Explain the different sorting strategies in Spark and which strategy you will be adopting when joining Parquet File 1 and 2 if you are implementing the code in Spark Dataframe

A. Sorting strategies in Spark

1. Sort-merge Join

Performs a merge join similar to the one used in traditional databases.

- Requirements:
  - When both DataFrames being joined are sorted based on the join key
  - Requires both DataFrames to be partitioned and sorted by the join key
- Advantages:
  - Efficient when the DataFrames are pre-sorted as it avoids shuffling
  - Can lead to better performance compared to other join strategies.

2. Shuffle Hash Join

Partitions both DataFrames based on the join key and redistributes the data across the cluster, then performs a hash join operation on the partitioned data

- Requirements:
  - When neither of the DataFrames can fit entirely in memory
  - Requires both DataFrames to be partitioned and sorted by the join key
- Advantages:
  - Memory efficient
- Disadvantages:
  - Significant network overhead due to data shuffling

3. Broadcast Hash Join

Broadcasts the smaller DataFrame to all worker nodes and then performs a hash join operation with the larger DataFrame

- Requirements:
  - When one of the DataFrames is small enough to fit entirely in memory across all worker nodes
- Advantages:
  - Efficient for small tables as it avoids shuffling and network overhead.

4. Broadcast Nested Loop Join

Broadcasts the smaller DataFrame and performs a nested loop join with the larger DataFrame.

- Requirements:
  - When the size of one DataFrame exceeds the broadcast threshold, making it too large to fit entirely in memory across all worker nodes for a broadcast hash join

- Advantages:
  - A fallback option when other join strategies are not feasible due to memory constraints or data sizes.
- Disadvantages:
  - Less efficient compared to other methods
  - Can lead to performance issues
  - Unsuitable for broadcasted DataFrame that is significantly larger than available memory or if the join key is not selective enough

B. <u>Assessing Current Datasets</u>

1. Dataset A

Table Size: Approximately ~1 million rows

Column Name	Column Type	Comment
geographical_location_oid	bigint	A unique bigint identifier for the geographical location
video_camera_oid	bigint	A unique bigint identifier for the video camera that the item was detected from.
detection_oid	bigint	A unique bigint identifier for each detection event.
item_name	varchar(5000)	Item name
timestamp_detected	bigint	timestamp for a given timestamp detected

2. Dataset B

Table Size: Approximately 10000 rows

Column Name	Column Type	Comment
geograhical_location	bigint	A unique bigint identifier for the geographical location
item_rank	varchar(500)	Item_rank = 1 corresponds to the most popular item detected in geographical_location
item_name	varchar(5000)	Item name

3. Conclusion

Dataset A has a large number of data and may be too large to fit the memory on load. Dataset B is significantly smaller in size as compared to Dataset A. Hence, the **Broadcast Hash Join** will be a suitable sorting strategy in this case, where Dataset B will fit entirely into the memory. Since Dataset B is smaller and can be efficiently broadcasted, the join operation can be performed locally on each worker node, resulting in a significant reduction in data shuffling and network overhead.

The resulting joined dataset is likely to have a size close to Dataset A, as it's the larger dataset. This means that broadcasting Dataset B would still be feasible, as it wouldn't result in excessive memory usage on the worker nodes.