

Primary vs. Secondary Storage

Primary Storage: Main memory (RAM)

Secondary Storage: Peripheral devices

- Disk Drives
- Tape Drives

Medium	Price	Price per Mbyte
32MB RAM	\$225	\$7.00/MB
1.4MB floppy disk	\$.50	\$0.36/MB
2.1GB disk drive	\$210	\$0.10/MB
1GB JAZ cassette	\$100	\$0.10/MB
2GB cartridge tape	\$20	\$0.01/MB

RAM is usually volatile.

RAM is about 1/4 million times faster than disk.

Golden Rule of File Processing

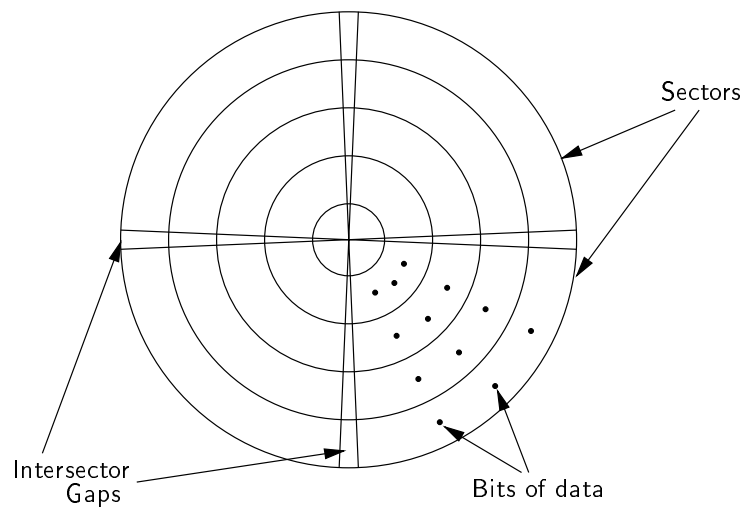
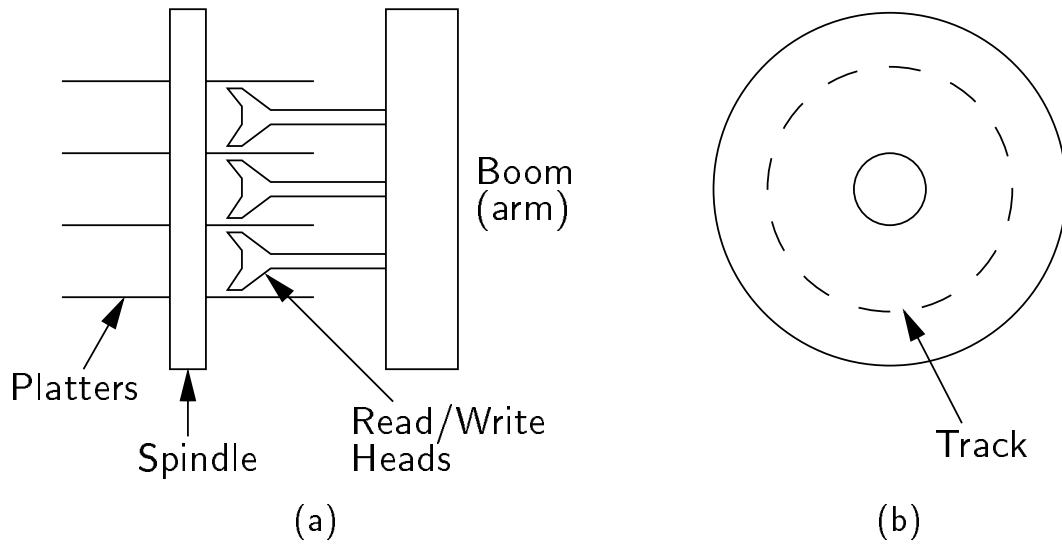
Minimize the number of disk accesses!

1. Arrange information so that you get what you want with few disk accesses.
2. Arrange information so minimize future disk accesses.

