This assignment is a collaboration between:

- Megan Fantes (U56999246)
- Efstathios Karatsiolis (U65214507)

PA 2
Design Document

Describe any design decisions you made. Discuss and justify any changes you made to the API. Describe any missing or incomplete elements of your code.

- BufferPool.evictPage()
    - We get a list of the keys in the ConcurrentHashmap and then remove the first key. This then allows us to add another key to the hashmap.

- BTreeFile.findLeafPage()
    - Based on the category of the page passed into the method, we either return the current page (if the category of the page is a Leaf), or we recurse to one of the children of the current page to continue searching.
    - If the field we are looking for is null, we return the left-most leaf page

- BTreeFile.splitLeafPage()
    - If we want to insert a new tuple into a full leaf page, we create a new page to the right of the current page and put half of the values onto the new page.
    - When we add the key value for the new page to the parent: if the parent is full, we recurse upwards until we can add a new kay value to a parent that is not full. (This uses splitInternalPage.)

- BTreeFile.splitInternalPage()
    - We create a new, empty page to the right of the page we want to split, then we move half of the values to the new page. We then find a parent page with an empty slot, and recursively split pages and bubble up key values. After that, we update all sibling and parent pointers.

- BTreeFile.stealFromLeafPage()
    - We figure out how many more tuples the given sibling page has, then taken half of them.

- BTreeFile.stealFromLeftInternalPage()
    - Figure out how many more entries the left sibling has than the current page, and take half.
    - Iterate backwards through the entries of the left sibling, move them to the new page, and iteratively update their child pointers so the values point to the appropriate left and right children.
    - Once done, update the parent pointers for both the current page and the left sibling.

- BTreeFile.stealFromRightInternalPage()
    - Same as above, except for the right sibling, and we iterate forwards instead of backwards.

- BTreeFile.mergeLeafPages()
    - We move all entries in the given right page to the given left page, then mark the right page as empty. We then delete the parent entry and right child that pointed to the left page with the method deleteParentEntry(), which cleans up the tree and handles reorganization if necessary.

- BTreeFile.mergeInternalPages()
    - We first create a new entry on the given left page that points to the right-most child on the left page and the left-most child on the right page. This new entry ensures that the final merged page will be one seamless block of data, without this new entry, there would be a gap between the data of the left and right pages that would not have the appropriate pointers.
    - From here, the logic is the same as the mergeLeafPages() method above.

Describe how long you spent on the lab, and whether there was anything you found particularly difficult or confusing.

- This assignment took us about 30 hours of coding between 2 people.
- It is difficult to understand what methods we have available to us. It would have been nice to have time in our discussion sections to walk through BTreeFile.java and talk about the methods in the file, which ones are already implemented, and a rough idea of what the methods would help us do. The comments in the code for this assignment were very helpful in giving us hints for what methods to use, it just would have been nice to have that information reinforced in discussion.