# Exercise 1: Hard Disks

512 bytes/sector, 512 sectors/track, 4096 tracks/plater surface, 2 surfaces/plater, 5 platers/disk. Average seek: 10 ms, 7200 RPM rotation frequency.

a. How many bytes can be stored on the hard disk? $512\frac{byte}{sector} \cdot 512\frac{sectors}{track} \cdot 4096\frac{tracks}{plater} \cdot 2\frac{sides}{plater} \cdot 5\frac{platers}{disk} = 10737418240\frac{byte}{disk} = 10\frac{Gibibyte}{disk}$

b. total numbers of cylinders? 4096, as it's the number of sets with the same track diameter and there are 4096 tracks, one track beeing a circle of diameter $d_i, i \in [0, 4095]$

c. Maximum and average rotation delay? One full revolution takes: $7200^{-1}\frac{min}{revolutions} = \frac{60000ms}{7200revolutions} = 8.\overline{3}\frac{ms}{revolution}$.
Thus the maximum rotation delay is $8.\overline{3}$ ms and the average rotation delay is $\frac{8.\overline{3}ms}{2} = 4.\overline{16}ms$

# Exercise 2: Parities

a. parity sequence on 4th disk? Disk 4 (Parity): 10011001

b. How to restore disk 2?

```
int cum_set_bits;
for( disk : valid_data_disks)
    cum_set_bits += disk.bit[i]

if ((cum_set_bits[i] % 2) == parity_bit[i])
    restoring_disk.bit[i] = 0
else
    restoring_disk.bit[i] = 1
```

# Exercise 3: Creating and Reading files with Java

a. JMHBenchmark: Java benchmarks via annotations
try (ressource) {} => closes resource automatically; Channels: seq shitty if, then SeekableByteChannel, RandomAccessFile best. Files.readAllBytes am schnellsten, else similar

b. The read would have to be recursive and returning a map, list or similar as an array based read'll fail if the content to read is larger than 2GB as Java limits the array size to be of type integer (and positive).
Thus one could instead of recursing set a goto at the beginning of the method to avoid the duplication of the register space used to keep track of function calls like in the clang Lexer but for that one has to change the interface.
Both would involve a change in the signature/interface of the method and is ommited due to that fact.
Same thing for opening a channel only once to reduce the number of function calls which are expensive in Java:
That is only possible within a dynamic context, meaining not in a static method.
Besides that a FileChannel (subclass of ByteChannel) is used, that can be mmaped which is much faster than the usual read and write for large files (here it's not used due to the static context (need to safe the Pointer to the mmaped area resulting in opening, loading and closing it for each rw)):
javadoc v10.
Further the FileChannelImpl class is implemented with thread-safety in mind (cf. l. 84) and

based on raw byte-wise streaming (cf. l. 124 and FileOutputStream), thus not limited to characters.

To summarize:

- Fastest for static methods and only a single read is a FileChannel over a SeekableByteChannel as it creates/uses a RandomAccessFile which is fastest while creation. Using a SSD and only a short amount of time to test the read time is also the best, which imediately changes when one reads over a longer amount of time Seagate report on SSD benchmarks
- Given static methods and a HDD or longer test times the best option is to use FileInputStream and FileOutputStream as the two of them create/use sequential files.
- The actual fastest option should be the a FileChannel over FileInputStream & FileOutputStream using a MappedByteBuffer but only if the methods are non-static, thus allowing the mapped memory region to remain in scope, so that it doesn't have to be malloced, loaded and garbage collected on every read.