```
External Sorting  (lecture notes)
----------------------------------------------------------------------------------------------------
Problem:  Sorting more data than will fit in main memory.
Objective:  Minimize the number of disk accesses



----------------------------------------------------------------------------------------------------
Sort Merge
-- - --- -

Initial runs:
 Sort as much data as will fit in memory, using some O(n log n) method like Quicksort.
 Write the sorted data out to a file as a "run".
 Repeat the process, writing out runs alternately to two output files.

Now we have two files with some number of sorted runs on them.  Given N data points, and room in memory for M data, we
have N/M runs on two files, or N/M/2 runs on each.

Repeat:
 - merge the first run from one file with the first run from the second file to produce
   a sorted run of length 2*M
 - write the new run to one of two new output files
 - switch output files
Until no more runs to merge.

Goes to disk log N/M times after initial runs are created.

----------------------------------------------------------------------------------------------------
What if log N/M is still too often?

There are several ways to decrease the number of times we go to disk.  One way is to increase the
size of each run.  How can we do this when we are already maximizing the use of primary memory?

Replacement selection with a heap.

Now we have N/2*M initial runs.

----------------------------------------------------------------------------------------------------
K-Way Sort Merge
- - - - - - - -
Reduce the number of disk accesses by increasing the number of output (hence input) files to k.
Problem:  Need a way to merge more than two runs.
Solution:  Use a heap.

----------------------------------------------------------------------------------------------------
Can we do better than this?

K-way needs 2 * K files open.  The bigger K is the fewer merges we need.  Can we maximize the number of
input files?

Polyphase Sort Merge
- - - - - - - - - -
 Distribute initial runs over K-1 output files.
Repeat:
 Merge K-1 runs and write to one output file.
 Repeat until one input file becomes empty.
 Make output file an input file and the empty input file the output file.
Until all input files are empty (one output run).

E.g.
8,8,8,7,0

After seven merges, the fourth input file is empty and the fifth
has 7 runs of length 4M

After one more run, the first three input files become empty, meaning we have
6 more merges to empty the fifth file.  Needs 8 passes in total.

If we are more careful in how we distribute the intial runs into the input files, we can
reduce this considerably.

E.g.
10,8,7,6,0

Or

9,8,7,7,0

How to distribute the initial runs over k input files?  Start from the last merge and
work your way up until the number of input runs is sufficient to cover the N data.
```

```
If N/M == 497 and k = 6

1 0 0 0 0 0
0 1 1 1 1 1
1 0 2 2 2 2
3 2 0 4 4 4
7 6 4 0 8 8
15 14 12 8 0 16
31 30 28 24 16 0
0 61 59 55 47 31
61 120 116 108 92 == 497

120
116
108
092
061
---
497

Thus the sequence 120,116,108,92,61,0 represents a set of generalized Fibonacci numbers.
```