In this lab, we collected data for 3 types of I/O patterns (sequential, stride, and random) for both HDD and SSD to see the difference in their performance characteristics. Below are the results we found from our data.

For sequential writes on HDD, we see that the throughput increases roughly logarithmically until it peaks at around 32768 byte block size. The same cannot be said for sequential reads, which have roughly the same throughput on sequential reads with a block size larger than 32 bytes. The contrast between the sequential writes and reads is interesting because it shows that smaller block sizes affect the throughput of writes more than reads on HDD. This suggests that the HDD has to do some bookkeeping on every single write transaction regardless of stride or location. For both SSD sequential reads and writes, the throughput of the I/O operation was higher for larger block sizes. This makes sense because writing to one large contiguous logical block address is faster than writing to multiple smaller LBA ranges.

For random read speed, we noticed a clear downwards trend in throughput as the total write space increased, in both SSD and HDD: This makes sense, as in HDDs, the seek time would be higher, and in SSD's, you'd likely have to traverse through multiple indirect blocks to reach the desired data. However, in random write speed for HDD and SSDs, we didn't see this downward trend, likely because of how HDD writes have this bookkeeping mentioned above, or in the SDD writes how it usually combines writes in the FTL order to increase efficiency and maintain throughput.

Looking at the sequential writes with varying strides on HDD, we can see that larger blocks have a higher throughput, which is expected. Interestingly though, across all block sizes the throughput seems to peak at 1024 KB stride length. This trait is not shared with the sequential reads with varying strides on HDD, where we see that throughput has a strong

negative relationship with stride length across all block sizes. This likely has to do with the cache size of the machine which is probably 1024 KB. When we write anything bigger than 1024 KB, the cache must immediately be backed up in order to store new changes, which is why we see the increased throughput onto disk. For sequential writes with varying strides on SSD, there is a rough uniformity in the 4KB, 32KB, 128KB blocks and a bit of variance in the 256KB and 512KB blocks. This follows with what we learned in class, since the distance between the LBAs does not affect the seek time in SSDs, the throughput should remain the same regardless of the size of the stride. Again we see that the larger blocks correspond with higher throughput. For the sequential reads with varying strides on SSD, the story is a little different. We see a downward trend on throughput as the stride length increases, meaning that fragmentation does actually affect the speed of the read operations. Why this result is more pronounced in the read operations than the write operations is unclear, but we can still draw away that even on SSD, a large distance between LBAs will cause poorer performance, perhaps because we have to traverse through many indirect blocks which requires more seeks, or possibly because when we access close memory addresses they might've gotten cached, whereas accessing distant chunks guarantees no cache hit.

From this experiment, we learned that there is no real universally optimal I/O size for HDD or SSD memory devices. However, in general, larger I/O sizes are much more performant than smaller I/O sizes. Additionally, even in the same storage and I/O size/pattern, we also learned that oftentimes read and write operations will exhibit different behavior depending on the file system or the size of cache. Some lessons to take away from this for designing a file system is that for both HDD and SSD, locality matters. Designing file systems that store related data into contiguous or near contiguous logical block addresses can improve performance on both SSD

and HDD. Thus, using larger block sizes for I/O operations, say around 4MB would naturally raise the throughput on the file system. If there are many small I/O operations, we can delay the operations and try to aggregate the smaller requests into a single large request.