Implementation of DBMS
Exercise Sheet 4

1. Suppose that we have 8,192-byte blocks in which we store records of 200 bytes. The block header consists of an offset table using 4-byte pointers to records within the block. Each day one record is deleted (if records are in the block) and afterwards three records are inserted. A deleted record must have its pointer on the offset table replaced by a tombstone. If the block is initially empty, for how many days can we insert records into a block? (ans. 19 days)

2. We have a data file with 10^6 records. Records and blocks are like in Task 3a) of Sheet 3. How many blocks do we need for the data file?
   a) We use spanned storage.
   b) We use unspanned storage.

Given problem (restated)

> We have a data file with $10^4 = 10{,}000$ records. Each record is **48 bytes**. Each disk block is **4096 bytes** but each block has a **40-byte header**, so only $4096 - 40 = 4056$ bytes per block are available for storing record bytes.
>
> (a) If we use **spanned** storage (records may cross block boundaries), how many blocks are needed?
> (b) If we use **unspanned** storage (each record must be wholly contained in a single block), how many blocks are needed?

Key concepts before the math

- **Record size** = 48 bytes (given).
- **Number of records** = 10,000.
- **Raw total data bytes** = `record_size × number_of_records = 48 × 10,000 = 480,000 bytes`. This is the total payload we must store.
- **Block size** = 4096 bytes, but each block has a 40-byte header that cannot be used for storing record bytes.
- **Usable bytes per block for records** = `4096 − 40 = 4056 bytes`.

Spanned vs Unspanned:

- **Spanned:** records may continue from the end of one block into the start of the next. That means we can use *all usable bytes in every block* to store record bytes; effectively we treat the storage as one long byte stream. So number of blocks = `ceil(total_payload_bytes / usable_bytes_per_block)`.
- **Unspanned:** each record must be placed wholly inside a single block. You must compute how many whole records fit in one block (`floor(usable_bytes_per_block / record_size)`), call this `rpb` (records per block). Then number of blocks = `ceil(number_of_records / rpb)`.

(a) Spanned storage — detailed steps

1. Total bytes to store = $48 \times 10{,}000 = 480{,}000$ bytes.
2. Usable bytes per block = $4096 - 40 = 4056$ bytes.
3. Since records may be split across blocks, we can fully utilize the 4056 bytes per block. Compute number of blocks:

$$\text{blocks} = \left\lceil \frac{480{,}000}{4056} \right\rceil .$$

4. Compute the division:

$$\frac{480{,}000}{4056} \approx 118.36\ldots$$

5. Take the ceiling (you cannot have a partial physical block):

$$\text{blocks} = \lceil 118.36\ldots \rceil = 119.$$

Answer (a): 119 blocks.

(b) Unspanned storage — detailed steps

1. Usable bytes per block (same as above) = 4056 bytes.
2. Compute how many whole records fit into one block:

$$\text{records\_per\_block} = \left\lfloor \frac{4056}{48} \right\rfloor .$$

Calculate: $4056 \div 48 = 84.5$. The floor of that is 84.

So each block can store **84 full records** (the remaining 0.5 record worth of bytes is unusable under unspanned rules).

3. Now compute number of blocks needed:

$$\text{blocks} = \left\lceil \frac{10{,}000}{84} \right\rceil .$$

Calculate: $10{,}000 \div 84 \approx 119.0476\ldots$. Ceiling gives 120.

Answer (b): 120 blocks.

Intuition for the small difference (119 vs 120)

- With **spanned** storage you squeeze the raw byte stream into blocks: 480,000 bytes / 4056 bytes per block → 118.something blocks → 119 blocks.
- With **unspanned** storage you waste the fractional remainder in each block (because a record cannot be split). Each block holds only 84 records = 84×48 = 4,032 bytes used; unused per block = 4056−4032 = 24 bytes wasted. That small per-block waste accumulates across many blocks and requires one extra block overall (120 instead of 119).

Two similar practice questions (same style)

Q2a)

A file contains $5{,}000$ records. Each record is **92 bytes**. Disk blocks are **4096 bytes** and each block has a **32-byte header** (so usable bytes per block = 4096 − 32).

