**Overall Status**

\_Delete has been implemented, it takes the key as well as the record id, Delete all the keys that match (Duplicate keys).

I have researched and read codes that were online, also created my own code for in memory implementation of the algorithm.

\_Delete is a recursive function, which calls itself recursively till it reaches the leaf that contains the key. If key exist then it deletes it and check if the leaf underflows.

I have created a private method checkIfUnderflow method which is common for leaf and index. It checks for 50% fill ratio, it also adds the space that was deallocated by removing the slot that was holding the pointer to the key.

If it underflows first it tries to borrow from sibling leaf, if not possible, check if merging is possible.

For merging leaf there is another private function which takes all the keys from right sibling and inserts into the left sibling. The key in the parent pointing to right sibling is returned.

If merging returns a key, which is the key that needs to be removed from the parent, we delete it from the parent and check for underflow.

We again check for redistribution and merge as above and each level returns the value to the parent level, till it reaches the root, from where the recursive call started.

We also update the index if deleting from the leaf, so that it points to the latest and it also helps if there are duplicate records.

**File Descriptions**

No new file is created, though some private functions were added.

**Division of labor**

I was the only team member so there is no division, the whole code was implemented by me. It took me around 30-40 hours to understand the codebase, run it on my computer and omega, and coding.

**Logical Errors**

1. One exception which was quite frequent was pin and unpin exception. If I tried to free a page which was still pinned by someone other call, it would throw an error. So I unpinned any page as soon as it was not required at that time.
2. The index pages are not connected to each other, like leaf pages are connected as doubly linked list. As both have getPreviousPage, it gives a wrong impression as if it is pointing to its sibling, whereas only leaf page points. Initially I tried to make index point to its sibling due to which it lost its pointer to child.
3. The implementation checks for sibling that is left until unless the leaf itself is the leftmost, so we can never check if it can have borrowed from the other sibling.
4. If we try to insert a duplicate key that is exactly at the midpoint of the leaf that is full, we have the last of left child and first of right child having the same key, which means parent node always point that the key exist on the right of it, but in reality it is also present on the left. This issue is also propagated up the tree, as the parent can have the same key on the left child of it. This is due to implementation in insert. I update the index whenever the first child of a leaf is deleted, which somehow reduces the error, but if there are a lot of duplicates, this issue can be seen.
5. Fan out still remains an issue, if it is above .5 it cannot be followed while deleting.
6. There are few functions that are common to leaf and index and could have been moved to parent class, to allow polymorphism. Due to duplication of this, I had to write 2 different methods for leaf and index for merging.