QUESTION 1: Big-Oh Run-Time Calculations

Block transfer

Best case: File content is contiguous (contiguous as in compacted into blocks. If a block spreads over all platters, than so does the file) on disk, there is absolutely no file fragmentation (other than physical fragmentation from block to block, depending on the physical layout of a block). The best case time required to load that file on disk is (time to transfer a block)*(length of file/length of a block).

Big Oh: If the files are typically smaller than a block, the time requirements are O(1), since we only need 1 block per file. Otherwise, we need more specs on how big a typical file and how big a block in order to give a magnitude to O(fileSyze/BlockSize)

Worst case: Worst thing that can happen to a block transfer system is mad file fragmentation. Worst case would be if every byte of a file is separated from the next one, in such a way that there is only a single byte per block. Also, that the next byte is in the block as far away as can be from the previous one with respect to scan path.

Big Oh: The time required to load that file on disk would be...huge! The system would need to wait for worst case (seek+latency time)+(block transfer time) for every byte. (not to mention that the HDD would contain more pointers than data!). Estimating worst case seek time of 5ms latency at 5200RPM + 15-20ms seek time gives total wait of 20-25ms/byte (neglecting the actual transfer time), so to load 1 MB file would take 6.9 hours!! Time would be (total overhead)*(bytes) so in O(n) notation would be some constance k in the range of the thousands O(kn)

Semi contiguous transfer

Best case: Same as above, best case of the files are not fragmented and that the drive can access the next byte just besides the previous one.

Big Oh: Since we are doing a byte per byte transfer, this method takes O(n) time (n being the number of bytes)

Worst case: Same as above, the worst case is if the disk is fragmented in such a way that the next byte is a far as possible from the previous one with respect to seek and latency time.

Big Oh: Just like above, the time requirements would be O(jn) where j is a constance in the thousands, but slightly smaller than k presented above, since the time to copy a byte is much smaller than the time to copy a block (if one block is 1000 byte, and a byte is loaded in 2nS, this difference is 2 mS, but this number is diluted by adding the 20-25mS of disk seek and latency times, so it's effect is minor)

Which is better, in what case and why?

In the case of a highly fragmented disk, it is A LITTLE FASTER to use byte access because we are not waisting time loading up thousands of unnecessary bytes, we can just load the one byte needed then have the boom immediately start to travel towards the next spot on disk.

In the case of contiguous bytes within files, it is MUCH faster to use block transfer since we only need to go through the overhead of getting the reading heads to the block area once per block, which is extremely time consuming.

Overall, since bytes on a file are (should be) generally more contiguous than fragmented (since when a user purchases a disk, it is empty, so the until he starts to delete content, there is no fragmentation), it is better to implement a block transfer.

Comparing the findings above, on a contiguous disk, we are looking at a huge performance advantage of block transfer over byte transfer, whereas in the worst case (fragmented disk), the byte transfer only brings a slight time advantage. So we have a choice between potentially saving a lot of time with block transfers, or potentially saving a little time with byte transfer.

Since it is relatively easy to defrag a disk, we have easy access to transform a slight disadvantages into a significant advantage by implementing block transfers.