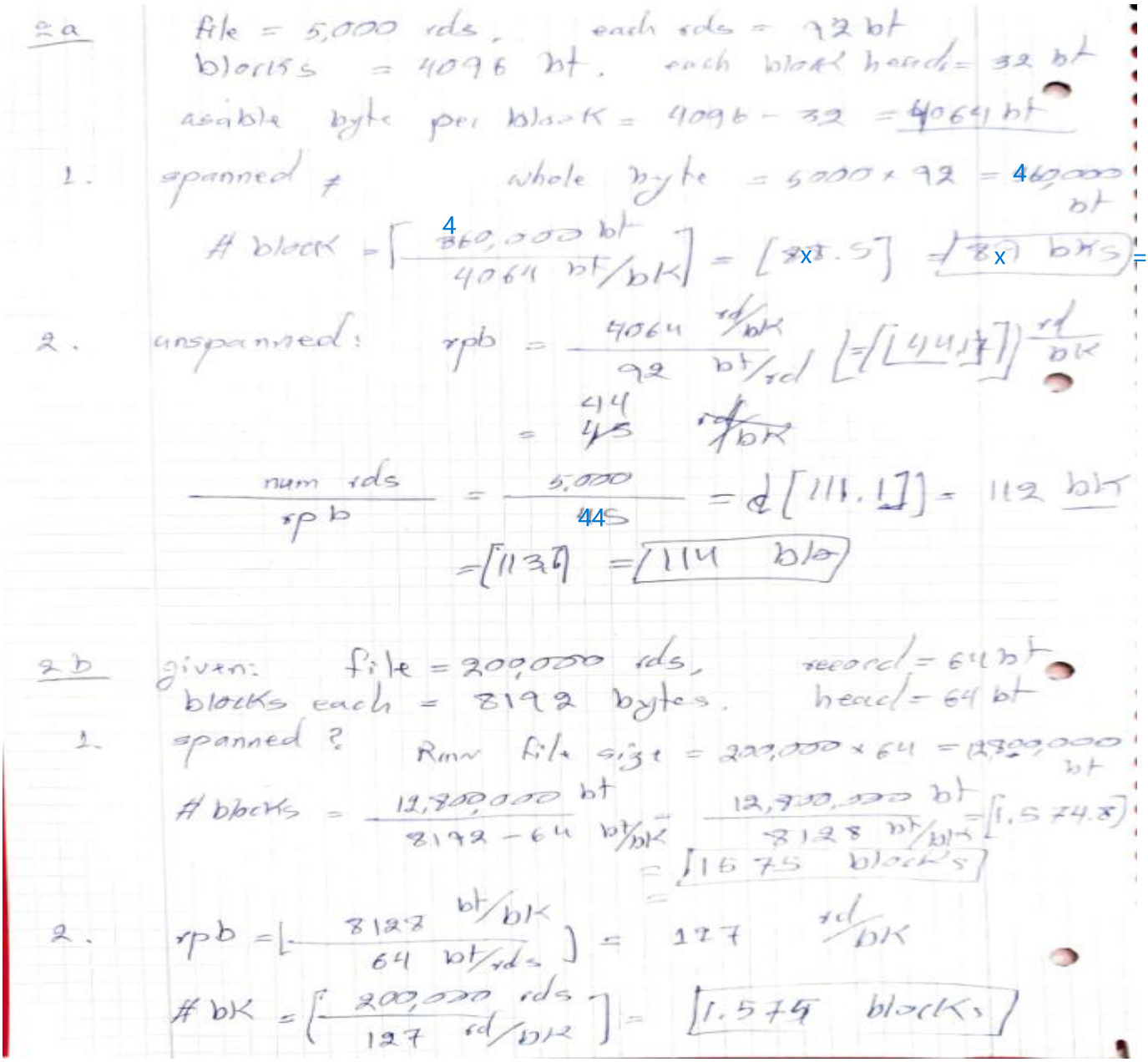
1. How many blocks are needed with **spanned** storage? 51 blocks
2. How many blocks are needed with **unspanned** storage? 52 blocks

---

Q2b)

A file contains $200{,}000$ records. Each record is **64 bytes**. Disk blocks are **8192 bytes** and each block has a **64-byte header**.

