1. (Exercise 3.2.5 on page 96 of your textbook): Suppose a record has a record header consisting of two 4-byte pointers and a character, plus the following fields in this order: A character string of length 15, an integer of 2 bytes, an SQL2 date, and an SQL2 time (no decimal point). Suppose we wish to pack as many records as we can into a block of 4096 bytes, using a block header that consists of ten 4-byte integers. How many records can we fit into the block in each of the following three situations regarding field alignments:

a. Fields can start at any byte.

Under this arrangement a field has the following format (lengths are in parentheses)

<table>
<thead>
<tr>
<th>header</th>
<th>character string</th>
<th>int</th>
<th>SQL2 date</th>
<th>SQL2 time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4)</td>
<td>(4)</td>
<td>(1)</td>
<td>(15)</td>
<td>(2)</td>
</tr>
</tbody>
</table>

So each record spans 44 bytes. If a block header comprises ten 4-byte fields, then the first 40 bytes of each block is taken up with this header, leaving 4056 bytes for record storage, which means that we can store

\[
\left\lfloor \frac{4056}{44} \right\rfloor = \left\lfloor 92.18... \right\rfloor = 92 \text{ records in each block}
\]

b. Fields must start at a byte that is a multiple of 4.

Here a field has the following format, where a block with a hashed indicates a number of bytes of padding necessary to align the fields on the proper boundary. We assume the individual components of the header also need such alignment. Below each field is the staring address of each byte

<table>
<thead>
<tr>
<th>header</th>
<th>character string</th>
<th>integer</th>
<th>SQL2 date</th>
<th>SQL2 time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4)</td>
<td>(4)</td>
<td>(1)</td>
<td>(15)</td>
<td>(2)</td>
</tr>
</tbody>
</table>

So each record now spans 52 bytes. If a block header comprises ten 4-byte fields, then the first 40 bytes of each block is taken up with this header (each field of which is already properly aligned), leaving 4056 bytes for record storage, which means that we can store

\[
\left\lfloor \frac{4056}{52} \right\rfloor = \left\lfloor 78 \right\rfloor = 78 \text{ records in each block}
\]

c. Fields must start at a byte that is a multiple of 8.

Here a field has the following format, where a block with a hashed indicates a number of bytes of padding necessary to align the fields on the proper boundary. We assume the individual components of the header also need such alignment. Below each field is the staring address of each byte

<table>
<thead>
<tr>
<th>header</th>
<th>character string</th>
<th>integer</th>
<th>SQL2 date</th>
<th>SQL2 time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4)</td>
<td>(4)</td>
<td>(4)</td>
<td>(1)</td>
<td>(15)</td>
</tr>
</tbody>
</table>

So each record now spans 72 bytes. If a block header comprises ten 4-byte fields, then these must also be properly aligned, which means they must now be ten 8-byte fields, so now the first 80 bytes of each block is taken up with this header, leaving 4016 bytes for record storage. This means that we can store

\[
\left\lfloor \frac{4016}{72} \right\rfloor = \left\lfloor 55.777... \right\rfloor = 55 \text{ records in each block}
\]
2. (Exercise 3.3.3 on page 106) Assuming we want addresses for a single Megatron 747, a separate byte or bytes to each of the cylinder, track within a cylinder, and block within a track, and that we want to represent record addresses as well.

a. How many bytes do we need for such a physical address if we included the number of the byte within a block as part of the physical address?

Since we have 8192 cylinders, we need at least $2^{14}$ different cylinder addresses. Since two bytes will yield $2^{16}$ addresses, we allocate two bytes for cylinder addresses.

We have 8 ($=2^3$) different surfaces/tracks within a cylinder, so 1 byte suffices for this.

Although the number of blocks can vary from 747 to 747, it is reasonable to assume that we will have at least 2 sectors per track. Since the 747 has at most 320 sectors per track, we can then assume we will have at most 160 blocks per track, which means that 1 byte (which can handle $2^8 = 256$ block addresses per track) will suffice.

Since record addresses require offsets into a 4096 = $2^{12}$ byte block, then allocating 2 bytes (216 offsets) will suffice.

Thus we see that physical addresses of records on the 747 can be represented using $2 + 1 + 1 + 2 = 6$ bytes.

b. How many bytes do we need for such a physical address if we structured addresses for the records?

Assume the stored records have a 4-byte integer as a key.

In part a, we saw that block addresses will use 4 bytes. Incorporating a 4-byte integer into the format will add another 4 bytes, so altogether the physical address will use 8 bytes.

3. (Exercise 3.3.5 on page 106) Today IP addresses have 4 bytes, but in the future IP addresses will use 16 bytes and devices will have their own IP addresses. How many bytes will we need to represent physical record addresses in these circumstances assuming devices were Megatron 747 devices.

In the above problem we saw that 6 bytes were needed for record addressing on a Megatron 747. If we include the 16 bytes for its IP address, then we will need 22 bytes for record addresses.

