1. The workloads these systems support share several characteristics: they are I/O, not computation, intensive, requiring random access over large datasets, ..., and the size of objects stored is typically small.” Read the above statement, indicate why workloads of these characteristics represent a challenge to the system design?

Ans:

- When the object becomes small, the respective metadata size becomes more compared to the size of the data.
- The poor seek performance of the disks make them inefficient in terms of both system performance and performance per watt.
- Clusters which also include the DRAM are expensive and consume a surprising amount of power, because they require a lot of storage space and which become expensive.

2. The key design choice in FAWN-KV is the use of a log structured per-node datastore called FAWN-DS that provides high performance reads and writes using flash memory.” “These performance problems motivate log-structured techniques for flash filesystems and data structures” What key benefit does a log structured data organization bring to the KV store?

Ans:

- By using a log-structured datastore, random writes are replaced with sequential writes.
- This way the throughput of the SET and GET actions increases significantly.

3. “To provide this property, FAWN-DS maintains an in-DRAM hash table (Hash Index) that maps keys to an offset in the append-only Data Log on flash.” What are potential issues of the design? [Hint: consider metadata size and volatility of DRAM.]

Ans:

To reduce the memory requirement.
For large metadata, the size of DRAM is not sufficient to hold the metadata. Therefore it maintains a hash table in the DRAM to map keys to offset in the Data log on flash.

4. “It stores only a fragment of the actual key in memory to find a location in the log;” Is there correctness concern in this design? 

Ans: No. In terms of correctness, this design is not wrong.

- First, the FAWN-DS checks the first 5 bits to find a location in the log and verifies that the key it read was correct.
- If the key matches then there would not be any problem, otherwise if the key does not match then the key fragment is searched again.
- Mostly, there would not be an issue of searching twice. It happens only once in $2^{15}$ times. Even this would not be an issue since flash is fast in reading (up to 2000 reads per second).

5. “Basic functions: Store, Lookup, Delete” Use Figure 2(a) to explain how these basic functions are executed?

Ans:  
- **Store**
  - Store adds an entry to the log, updates the corresponding hash table entry to point to this offset within the Data Log, and sets the valid bit to true.

- **Lookup**
  - Lookup retrieves the hash entry containing the offset and index into the Data Log and then returns the value(data blob).

- **Delete**
  - Delete invalidates the hash entry by clearing the valid flag and adds a Delete entry to the end of the log.

6. “As an optimization, FAWN-DS periodically checkpoints the index by writing the Hash Index and a pointer to the last log entry to flash.”. Why does this checkpointing help with the recovery efficiency? Why is a Delete entry needed in the log for a correct recovery?
Ans:

- Deletes do not immediately reclaim space.

- They periodically perform garbage collection and then reclaim the space through compaction.

- If there is any failure or any kind of data loss, all of the Hash Index must be recovered totally from the Data log, which takes a lot of time when the Data log is huge.

- So, it checkpoints the index by writing the Hash Index and a pointer to the last log entry to the flash, which can then use the checkpoint as a starting point to reconstruct the in-memory Hash Index quickly.

- The delete entry is necessary for fault tolerance.