CSc 345 — Analysis of Discrete Structures
(McCann)

Last Revised: October 2013

Multiway External Merge Sort

Background

When data to be sorted doesn't fit in main memory, an external sorting algorithm can do the job. Such algorithms have two main steps: Create initial sorted runs, then merge those runs into one. The longer we can make the initial runs, and the more runs we can merge at a time, the fewer passes (and secondary storage operations) will be needed, and the sooner the sort will complete.

The Algorithm

This algorithm summary omits many details in the interests of space and intellectual investigation. Assume that an unordered file of records is given, and a quantity (`B`, where `B` ≥ 3) of buffer pages is provided:

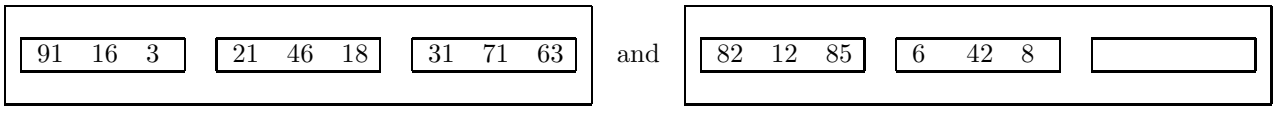
```
1           // Create the initial collection of sorted runs
2
3   Pass 0:       repeat
4                     Fill all B buffers with blocks from the file
5                     Sort the records in the buffers in-place
6                     Output the sorted run of records
7                 until the input file is exhausted
8
9           // Now use B-1 buffers for input and 1 for output
10
11  Passes 1-n:   while the number of sorted runs is > 1:
12                    while the input file still has unread runs:
13                        repeat
14                            Read in next block from each of the next B-1 runs
15                            Merge those blocks into the remaining buffer,
16                                writing it out as it fills, and
17                                replacing input blocks as content is exhausted
18                        until the longest of those B-1 runs is exhausted
19                    end while
20                end while
```

Example

Assume that we can store three records per file block, and there are `B` = 3 available buffers. We wish to sort this collection of key values:

```
91  16  3  21  46  18  31  71  63  82  12  85  6  42  8
```

Pass 0: Fill buffers, sort, output the sorted run, and repeat until each key is in a sorted run:

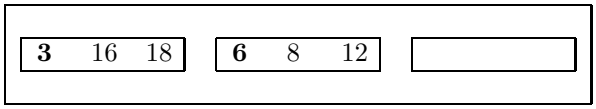


⇓ ⇓

(3 16 18 21 31 46 63 71 91) (6 8 12 42 82 85)

(Continued...)

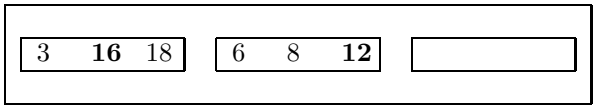
Pass 1: This is the only merging pass, because there are two sorted runs and $B - 1 = 2$. To start, bring in the first block of each sorted run. Current [window] positions within runs are shown on the right:




```
([3 16 18] 21 31 46 63 71 91)

([6  8 12] 42 82 85)
```

Merge values from the input buffers to the output buffer until the output is full or an input is exhausted. Here the output buffer filled first, it was written, and 16 & 12 (in bold) are remembered as the next to be compared:

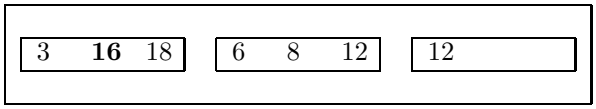



```
([3 16 18] 21 31 46 63 71 91)

([6  8 12] 42 82 85)

Output:  (3  6  8 ...
```

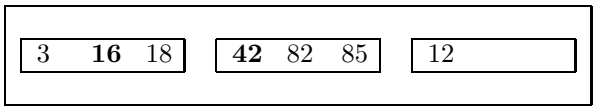
After comparing 16 and 12, the second input buffer is exhausted. We bring in the next window of data from the same run and continue ('before' and 'after' pictures are shown):




```
([3 16 18] 21 31 46 63 71 91)

([6  8 12] 42 82 85)

Output:  (3  6  8 ...
```

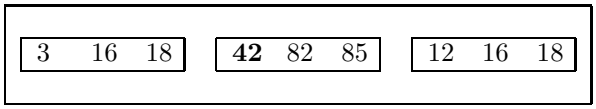



```
([3 16 18] 21 31 46 63 71 91)

( 6  8 12 [42 82 85])

Output:  (3  6  8 ...
```

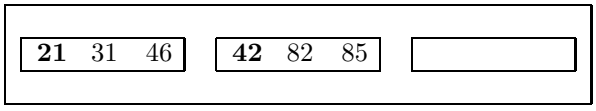
Soon, the output buffer fills again, but the first run's input buffer is exhausted at the same time. We write the output buffer and reload the first run's input buffer before continuing ('before' and 'after' pictures are shown):




```
([3 16 18] 21 31 46 63 71 91)

( 6  8 12 [42 82 85])

Output:  (3  6  8 ...
```

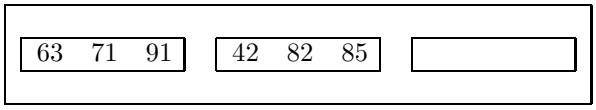



```
( 3 16 18 [21 31 46] 63 71 91)

( 6  8 12 [42 82 85])

Output:  (3  6  8  12  16  18...
```

Jumping well ahead: When one of the runs has been completely consumed (in this example, the second), we copy all remaining items from the remaining (first) run to the output and write it:




```
( 3 16 18 21 31 46 [63 71 91])

( 6  8 12 [42 82 85])

Output:  (3  6  8  12  16  18  21  31  42  46  63  71  82  85  91)
```

As these runs have been merged, and there are no more unread runs to merge on this pass, Pass 1 is complete. Further, as only one output run was produced, the sort is done.