Two-Way External Merge Sort, General Merge Sort, and Why?

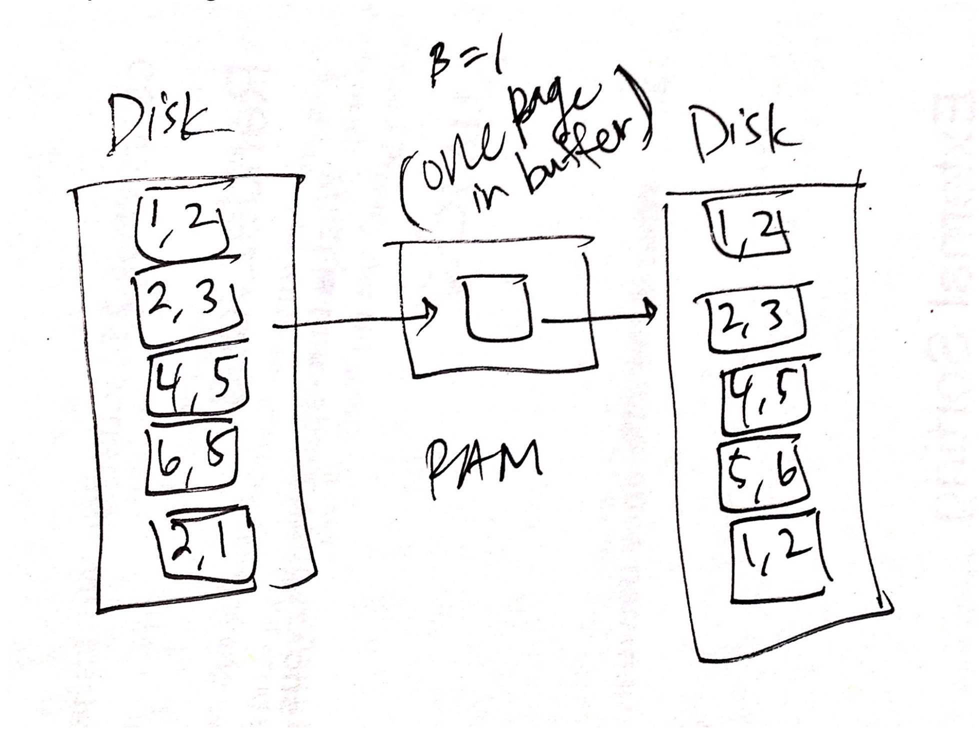
Here are the problem specs: 1. We want to sort large amounts of data from disk without having to always read and write directly to disk! (too expensive!), and 2. We have RAM (random access memory; a place where we can do computations like sorting and hashing with A LOT less cost than directly reading and writing to disk memory) to do our computations in and we also have allocated pages for the buffer in RAM ("buffer pages"). For example, we can say that our RAM has a buffer of 4 pages (or RAM has 4 "buffer pages"). Pages will be the units of memory we will be using in this class so don't worry about anything about blocks and stuff like that! Just remember to use pages as the unit of memory!

Let's use RAM to our advantage!!

TWO-WAY EXTERNAL MERGE SORT:

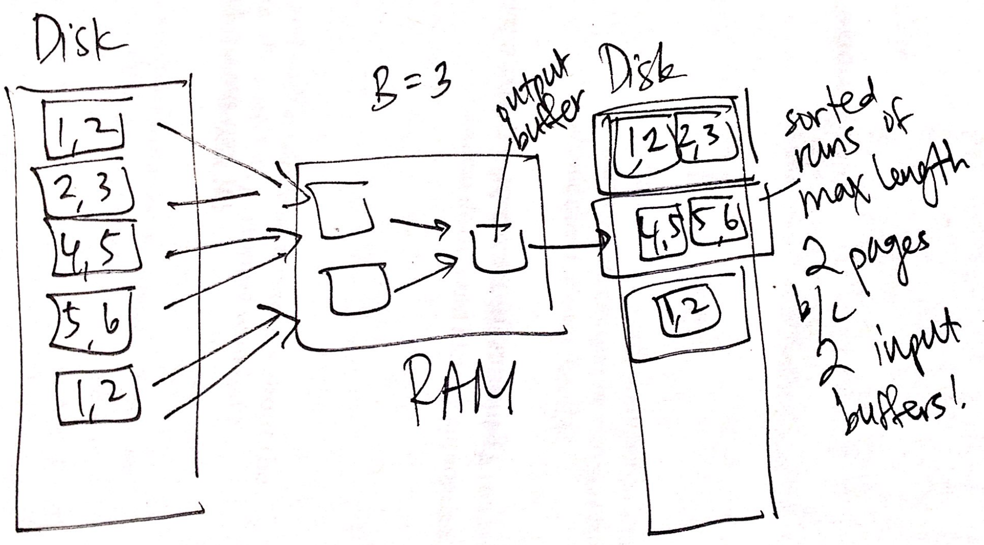
Problem: We have N=5pgs. of data we want to sort! (Keep in mind that two way is very specific so I'm not specifying how much memory we have in RAM).

Pass 0: Let's sort N pages of data using one "buffer page" in RAM. Here's an example image of what that looks like:



Notice how after pass 0, we still have N sorted runs of length 1 page.

Pass 1, 2, ... N: Let's sort what we got from pass 0, but now with 3 buffer pages (one for output and two for input). The output buffer is used to **stream** (meaning we don't keep the pages around in RAM forever, just read/write to disk, and RAM forgets about it) and write the pages to disk.



As you can see, we merge each run by a factor of 2, meaning that if we have runs of 2 pages, the next pass we do will result in runs of 4 pages, then the next pass will result in runs of 8, etc.

This is why we calculate the number of passes as: 1 + ceil($\log_2$(N)) where 1 is pass 0, and the number of subsequent passes are determined by how many passes it will take to merge N runs by a factor of 2 into one N page length run. For example, if we had 40 pages of data and 2 input buffers, pass 0: 40, 1 pg. length runs → pass 1: 20, 2 pg. length runs → pass 2: 10, 4 pg. length runs → pass 3: 5, 8 pg. length runs → pass 4: 2, 16 pg. length runs and 1, 8 pg. length run → pass 5: 1, 32 pg. length run and 1, 8 pg. length run → pass 6: 1, 40 pg. sorted run!! See how each max run length is increased by a factor of 2? It took 7 passes in two way external merge sort to sort a 40 page file. Let's see if we get the same number of passes when we calculate with the equation = 1 + ceil($\log_2$(40)) = 7. YAY!!