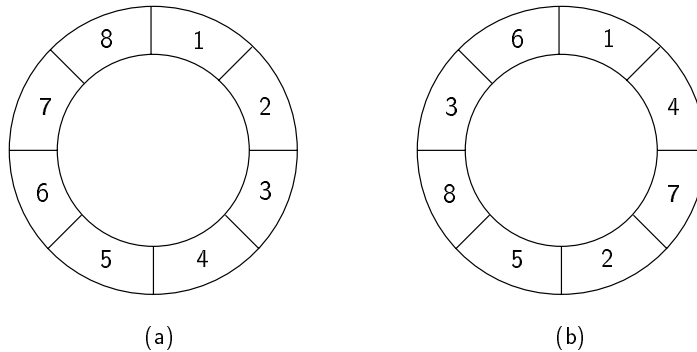
An organization for data on disk is often called a **file structure**.

Disk based space/time tradeoff: Compress information to save processing time by reducing disk accesses.

Disk Drives



Sectors



A sector is the basic unit of I/O.

Interleaving factor: Physical distance between logically adjacent sectors on a track, to allow for processing of sector data.

Locality of Reference: If a record is read from disk, the next request is likely to come from near the same place in the file.

Cluster: Smallest unit of file allocation – several sectors.

Extent: A group of physically contiguous clusters.

Internal Fragmentation: Wasted space within a sector if record size does not match sector size, or wasted space within a cluster if file size is not a multiple of cluster size.

Factors affecting disk access time

1. **Seek time**: time for I/O head to reach desired track. Largely determined by distance between I/O head and desired track.

Seek time: $f(n) = t * n + s$ where t is time to traverse one track and s is startup time for the I/O head.

2. **Rotational delay** or **latency**: time for data to rotate to I/O head position.

At 3600 RPM, one half rotation of the disk:
 $16.7/2 \text{ msec} = 8.3 \text{ msec}$.

3. **Transfer time**: time for data to move under the I/O head.

Number of sectors read / Number of sectors per track * 16.7 msec

Disk Access Cost Example

675 Mbyte disk drive

- 15 platters \Rightarrow 45 Mbyte/platter
- 612 tracks/platter
- 150 sectors/track \Rightarrow 512 bytes/sector
- 8 sectors/cluster (4K bytes/cluster) \Rightarrow 18 clusters/track
- Interleaving factor of 3 \Rightarrow 3 revolutions to read one track (50.1 msec)

How long to read a file of 128 Kbytes divided into 256 records of 512 bytes?

Number of Clusters:

If file fills minimum number of tracks:

150 sectors of one track, 106 of the next

Total time:

$$612/3 * 0.08 + 3 + 3.5 * 16.7 + 0.08 + 3 + 3.5 * 16.7 = 139.3 \text{ msec.}$$

If clusters are spread randomly across disk:

$$32 * \left(\frac{612}{3} * 0.08 + 3 + 16.7/2 + \frac{24}{150} * 16.7 \right) = 32 * 30.3 = 969.6 \text{ msec.}$$

Magnetic Tape

Example: 9 track tape at 6250 bytes per inch (bpi).

At 2400 feet, this yields 170 Mbytes for \$20, or \$0.12/Mbyte.

Workstation/PC cartridge tape is similar.

Magnetic tape requires sequential access.

Magnetic tape has two speeds:

- High speed for “skipping.”
- Low speed for “reading.”

Interblock Gap is space required for I/O head to recognize beginning of a record at high speed. This is a significant amount of space compared to typical record size.

Blocking Factor: Number of records in a block between gaps.

Buffers

Read time for one track:

$$612/3 * 0.08 + 3 + 3.5 * 16.7 = 77.8 \text{ msec.}$$

Read time for one sector:

$$612/3 * 0.08 + 3 + 16.7/2 + 16.7/150 = 27.8 \text{ msec.}$$

Read time for one byte:

$$612/3 * 0.08 + 3 + 16.7/2 = 27.7 \text{ msec.}$$

Nearly all disk drives read/write one sector at every I/O access.

- Also called a page.

The information in a sector is stored in a buffer or cache.

If next I/O access is to same buffer, then no need to go to disk.

There are usually one or more input buffers and one or more output buffers.

Buffer Pools

A series of buffers used by an application to cache disk data is called a **buffer pool**.

Virtual memory uses a buffer pool to imitate greater RAM memory by actually storing information on disk and “swapping” between disk and RAM.

Organization for buffer pools: which one to use next?

- First-in, First-out: Use the first one on the queue.
- Least Frequently Used (LFU): Count buffer accesses, pick the least used.
- Least Recently Used (LRU):
Keep buffers on linked list.
When a buffer is accessed, bring to front.
Reuse the one at the end.

