

# COMP 282

## Lecture 10 External Methods: Sorting

# Motivation

- Many sources of data are far larger than electronic memory can hold.
- Databases for instance rapidly grow beyond such capacity.
  - Databases are also good example of large stores of information that benefit greatly from being maintained according to some criteria.
    - Sorted,
    - Index,
    - Organized, etc.

# Sorting

- In previous classes several techniques for sorting data were presented.
- $O(n^2)$  and  $O(n \lg(n))$  were shown to be possible
- Quicksort and mergesort were shown to be  $O(n \lg(n))$ 
  - Quicksort is generally slightly more efficient both in terms of memory consumption and in terms of speed.
  - But... a modification of merge sort will allow you to sort arbitrarily large files of information.

# Mergesort

- $O(n \lg(n))$  behavior is very advantageous because we are dealing with extremely large amounts of data to sort.
- The fact that merge sort requires twice as much space (memory) than quicksort to perform its task will not be an issue because we are dealing with an environment that provides excessive amounts of space. (disk)

# buffers

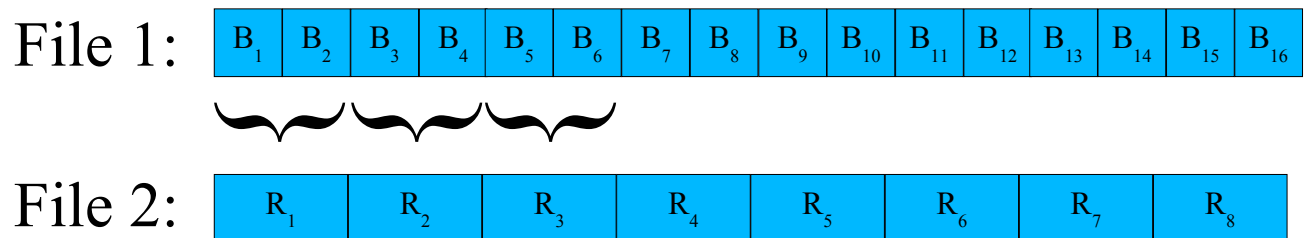
- The modification to mergesort is that it will use three buffers each the size of a block of records.
- The buffers can be named and thought of as:
  - In1
  - In2
  - Out
- The merge operation of merge sort will attempt to merge the two in buffers into the out buffer.
  - (but note that the out buffer is not twice as large as the in buffers )

# First pass: block sorting

- Merging of data can only occur after the pieces being merge have been sorted.
- This requires a first pass on the file takes each individual block and sorts the records within that block.
  - Any sort algorithm can be used perform this sort.
- Once the pass is completed. We can merge adjacent pairs of blocks into larger blocks.

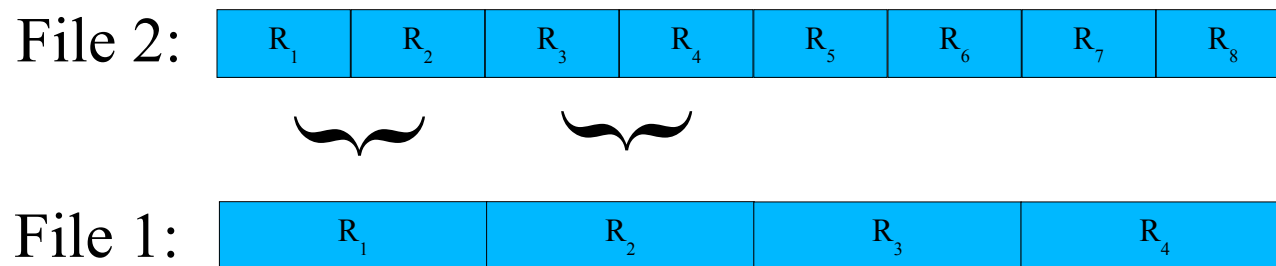
# First merge pass

- Now that the data is sorted adjacent pairs of blocks (and later regions) can be merged to form larger sorted regions until the entire file is sorted.
- In the first pass two blocks are merged.
- In1 and In2 fill exactly once and Out fills and empties exactly twice.



# Second pass

- Adjacent regions can then be merged:
- Second pass:
  - In1 and In2 each fill and empty exactly twice
  - Out will fill and write out exactly four times



# Repetition

- This process of merging will repeat until the file consists of a single, sorted region.
- The current data keeps shifting between the two files File1 and File2.
- At each pass the number of times that the In buffers will fill and empty doubles from the previous time. As does the number of times the Out buffer will fill and write out.

# Buffer Abstraction

- The buffers are important.
- Their fixed size makes it necessary that they be refilled when exhausted (In buffers) and that they be written out when full (Out Buffer)
- This process can be abstracted rather well.
  - If abstracted well the merge code doesn't need to keep track of buffer status. It can just view the world as one big region even though the regions are really being processed as multiple block fetches.

# InBuffer

- Class InBuffer

```
{
    // create a buffer that we supply at most
    // maxrecors from a file starting at the nth
    // record
    public Buffer(String file, int blocksize,
                 int startingblock, int maxblocks);

    // retrieve a new block
    private underflow();

    // return the next record
    public SomeClass read();
}
```

# OutBuffer

- Class OutBuffer

```
{  
    // create an outbuffer that holds at most  
    // blocksize records and appends to file.  
    public Buffer(String file, int blocksize);  
  
    // cause the entire block to be written  
    private overflow();  
  
    // cause what the currently held records to  
    // be written  
    public flush();  
  
    // add a record to the buffered block  
    public write(SomeClass record);  
}
```

# Non-orthogonal

- An orthogonal programming language is one in which something can only be done one way.
- Java is non-orthogonal: There are at least three different ways to implement a loop structure.
- Perl is really, really non-orthogonal.
- The buffer abstraction is also non-orthogonal. The class interfaces presented here are just one example of how the buffers can be implemented.