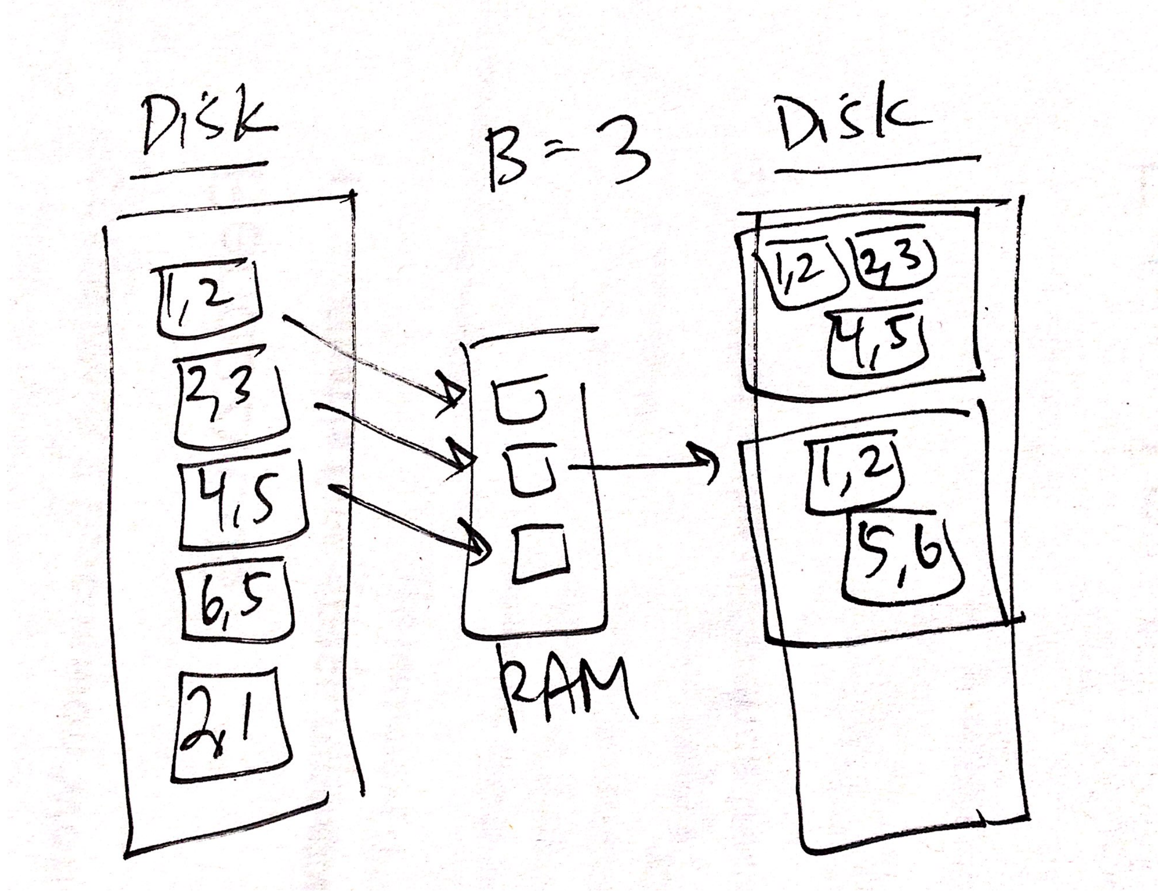
Thus, number of I/Os this will do is: 2N * (# of passes) since we read and write N pages per pass

Now that we're familiar with two-way external merge sort, let's see how we can make it better with utilizing ALL buffer pages we can get in RAM!

GENERAL EXTERNAL MERGE SORT:

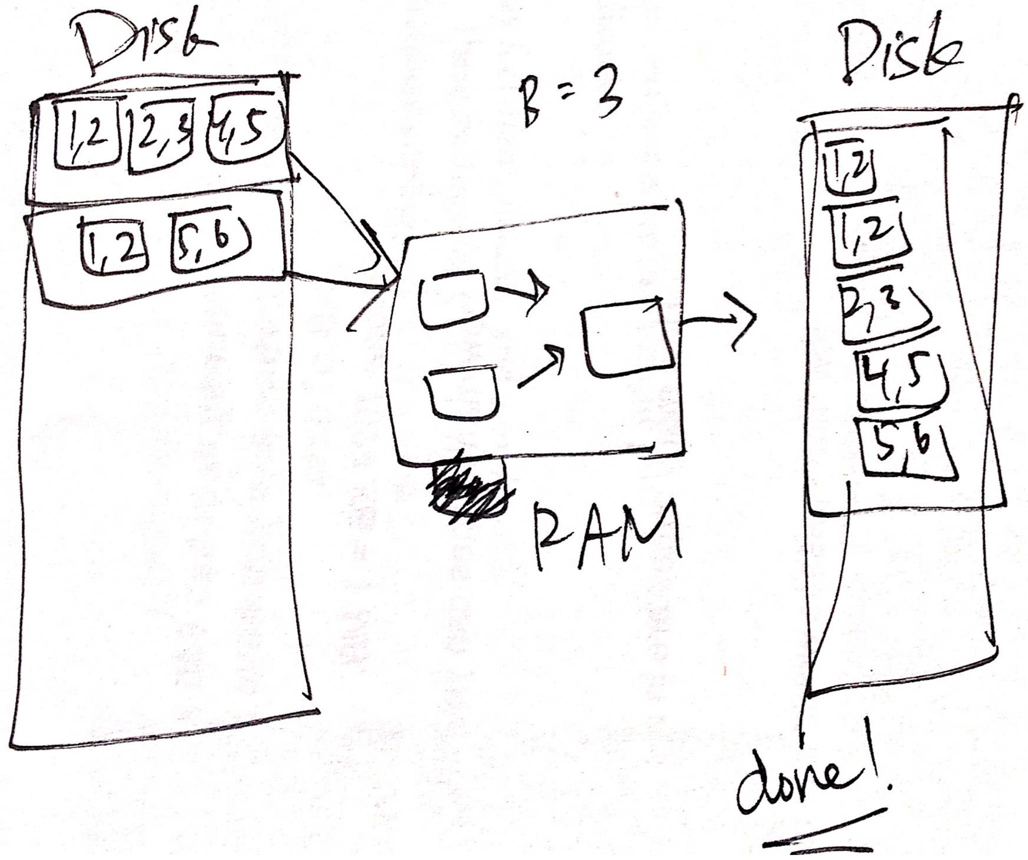
Problem: Let's take the same problem from 2-way external merge sort, we have N=5pgs. of data we want to sort but now we have B=3 pages in RAM.