Double Buffering: Read data from disk for next buffer while CPU is processing previous buffer.

Programmer's View of Files

Logical view of files:

- An array of bytes.
- A file pointer marks the current position.

Three fundamental operations:

- Read bytes from current position (move file pointer).
- Write bytes to current position (move file pointer).
- Set file pointer to specified byte position.

Java File Functions

`RandomAccessFile(String name, String mode)`

`close()`

`read(byte[] b)`

`write(byte[] b)`

`seek(long pos)`

External Sorting

Problem: Sorting data sets too large to fit in main memory.

- Assume data stored on disk drive.

To sort, portions of the data must be brought into main memory, processed, and returned to disk.

An external sort should minimize disk accesses.

Model of External Computation

Secondary memory is divided into equal-sized **blocks** (512, 2048, 4096 or 8192 bytes are typical sizes).

The basic I/O operation transfers the contents of one disk block to/from main memory.

Under certain circumstances, reading blocks of a file in sequential order is more efficient.
(When?)

Typically, the time to perform a single block I/O operation is sufficient to Quicksort the contents of the block.

Thus, our primary goal is to minimize the number of block I/O operations.

Most workstations today must do all sorting on a single disk drive.

Key Sorting

Often records are large while keys are small.

- Ex: Payroll entries keyed on ID number.

Approach 1: Read in entire records, sort them, then write them out again.

Approach 2: Read only the key values, store with each key the location on disk of its associated record.

If necessary, after the keys are sorted the records can be read and re-written in sorted order.

External Sort: Simple Mergesort

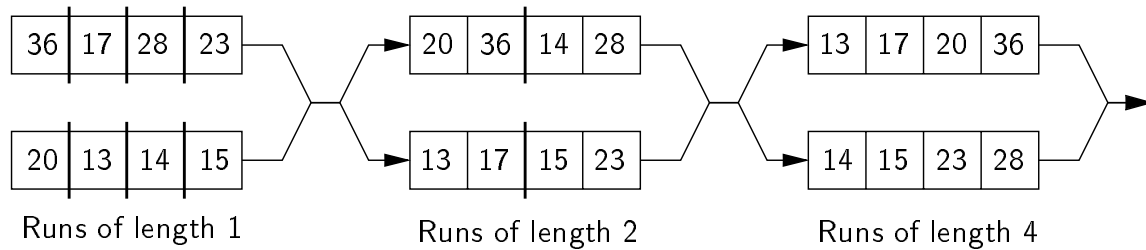
Quicksort requires random access to the entire set of records.

Better: Modified Mergesort algorithm

- Process n elements in $\Theta(\log n)$ passes.
1. Split the file into two files.
 2. Read in a block from each file.
 3. Take first record from each block, output them in sorted order.
 4. Take next record from each block, output them to a *second* file in sorted order.
 5. Repeat until finished, alternating between output files. Read new input blocks as needed.
 6. Repeat steps 2-5, except this time the input files have groups of two sorted records that are merged together.
 7. Each pass through the files provides larger and larger groups of sorted records.

A group of sorted records is called a run.

Problems with Simple Mergesort



Is each pass through input and output files sequential?

What happens if all work is done on a single disk drive?

How can we reduce the number of Mergesort passes?

In general, external sorting consists of two phases:

1. Break the file into initial runs.
2. Merge the runs together into a single sorted run.

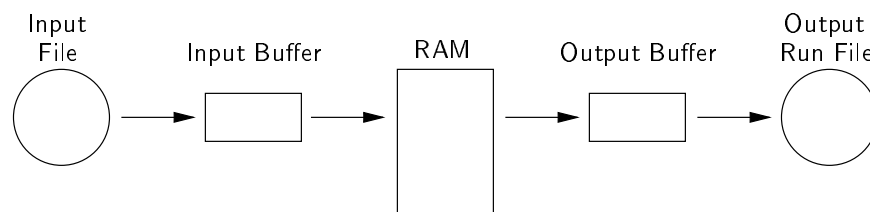
Breaking a file into runs

General approach:

- Read as much of the file into memory as possible.
- Perform an in-memory sort.
- Output this group of records as a single run.

Replacement Selection

1. Break available memory into an array for the heap, an input buffer and an output buffer.
2. Fill the array from disk.
3. Make a min-heap.
4. Send the smallest value (root) to the output buffer.
5. If the next key in the file is greater than the last value output, then
 Replace the root with this key.
else
 Replace the root with the last key in the array.
Add the next record in the file to a new heap (actually, stick it at the end of the array).