4. (Exercise 3.3.6 on page 106) Suppose we wish to represent the addresses of blocks on a 747 logically, that is, using identifiers of k bytes for some k. We also need to store on the disk itself a map table consisting of pairs of logical and physical addresses. Assuming that both physical addresses (as in problem 2) and logical addresses use the minimum possible number of bytes, how many blocks of 4096 bytes does the map table for the disk occupy?

Suppose we assign each block an integer as a logical identifier. With 4K blocks (= 8 sectors) we see that the 747 has the following number of blocks:

- $8 \times 2730 \times (192+8) = 514160$ in region 1,
- $8 \times 2731 \times (256+8) = 669136$ in region 2, and
- $8 \times 2731 \times (320+8) = 873920$ in region 3, or

for a total of $524160 + 699136 + 873920 = 2097216$ block $\approx 2^{21}$ blocks.

Since 2 bytes would clearly not be enough to represent a reasonable number of blocks (only 65536), we shall use 3 bytes for logical addresses. We note that this would suffice even if we would decide to use just 1K blocks. It is not enough for 1-sector blocks, however, but it is more common to use multi-sector blocks, so we assume this.

We saw in problem 2 that it is reasonable to work with 4-byte block addresses, so that a logical-address + physical-address pair in a map table will use 7 bytes. Now, we shall need to represent (at most) 2097216 logical addresses in the map table. At 7 bytes each, we can store $\left\lfloor \frac{4096}{7} \right\rfloor = \left\lfloor 585.142.. \right\rfloor = 585$ logical-physical address pairs per block, which means we shall require $\left\lceil \frac{2097216}{585} \right\rceil = \left\lceil 3584.98.. \right\rceil = 3585$ blocks for the map table.
5. (Exercise 3.3.7 on page 107) Suppose we have 4096-byte blocks in which we store records of 100 bytes. The block header consists of an offset table using 2-byte pointers to records within the block. On an average day, two records per block are inserted, and one record is deleted. A deleted record must have its pointer replaced by a tombstone because there may be dangling pointers to it. Assume the deletion on any day occurs before the insertions. If the block is initially empty, after how many days will there be no room to insert any more records.

At the end of the first day the size of the block decreases by 204 bytes (2 records plus two, 2-byte entries in the block header), leaving us with 3892 bytes. Based on our assumption there was no deletion the first day. Now on each subsequent day, if we assume that the space for a deleted record can be re-used (but not its storage in the block header since that stores the tombstone) then each day the amount of available storage in the block decreases by $100 + 2\times 2 = 104$ bytes. If $D$ represents the last day that the insertions can be made into the block, then we must have

$$3892 - 104\times (D-1) \geq 0,$$

but

$$3892 - 104\times D < 0 \text{ or } 3892 < 104\times D \leq 3996 \Rightarrow 37.42 < D \leq 38.42$$

whence after day 38 there will be no more room to insert any more records.

6. (Exercise 3.4.1 on page 115) A patient record consists of the following fixed length fields: the patient’s date of birth, Social Security number, and patient ID, each 10 bytes long. It also has the following variable-length fields: name, address, and patient history. If pointers within a record require 4 bytes, and the record length is a 4-byte integer, how many bytes, exclusive of the space needed for the variable-length fields, are needed for the record? You may assume that no alignment of fields is required.

Here we are assuming that the records will be stored with the fixed length fields following the block header. There are two options:

- If we store the variable length fields after the fixed length data as in Figure 3.12 on page 109, then we need not store a pointer to the name fields, whence our we will require $10 + 10 + 10 = 30$ bytes for the fixed length fields, 4 bytes for the record length, 4 bytes for the pointer to the address field, and 4 bytes for the pointer to the patient history field, for a total of $30 + 4 + 4 + 4 = 42$ bytes.

- If we store the variable length fields separate from the record as in Figure 3.14 on page 111 then we will need 30 bytes for the fixed length field 4 bytes for the record length, 4 bytes for the pointer to the name field and 4 bytes for its length, 4 bytes for the pointer to the address field and 4 bytes for its length, and 4 bytes for the pointer to the patient history field and 4 bytes for its length. Thus gives us a total of $30 + 4 + (4 + 4) + (4 + 4) + (4 + 4) = 58$ bytes.

7. (Exercise 3.4.2 on page 115) Suppose records are as in problem 6 above and the variable length fields name, address, and patient history each have a length that is uniformly distributed. For the name the range is 10-50 bytes, for address it is 20-80 bytes, and for patient history it is 0 – 1000 bytes. What is the average length of a patient record?

If the lengths are uniformly distributed then each length on the given range is equally as likely to occur, so the average length for each field is the value in the midrange. Consequently in the average record the length of the name field is 30 bytes, the length of the address field is 50 bytes, and the length of the patient history field 500 bytes. Thus, the average length of the variable portion of a record is $30 + 50 + 500 = 580$ bytes. When we include the fixed length data (42 or 58 bytes depending on the format chosen) then the average length of a record is either 622 bytes or 638 bytes.