1. How many blocks are needed with **spanned** storage?
2. How many blocks are needed with **unspanned** storage?



2a)
We have a data file with **245 records**. Each record is **180 bytes**, and each block can store **1,000 bytes**.
How many blocks do we need for the data file?

1. Using **spanned storage**.
2. Using **unspanned storage**.

Soln:

1. 114 blk
2. 113 blk

2b)
A customer database contains **92 records**, each of size **120 bytes**. The block size is **512 bytes**.
How many blocks are required for:

1. **Spanned** storage. 22 blocks
2. **Unspanned** storage. 18 blocks

2c)
We have a file with **350 student records**, each **220 bytes**. Block size is **2 KB**.
Calculate:

1. Number of records per block (**spanned** and **unspanned**).
2. Number of blocks needed for both cases.

2d)
You are storing **75 employee records**, each **300 bytes**.
Block size = **1,024 bytes**.
How many blocks do you need if:

1. **Spanned** storage is used.
2. **Unspanned** storage is used.

2e)
A log file has **500 entries**, each **128 bytes**. The block size is **1 KB**.

1. Calculate the number of blocks with **spanned storage**.
2. Calculate the number of blocks with **unspanned storage**.

Rules used

- **Spanned storage:** records may cross block boundaries → number of blocks = `ceil(total_bytes / block_size)` where `total_bytes = #records × record_size`.

- **Unspanned storage:** records must fit wholly inside a block → records_per_block = `floor(block_size / record_size)`, blocks = `ceil(#records / records_per_block)`.

Question 2a:

A database file contains **50000 records**. Each record has the following attributes: **Attribute A**: 32 bytes, **Attribute B**: 16 bytes, **Attribute C**: 24 bytes.

Each **record takes 72 bytes**, and the **block size is 8192 bytes** with a **block header of 128 bytes**.

(a) Calculate the **number of blocks needed** for the data file if **spanned storage** is used.
(b) Calculate the **number of blocks needed** for the data file if **unspanned storage** is used.

---

3. Suppose that we handle insertions into a sequential data file of n records by creating overflow blocks as needed. Also, suppose that the data blocks are currently two-thirds full. If we insert new records at random, how many records do we have to insert before the average number of data blocks (including overflow blocks if necessary) that we need to examine to find a record with a given key reaches 2? Assume that on a lookup, we search the primary block pointed to by the index first, and only search overflow blocks, in order, until we find the record, which is definitely in one of the blocks of the chain.

---

4. Suppose blocks hold either five records or 20 key-pointer pairs. As a function of n, the number of records, at least how many blocks do we need to hold:
   a) The data file.
   b) A dense index.
   c) A sparse index.
   You can ignore inaccuracies that result from rounding.