Example of Replacement Selection

Input	Memory	Output
16		12
29		16
14		19
35		21

Benefit from Replacement Selection

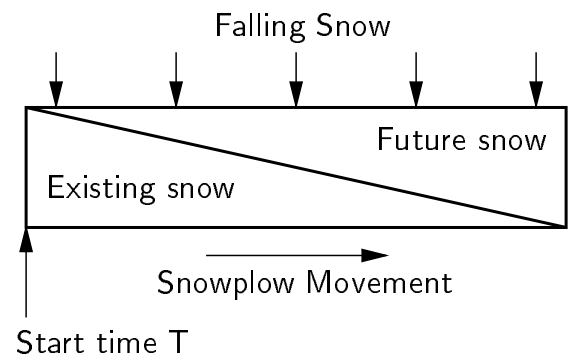
Use double buffering to overlap input, processing and output.

How many disk drives for greatest advantage?

Snowplow argument:

- A snowplow moves around a circular track onto which snow falls at a steady rate.
- At any instant, there is a certain amount of snow S on the track. Some falling snow comes in front of the plow, some behind.
- During the next revolution of the snowplow, all of this is removed, plus $1/2$ of what falls during that revolution.
- Thus, the plow removes $2S$ amount of snow.

Is this always true?



Simple Mergesort may not be Best

Simple Mergesort: Place the runs into two files.

- Merge the first two runs to output file, then next two runs, etc.

This process is repeated until only one run remains.

- How many passes for r initial runs?

Is there benefit from sequential reading?

Is working memory well used?

Need a way to reduce the number of passes.

Multiway Merge

With replacement selection, each initial run is several blocks long.

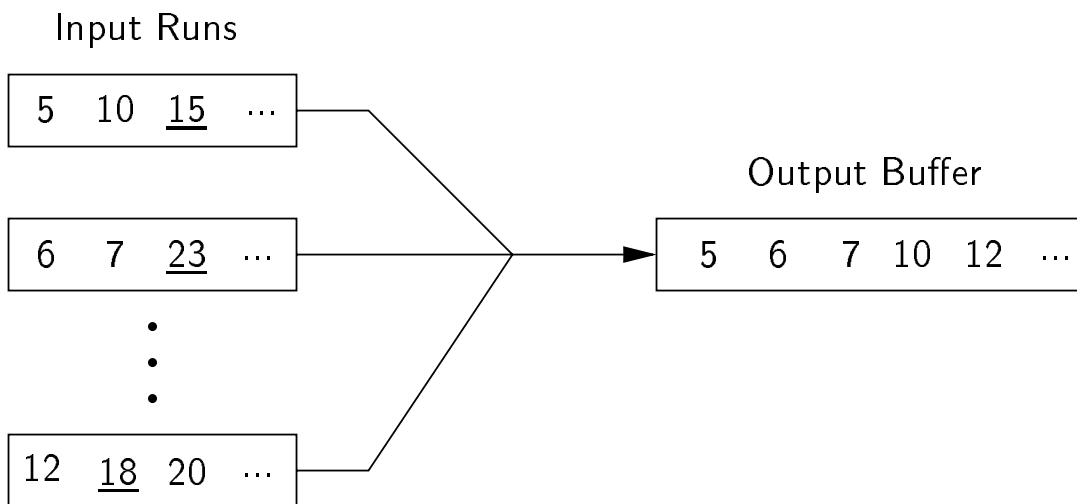
Assume that each run is placed in a separate disk file.

We could then read the first block from each file into memory and perform an r -way merge.

When a buffer becomes empty, read a block from the appropriate run file.

Each record is read only *once* from disk during the merge process.

In practice, use only one file and seek to appropriate block.



Limits to Single Pass Multiway Merge

Assume working memory is b blocks in size.

How many runs can be processed at one time?

The runs are $2b$ blocks long (on average).

How big a file can be merged in one pass?

Larger files will need more passes – but the run size grows quickly!

This approach trades $\Theta(\log b)$ (possibly) sequential passes for a single or a very few random (block) access passes.

General Principals of External Sorting

In 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.
- If possible, use additional disk drives for more overlapping of processing with I/O, and allow for more sequential file processing.