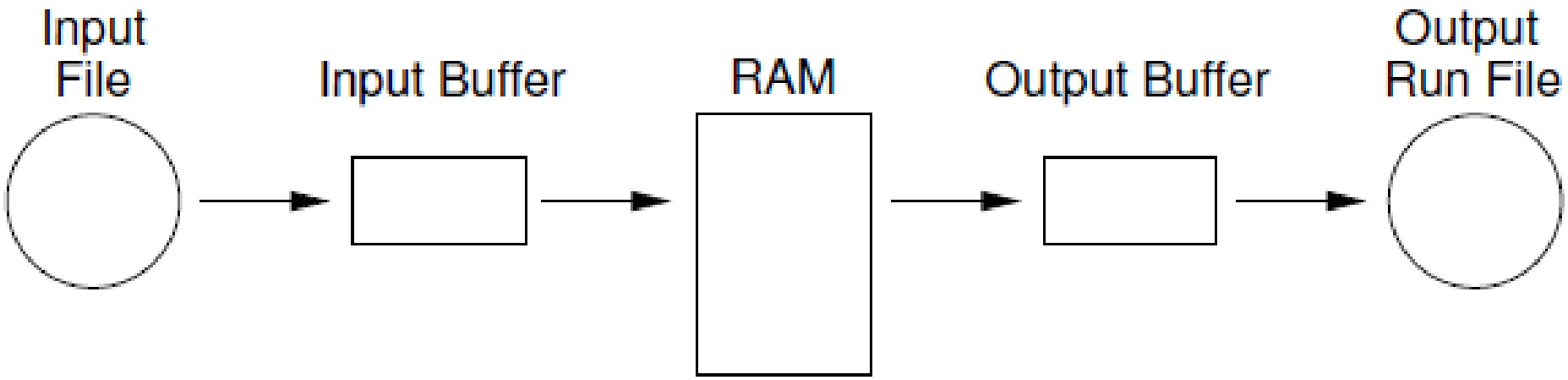

Figure: Overview of replacement selection

Replacement Selection (Cont.)

- Input records are processed sequentially.
- Initially RAM is filled with M records.
- As records are processed, they are written to an output buffer. When this buffer becomes full, it is written to disk.
- Meanwhile, as replacement selection needs records, it reads them from the input buffer.
- Whenever this buffer becomes empty, the next block of records is read from disk.

Replacement Selection (Cont.)

- Replacement selection works as follows. Assume that the main processing is done in an array of size M records.

1. Fill the array from disk. Set LAST = M - 1.

2. Build a min-heap

3. Repeat until the array is empty:

(a) Send the record with the minimum key value (the root) to the output buffer.

(b) Let R be the next record in the input buffer. If R's key value is greater than the key value just output ...

i. Then place R at the root.

ii. Else replace the root with the record in array position LAST, and place R at position LAST. Set LAST = LAST - 1.

(c) Sift down the root to reorder the heap.