Pass 0: Let's sort N pages of data using these 3 "buffer pages" in RAM. Here's an example image of what that looks like.



Notice how after pass 0, we DON'T have N sorted runs of length 1 page anymore, we have 2 sorted runs of max 3 pages because we used 3 buffers to create these sorted runs. This is what's different between general and two-way merge sort. We use more buffers in pass 0 for general merge sort to create sorted runs of length B! In two-way, we created initial sorted runs of length 1 pg. (which is not good!).

Pass 1, 2, ... N: Let's sort what we got from pass 0 with 3 buffer pages (one for output and two for input). If B=10, we would use 10 pages of buffer for pass 0 and 9 input, 1 output for pass 1, 2...N, etc. etc.



As you can see, we merge each run by a factor of B-1, meaning that if we have runs of B-1 pages, the next pass we do will result in runs of $(B-1)^2$ pages, then the next pass will result in runs of $(B-1)^3$, etc. etc.

This is why we calculate the number of passes as: `1 + ceil(log`$_{B-1}$`(N))` where 1 is pass 0, and the number of subsequent passes are determined by how many passes it will take to merge N runs by a factor of B-1 into one N page length run. For example, if we had 40 pages of data and 5 input buffers, one for output aka B=6, `pass 0:` 6, 6 pg. length runs and 1 4 pg. length run → `pass 1:` 1, 30 pg. length run and 1, 10 pg. length run → `pass 2:` 1, 40 pg. length sorted run! See how each max run length is increased by a factor of B-1? It took WAYYYY less passes than in two-way external merge sort because we initially created ~N/B runs of B pg. length and then merged them by a factor of B-1 in each subsequent pass. Let's see if we get the same number of passes when we calculate with the equation = $1 + \text{ceil}(\log_{B-1}(40)) = 1 + \text{ceil}(\log_5(40)) = 3$. YAAAYYY!!

Thus, number of I/Os this will do is: `2N * (# of passes)` since we read and write N pages per pass

What is the MAX amount of data that we can sort in 2 passes because that's the minimum amount of passes external merge sort can do! Let's find out.

Let's say we have B=5 pg. in our buffer. What's the max amount of data that we can sort in 2 passes?

Well, let's observe.

1. Pass 0 makes ~N/B runs of length B! This is because we use all buffers to create sorted runs. So for example, if we had 40 pages, our buffer would make 8 sorted runs of length 5 pg.

We need the next pass to merge ALL of the runs we made from pass 0 in order to make this a 2-pass merge sort. What needs to happen? Let’s do some algebra.

Since we created about N/B sorted runs of length B from pass 0 and we only have B-1 buffers to merge these runs, N/B must equal B-1!

N/B = B-1 → multiply the B to both sides
N = B(B-1)

The max N we can sort in 2 passes is B(B-1) pages because pass 0 splits the data into ~N/B sorted runs and we need to make sure that that number of runs is less than or equal to B-1 (the amount of buffers we have to merge!).

Answering the question, how what’s the max amount of data we can sort when B=5 pg.?

N = 5(5-1) → N = 5(4) → 20 pages of data.