External Sorting

Introduction

External sorting refers to the sorting of a file that is on disk (or tape). Internal sorting refers to the sorting of an array of data that is in RAM. The main concern with external sorting is to minimize disk access since reading a disk block takes about a million times longer than accessing an item in RAM (according to Shaffer -- see the reference at the end of this document).

Perhaps the simplest form of external sorting is to use a fast internal sort with good **locality of reference** (which means that it tends to reference nearby items, not widely scattered items) and hope that your operating system's virtual memory can handle it. (Quicksort is one sort algorithm that is generally very fast and has good locality of reference.) If the file is too huge, however, even virtual memory might be unable to fit it. Also, the performance may not be too great due to the large amount of time it takes to access data on disk.

Methods

Most external sort routines are based on mergesort. They typically break a large data file into a number of shorter, sorted **runs**. These can be produced by repeatedly reading a section of the data file into RAM, sorting it with ordinary quicksort, and writing the sorted data to disk. After the sorted runs have been generated, a merge algorithm is used to combine sorted files into longer sorted files. The simplest scheme is to use a 2-way merge: merge 2 sorted files into one sorted file, then merge 2 more, and so on until there is just one large sorted file. A better scheme is a multiway merge algorithm: it might merge perhaps 128 shorter runs together.

Analysis

According to Shaffer, a multiway merge using half a megabyte of RAM and a disk block size of 4 KB could hold 128 disk blocks in RAM at once. This would allow 128 runs to be merged together in one pass. The average initial run size would be 1 MB. (See Shaffer on how that can be obtained with only 1/2 MB of RAM.) A file of size 128 MB could be sorted in 2 passes (one to build the initial runs and one to merge them). A file of size 16 gigabytes could be sorted in just 3 passes.

Note that you do not want to jump back and forth between 2 or more files in trying to merge them (while writing to a third file). This would likely produce a lot of time-consuming disk seeks. Instead, on a single-user PC, it is better to read a block of each of the 2 (or more) files into RAM and carry out the merge algorithm there, with the output also kept in a buffer in RAM until the buffer is filled (or we are out of data) and only then writing it out to disk. When the merge algorithm exhausts one of the blocks of data, refill it by reading from disk another block of the associated file. This is called **buffering**. On a larger machine where the disk drive is being shared among many users, it may not make sense to worry about this as the read/write head is going to be seeking all over the place anyway.

Practical Data

Shaffer presents the following practical data concerning external sorting. In this experiment a 4 MB

file was sorted on a particular computer. A simple mergesort that did *not* build initial sorted runs took 451 seconds. A 2-way mergesort that used initial runs of 128 KB took only 160 seconds. A multiway mergesort that used the same initial runs took only 103 seconds. Clearly, using initial sorted runs dramatically speeds up the sorting.