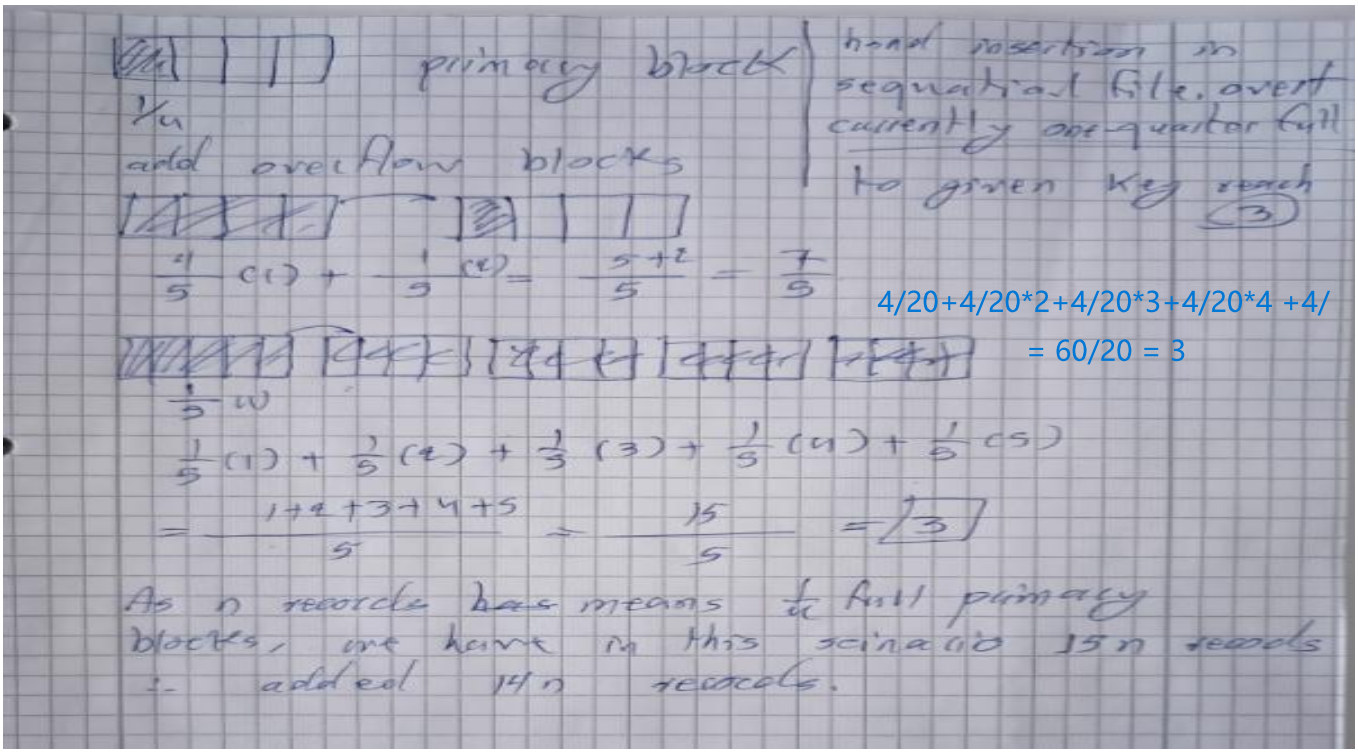
Implementation of DBMS, Exercise Sheet 4

1. Suppose that we have 2,048-byte blocks in which we store records of 150 bytes. The block header consists of an offset table using 3-byte pointers to records within the block. Each day two records are deleted (if records are in the block) and afterwards four records are inserted. A deleted record must have its pointer in the offset table replaced by a tombstone. If the block is initially empty, for how many days can we insert records into a block? (ans. 8 days)

2. A **customer database** consists of **80000 records**, each having the following attributes:
- **Customer ID**: 8 bytes, **Name**: 40 bytes, **Address**: 50 bytes, **Phone Number**: 20 bytes

Each record is **118 bytes** in total. The **block size is 8192 bytes**, with a **block header of 96 bytes**.
(a) Compute the **number of blocks required** if **spanned storage** is used.
(b) Compute the **number of blocks required** if **unspanned storage** is used.

3. Suppose that we handle insertions into a sequential data file of n records by creating overflow blocks as needed. Also, suppose that the data blocks are currently one-quarter full. If we insert new records at random, how many records do we have to insert before the average number of data blocks (including overflow blocks if necessary) that we need to examine to find a record with a given key reaches 3? Assume that on a lookup, we search the primary block pointed to by the index first, and only search overflow blocks, in order, until we find the record, which is definitely in one of the blocks of the chain.



4. Suppose blocks hold either eight records or 25 key-pointer pairs. As a function of n, the number of records, at least how many blocks do we need to hold:
   a) The data file.
   b) A dense index.
   c) A sparse index.
   You can ignore inaccuracies that result from rounding.

2a, 2b, and 2c - Similar Questions:

2a) We have a data file with 5×10^5 records. Each record is 64 bytes, and the block size is 8192 bytes. How many blocks are needed for the data file if:
a) We use spanned storage.
b) We use unspanned storage.

2b) A file contains 2×10^6 records. Each record occupies 32 bytes, and each block has a size of 4096 bytes.
Determine the number of blocks required in:
a) Spanned storage.
b) Unspanned storage.

2c) A database table holds 10^7 records. The size of each record is 128 bytes, and the block size is 16 KB (16384 bytes).
Calculate the number of blocks needed in the following cases:
a) Spanned storage.
b) Unspanned storage.

Question 2b:

A **log file** contains **200000 records**, with each record consisting of: **Timestamp**: 12 bytes, **UserID**: 24 bytes **EventType**: 16 bytes, **Details**: 48 bytes.
The **block size is 4096 bytes**, with a **block header of 64 bytes**.
(a) Determine how many **blocks** are required if **spanned storage** is used.
(b) Determine how many **blocks** are required if **unspanned storage** is used.

