1.) This happens due to the fact that the smaller buffer sizes must call the read\_write commands more times than a larger buffer due to the fact that a smaller buffer can hold much less data than a larger buffer. A larger buffer will be simillar in time variations to the memmap because as stated before the larger the buffer the less times read\_write is called. Since, the memmap is efficient then a more efficient buffer of larger size is going to be more comparable to the memmap in reguards to time.

The read\_write commands are called once on the read aspect and then again to write. This results in a longer time of completion and proves for a fairly inefficient process. The memmap maps the read and the write directly in one motion creating a much ore proficcient way of using I/O.

2.) The mistake is that the read\_write will make calls directly until there is nothing left. Meaning that if the number of bytes is not divisible by the amount that is being copied then there will be a remainder left over, which is where this differentiation in size comes from. This can be corrected by making sure that once the entire file has been read that the read\_write function is not continually called.