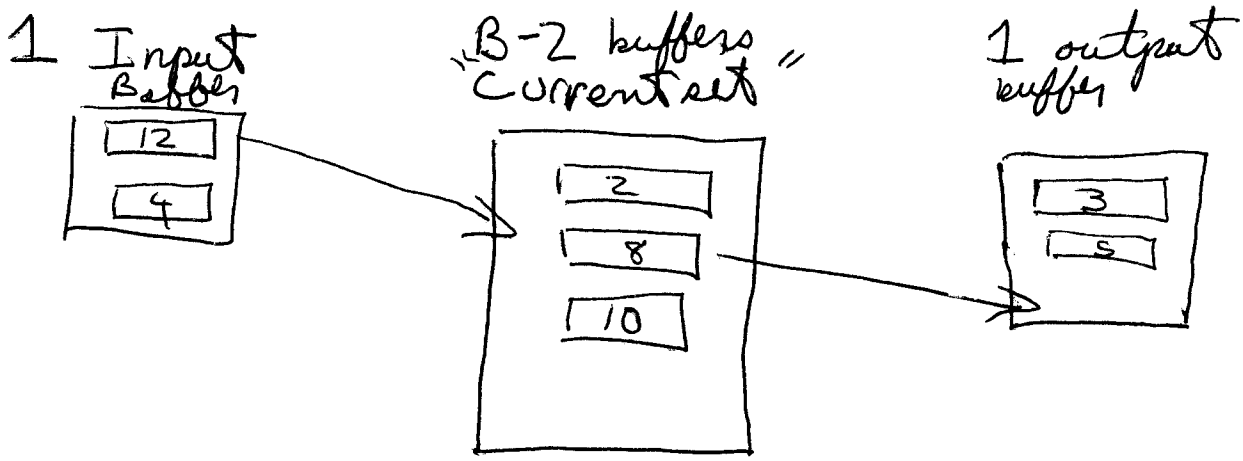
Tricks to improve performance of external sorting

Replacement Sort

Rather than ~~B~~ bfr input ~~B~~ buffers & 1 output buffer have:



- Idea: want to make run size increase by 2x bfr in length each time
- To decide w/c tuple to put in output buffer pick least tuple ~~B~~ in current set
- In above this is 8. Moving 8 to output buffer frees a slot in current set. Fill this slot from input buffer (12 set copied)
- If input buffer is empty we read in new block from one of our source files (cycle through)
- If output buffer fills write it out but remember largest value. Keep going on current run.
- If no tuple in current set larger than output buffer then run ends and start a new run.

Can show on average set twice as long runs.

Not used currently but recent work to minimize main memory usage might make used in future

I/O cost versus I/Os

Total numbers of I/Os is not a perfect metric because it ignores fact some I/O cheaper than others. Ex) blocked I/O

Does it make sense to read in b pages of data at a time. i.e make a buffer block b pages? Hope blocked I/O will improve performance

Let B = total # of pages in buffer pool

then get $\left\lfloor \frac{B-b}{b} \right\rfloor$ ← output Buffer block

runs can be merged in one pass

Ex) Suppose B = 10. Then originally

$\left\lfloor \frac{10-1}{1} \right\rfloor = 9$ runs could be merged in one pass

If b = 4 then get $\left\lfloor \frac{B-4}{4} \right\rfloor = 4$

Still current main memory sizes large enough that all but ridiculously large files can be sorted in around two passes

So even though number of merge runs not as large blocking can be faster