DESIGN DOCUMENT - PART A

My Storage Engine is optimized for **balanced** workloads. My implementation has comparable read and write speeds applicable to production systems.

DATA STRUCTURE

- ➔ My Storage Engine is entirely based on **LSM Tree** data structure.
- ➔ The in-memory (**MEMTable**) key-value storage structure is an ordered map of STL.
- ➔ The disk key-value storage structure is a Sorted String Table (**SSTable**).
- ➔ The SSTable is also implemented using an ordered **map** of STL.
- ➔ Each SStable has a **BloomFilter** attached to it for probabilistic membership testing.
- ➔ The BloomFilter uses **bitset** of STL for its implementation.

CONFIGURATION

- ➔ MEMTable capacity is **10000** key-value pairs.
- ➔ SSTable capacity is **10000** key-value pairs.
- ➔ BloomFilter bit array size is **200000**.
- ➔ BloomFilter uses **9** hash functions.

WORKFLOW - READ

1. Search key in the MEMTable.
2. If key found in MEMTable return the value.
3. If key is not found in MEMTable, search the SSTables in order latest to oldest.
4. If BloomFilter says that the key might be present in the SSTable then only the SSTable is searched.
5. Searching continues till the key is found or all the SSTables in the disk are searched.
6. If a tombstone value is found for the key, it returns NULL as the key has already been deleted.

WORKFLOW - WRITE

1. The key and value pair is inserted to the MEMTable.
2. If the MEMTable reaches capacity, a new SSTable is created in the disk and all the entries in MEMTable are written to the SSTable and MEMTable is flushed out.

WORKFLOW - DELETE

1. A tombstone value is inserted in the MEMTable for the key to be deleted.