2a) Block Storage Calculation

We have a **data file** with **250 records**. Each **block** can store up to **20 records**.

a) If we use **spanned storage** (where records can be split across multiple blocks), how many blocks do we need?
b) If we use **unspanned storage** (where records must be fully stored in a single block), how many blocks do we need?

2b) Block Storage for Variable Record Sizes

A **data file** contains **500 records**, each of size **250 bytes**. The **block size** is **2,000 bytes**.

a) How many **blocks** are required if we use **spanned storage**?
b) How many **blocks** are required if we use **unspanned storage**?

2c) Indexing and Block Access

A **database table** has **1,000 records**, and each block can store **50 records**. The table is indexed using a **single-level index**, where each index entry takes **10 bytes**, and an index block has a size of **400 bytes**.

a) How many **blocks** are needed to store the data file?
b) How many **index entries** fit into a single index block?
c) How many **index blocks** are required for the first level of indexing?

---

spanned storage and **unspanned storage**

Key Concepts

1. Spanned Storage

- In **spanned storage**, a record can be split across multiple blocks if it doesn't fit entirely in one block.
- This allows for more efficient use of space, as no block is left partially empty.
- However, it adds complexity because the system needs to manage pointers or links to connect the parts of a record stored in different blocks.

2. Unspanned Storage

- In **unspanned storage**, a record must fit entirely within a single block. If a record is too large to fit in a block, it cannot be stored.
- This leads to potential wastage of space, as some blocks may have unused space if the records don't perfectly fill them.
- It is simpler to implement because there's no need to manage record fragmentation.

Problem Setup

- **Total records**: 10,000, **Record size**: 48 bytes per record, **Block size**: 4096 bytes (4 KB)
- **Block header size**: 40 bytes per block (used for metadata, such as pointers or block IDs)
- **Available space per block for records**: 4096 bytes − 40 bytes = 4056

Practical Implications

- **Spanned storage** is better for systems where space efficiency is critical, but it adds complexity in managing fragmented records.
- **Unspanned storage** is simpler to implement and manage but may waste space.

Implementation of DBMS

*Exercise Sheet 5 from **sheet 4***

1. Suppose that we have 8192-byte blocks to store records of 250 bytes each. Each block has a header consisting of a 64-byte block ID and a 2-byte pointer table for offsets of records within the block. Each time a record is deleted, its pointer is replaced with a tombstone. If the block is initially empty, how many records can be inserted before the block is completely full?
2. A data file contains 256 records. The records and blocks follow the configuration described in Exercise 1. How many blocks are required for the data file if:
   a) Spanned storage is used.
   b) Unspanned storage is used.
3. Suppose we manage a sequential data file with 512 records, using overflow blocks to handle insertions. Assume the primary blocks are initially 75% full. If new records are inserted randomly, how many records must be inserted before the average number of blocks (including overflow blocks) that need to be examined for a lookup reaches 3? Assume the search starts at the primary block and continues through the overflow chain until the record is found.

Problem Setup:

We need to determine **how many records must be inserted** into a **sequential data file** with **512 records** stored in **primary blocks initially 75% full**, such that the **average number of examined blocks = 3**. Overflow blocks are created as needed, and the search starts at the primary block and continues through overflow blocks until the record is found.

Step 1: Analyze Initial State

1. **Primary Block Utilization:**
   - Primary blocks are **75% full**.
   - Total records in primary blocks initially = **512 records**.
   - Since these blocks are **75% full**, the **total capacity of the primary blocks** is:

$$C = \frac{512}{0.75} = 682.67 \approx 683 \,\text{records}$$

2. **Unused Space:**
   - Remaining space in primary blocks = **683 - 512 = 171 records**.

Step 2: Establish Lookup Probabilities

Search Process:

- The search first checks the **primary block**.
- Continues through **overflow blocks**, one by one, until the record is found.

We calculate the **average number of examined blocks** based on proportions of records in **primary and overflow blocks**.

Step 3: Fill the Primary Blocks

Phase 1: Fill the Remaining Space in Primary Blocks

- We add **171 records** to fill the primary blocks completely.
- Total records after this step = **512 + 171 = 683**.

