# External Sort: Heapsort

## Assignment:

You will implement an external sorting algorithm for binary data. The input data file will consist of many 4-byte records, with each record consisting of two 2-byte (short) integer values in the range 1 to 30,000. The first 2-byte field is the key value (used for sorting) and the second 2-byte field contains a data value. The input file is guaranteed to be a multiple of 4096 bytes. All I/O operations will be done on blocks of size 4096 bytes (i.e., 1024 logical records).

**Warning**: The data file is a **binary** file, not a text file.

Your job is to sort the file (in ascending order), using a modified version of the Heapsort. The modification comes in the interaction between the Heapsort algorithm and the file storing the data. The heap array will be the file itself, rather than an array stored in memory. All accesses to the file will be mediated by a buffer pool. The buffer pool will store 4096-byte blocks (1024 records). The buffer pool will be organized using the Least Recently Used (LRU) replacement scheme. See Section 8.3 in the book for more information about buffer pools.

## Design Considerations:

The primary design concern for this project will be the interaction between the logical heap as viewed by the Heapsort algorithm, and the physical representation of the heap as implemented by the disk file mediated by the buffer pool. You should pay careful attention to the interface that you design for the buffer pool, since you will be using this again in Project 4. In essence, the disk file will be the heap array, and all accesses to the heap from the Heapsort algorithm will be in the form of requests to the buffer pool for specific blocks of the file.

## Invocation and I/O Files:

The program will be invoked from the command-line as:

```
heapsort <data-file-name> <numb-buffers> <stat-file-name>
```

The data file `<data-file-name>` is the file to be sorted. The sorting takes place in that file, so this program does modify the input data file. Be careful to keep a copy of the original when you do your testing. The parameter `<numb-buffers>` determines the number of buffers allocated for the buffer pool. This value will be in the range 1–20. The parameter ¡stat-file-name¿ is the name of a file your program will generate containing runtime statistics; see below for more information.

At the end of your program, the data file (on disk) should be sorted. Do not forget to flush buffers from your bufferpool as necessary.

In addition to sorting the data file, you must report some information about the execution of your program.

1. You will need to report part of the sorted data file to standard output. Specifically, your program will print the first record from each 4096-byte block, in order, from the sorted data file. The records are to be printed 8 records to a line (showing both the key value and the data value for each record), the values separated by whitespace and formatted into columns. This program output must appear EXACTLY as described; ANY deviation from this requirement will result in a significant deduction in points.

2. You will generate and output some statistics about the execution of your program. Formatting does not matter as long as we can tell what each statistic is. Write these statistics to `<stat-file-name>`. Make sure your program DOES NOT overwrite `<stat-file-name>` each time it is run; instead, have it append new statistics to the end of the file. The information to write is as follows.

   (a) The name of the data file being sorted.

   (b) The number of cache hits, or times your program found the data it needed in a buffer and did not have to go to the disk.

   (c) The number of cache misses, or times your program did not find the data it needed in a buffer and had to go to the disk.

   (d) The number of disk reads, or times your program had to read a block of data from disk into a buffer.

   (e) The number of disk writes, or times your program had to write a block of data to disk from a buffer.

   (f) The time that your program took to execute the heapsort. Put two calls to the standard Java timing method "`System.currentTimeMillis()`'" in your program, one at the beginning and another at the end. This method returns a long value. The difference between the two values will be the total runtime in milliseconds. You should ONLY time the sort, and not the program output as described above.