**Installation instruction**

1. Unzip the file
2. Import the directory into Eclipse as an existing project
3. Build the project
4. Test cases from Task 3 and 5 are under \test\project2\ package.
5. For Task3, Run test.project2.CreateTestTables.java. This runnable function would clear the directory and initialize SimpleDB automatically, then create 5 new tables with 3 different indices, insert values into tables. Also it would run SelectTestTables.java and JoinTestTables.java. These two test would process select and join with tables in different indices, and print out the time cost of each.
6. For see the JUnit Test file, \test\project2\task4\Test Task4.java

**Design description**

**Task1**

Task 1 is relatively easy. Basically we just follow the instruction and implement it.

**Task2**

For task2, we build a two-level index to implement it. The external index (ExtHashIndex.java) stores the index of the directory, including its corresponding key of the bucket and its local depth. The internal index (SecHashIndex.java) stores the all the records inside the bucket.

The insertion and deletion would be passed to internal index to implement. However, if one bucket is full, the internal index would tell the external index to split the bucket by deleting all elements in the bucket, creating a new bucket and reinsert deleted elements. Also, it would increase the global index by adding new indices as needed.

During task2, we struggled for many times. For example the generation of local depth and global depth is pretty tricky. Sometimes we need the global depth dynamically and sometimes we need a static one. This task is kind of complicated and indicated us the importance of testing

**Task 3**

Task3 is about testing. Nothing special need to be designed.

**Task 4**

Idea 1: the first merge-join phase will run in open() function in MergeJoinPlan.java. The sorted flag in TableInfo will be used there to decide whether need to go through the first phase.

Idea 2: MySortScan.java is an extension of the original SortScan.java will contain the updateRecordFile() function, which is able to perform sorting on the original table based on the new generated templateTable.

The difficulties here is to figure out where to copy new template RecordFile to the original Table. This action has to be done after the first phase of merge join is completed.

**Task 5**

Two table to join: test1 & test2.

Two tables have the same primary keys with the range starting from 1 to the numRecordToInsert (a variable in source code). However the values/key were put in the reverse/opposite order in these two files, with the purpose to guarantee that the full functionality of merge-join will be tested.

Two Querys/Plan for MergeJoinPlan() job:

SELECT a1,a2 FROM test1

SELECT b1,b2 FROM test2