Phase 2: Begin Adding Overflow Blocks

- Overflow blocks start receiving records after the primary blocks are full.

Step 4: Calculate Average Examined Blocks

Case 1: Insert Overflow Records

- Assume **1 overflow block**, and let's calculate the average:

1. Primary Block:
   - Contains **683 records** (already filled).
   - Proportion: $P_1 = \frac{683}{(683+x)}$, where x is the number of overflow records.
2. Overflow Block 1:
   - Proportion: $P_2 = \frac{x}{(683+x)}$.

Average blocks examined:

$$E[B] = P_1(1) + P_2(2)$$

We need **E[B] = 3**.

Step 5: Solve for x (continued)

The formula for the average number of examined blocks is:

$$3 = \frac{683}{683 + x}(1) + \frac{x}{683 + x}(2)$$

Multiply through by $683 + x$ to eliminate fractions:

$$3(683 + x) = 683(1) + x(2)$$

Simplify each term:

$$2049 + 3x = 683 + 2x$$

Rearranging terms:

$$2049 - 683 = 2x - 3x$$

$$1366 = -x$$

$$x = 1366$$

Step 6: Total Inserted Records

Since **171 records** were added to **fill the primary blocks**, and then we inserted **1366 records** into **overflow blocks**, the **total number of inserted records** is:

$$171 + 1366 = 1537$$

Step 7: Final Answer

We need to insert **1537 records** before the **average number of examined blocks** reaches **3**.

This accounts for filling the **primary blocks** first and then distributing records into **overflow blocks** until the desired average is achieved.

4. Blocks can store either five records or 20 key-pointer pairs. Given n, the number of records, calculate the minimum number of blocks required for:
   a) The data file.
   b) A dense index.
   c) A sparse index.
   You may ignore inaccuracies caused by rounding.

---

5. Consider a block of 4096 bytes used for storing records of size 200 bytes each. The block header is 32 bytes, and an offset table requires 2 bytes per record.
   a) How many records can the block store under unspanned storage?
   b) If records are deleted at random and tombstones are used, how does this affect the total capacity for new records after 25% of the records have been deleted?

6. Suppose a file contains 1 million records and uses a primary index with key-pointer pairs. Each block stores 128 key-pointer pairs, and the key is 8 bytes while the pointer is 4 bytes. Calculate:
   a) The number of blocks required for the primary index.
   b) The number of blocks required for a sparse index if each sparse key points to a block in the primary index.

7. Imagine you are working with a database system where records of variable lengths are stored using a slotted-page structure. Each slot pointer requires 2 bytes, and the block header is 48 bytes. A block has a total size of 2048 bytes. How many variable-length records can be stored in a block if the average record size is 200 bytes?

8. Consider a block of 8192 bytes with a block header of 128 bytes and a record format as follows:
   - 4-byte integer ID
   - 8-byte double value
   - 2-byte short value

Assume records must align to 4-byte boundaries. How many records can be stored in a single block?

9. Discuss the implications of using dense vs. sparse indexes for a database with 10 million records. Consider factors such as the size of the index, lookup time, and block access during searches.

10. A file contains nnn records, and a secondary index is used to support range queries. Each key-pointer pair in the index requires 12 bytes. If the block size is 4096 bytes, calculate the number of blocks required for the secondary index and discuss how the index structure affects range query performance.

These questions are designed to build upon your understanding of DBMS concepts, focusing on record storage, indexing, and block management. Let me know if you'd like further refinements or additions!

Here are two new questions inspired by the given **Q.3 in Sheet-4** example:

Question 3a:
Suppose a sequential data file contains n records, with each block initially 60% full. When new records are inserted, overflow blocks are created as necessary. If records are inserted at random, how many additional records must be inserted before the average number of blocks (including overflow blocks) accessed during a lookup reaches 2.5? Assume that lookups search the primary block first, followed by overflow blocks in order, until the record is found.

Question 3b:
A sequential data file of n records is stored in blocks that are currently 75% full. Overflow blocks are used to store additional records when a block is full. Records are inserted in a random order, and all blocks in the file have an equal probability of receiving the new record. If each lookup starts with the primary block and sequentially searches overflow blocks, how many records need to be inserted for the average number of blocks searched during a lookup to reach 3?