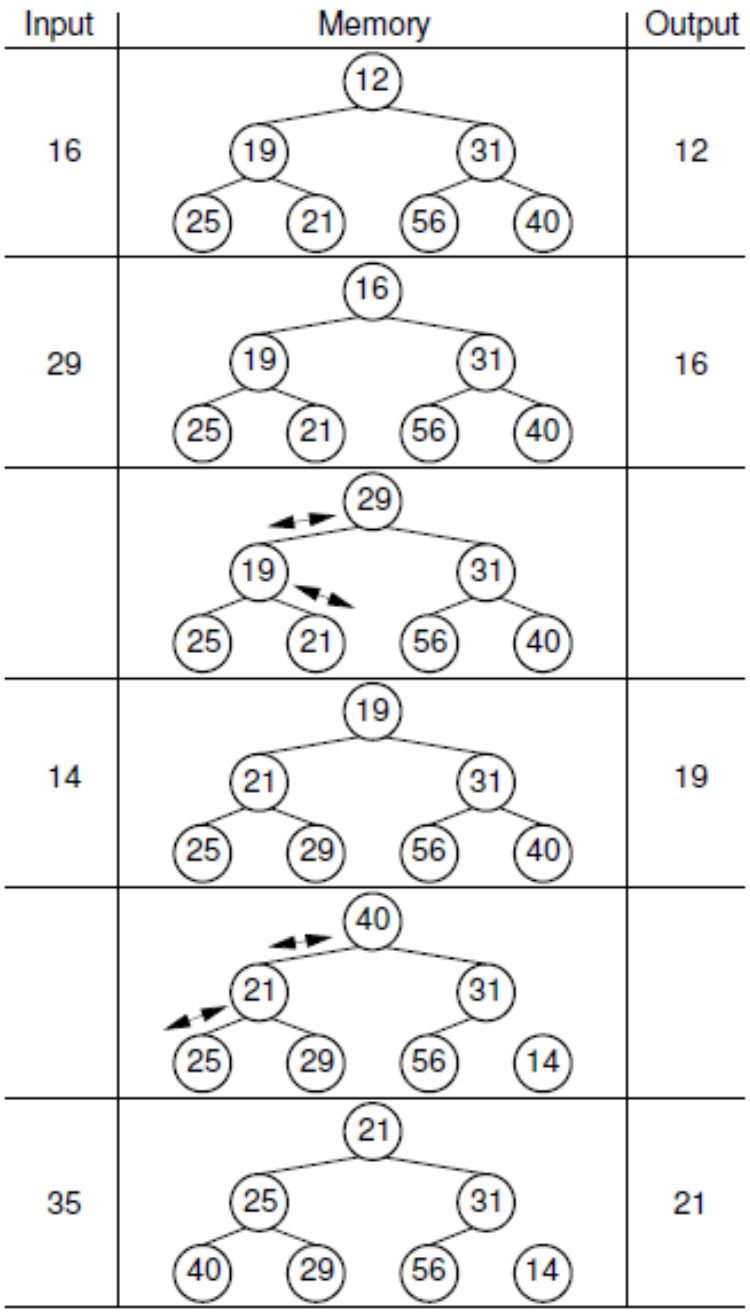

Figure: Replacement selection example.
After building the heap, root value 12 is output and incoming value 16 replaces it. Value 16 is output next, replaced with incoming value 29. The heap is reordered, with 19 rising to the root. Value 19 is output next. Incoming value 14 is too small for this run and is placed at end of the array, moving value 40 to the root. Reordering the heap results in 21 rising to the root, which is output next.

Multiway Merging

- The second stage of a typical external sorting algorithm merges the runs created by the first stage.
- R runs to merge.
- Twoway merging does not make good use of available memory.
- Sequential process on the two runs, only one block of records per run need be in memory at a time.
- Better use of this space and reduce the number of passes needed to merge
- Multiway merging can reduce the number of passes required

Multiway Merging (Cont.)

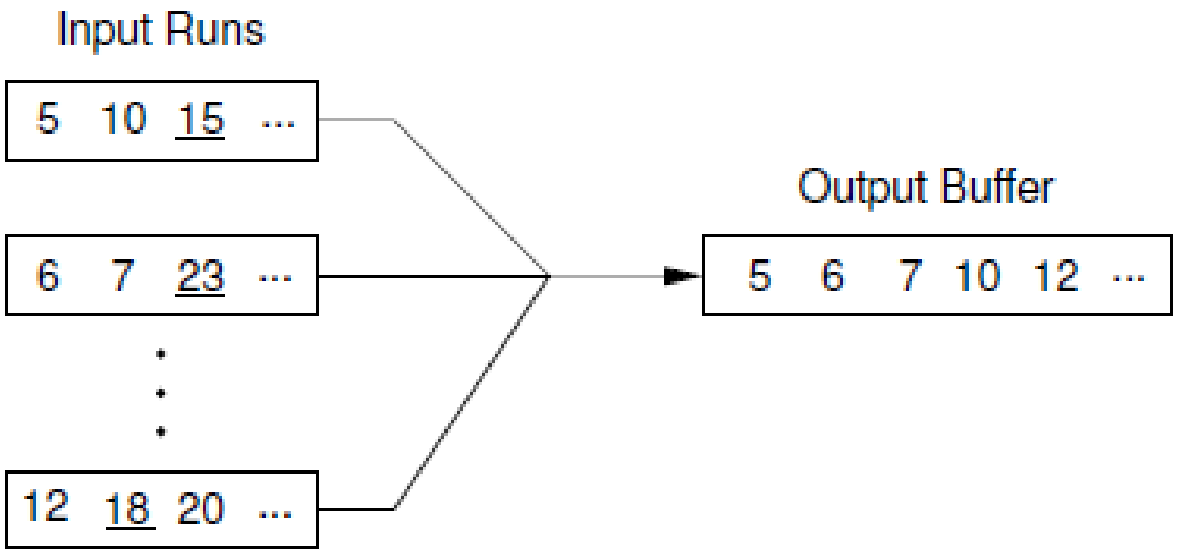

Figure: Illustration of multiway merge.

Summary

- A good external sorting algorithm will seek to do the following:
- Make the initial runs as long as possible.
- At all stages, overlap input, processing, and output as much as possible.
- Use as much working memory as possible. Applying more memory usually speeds processing. In fact, more memory will have a greater effect than a faster disk. A faster CPU is unlikely to yield much improvement in running time for external sorting, because disk I/O speed is the limiting factor.

Comparison of the Running Time

File Size (Mb)	Sort 1	Sort 2 Memory size (in blocks) 2	4	16	256	Sort 3 Memory size (in blocks) 2	4	16
1	0.61 4,864	0.27 2,048	0.24 1,792	0.19 1,280	0.10 256	0.21 2,048	0.15 1,024	0.13 512
4	2.56 21,504	1.30 10,240	1.19 9,216	0.96 7,168	0.61 3,072	1.15 10,240	0.68 5,120	0.66* 2,048
16	11.28 94,208	6.12 49,152	5.63 45,056	4.78 36,864	3.36 20,480	5.42 49,152	3.19 24,516	3.10 12,288
256	220.39 1,769K	132.47 1,048K	123.68 983K	110.01 852K	86.66 589K	115.73 1,049K	69.31 524K	68.71 262K

Figure: A comparison of three external sorts on a collection of small records for files of various sizes. Each entry in the table shows time in seconds and total number of blocks read and written by the program. File sizes are in Megabytes. For the third sorting algorithm, on a file size of 4MB, the time and blocks shown in the last column are for a 32-way merge (marked with an asterisk). 32 is used instead of 16 because 32 is a root of the number of blocks in the file (while 16 is not), thus allowing the same number of runs to be merged at every pass.

Next Week Lecture (Week 9)

Lecture 9: Searching

Thank you!