Project 1

Name: Wen-Yuh Su UIN: 671937912

After implementing the project 1, and doing some experiments on it, I have some results which show in table1.

First of all, the results show that the Bε tree with ε = 1/2 has the least I/O counts in the insert operation which means it would cost the least time for I/O. Yet, for the lookup operation, it is almost the similar to the B+ tree. In addition, I found that all of three methods on the negative lookup is almost the same, since when finding the non-exists key, the program has to find until the leaf of the tree. In this case, as a result, they all need the mount of I/O counts to lookup.

Secondly, for the Bε tree with ε = 1/4, it performs not as well with the ε = 1/2. The reason is that it has to spend more I/O to look into deep tree so that it will cost more I/O counts than the Bε tree with ε = 1/2 because of the less buffer it has. But I think I might have some mistakes here, since the I am not sure whether the Bε tree with ε = 1/4 would perform worse than B+ tree.

Eventually, the number of the key would also affect the performance of the Bε tree because if the data is not large enough, the Bε tree would not utilize the capacity of the buffer. Thus, when using insert or lookup operation, the program would store data in the leaf and resulted in that Bε tree does not improve anything.

	B+ tree	Bε tree(ε = 1/4)	Bε tree(ε = 1/2)
data		1000000	
insert	5304441	6810731	4919968
lookup	5000000	5887267	3940112
non lookup	5000000	6000000	4000000
data		5000000	
insert	31166936	39025471	29539435
lookup	25000000	34331479	24700075
non lookup	25000000	35000000	25000000
data		10000	
insert	40991	37988	33655
lookup	30000	29116	29399
non lookup	30000	30000	30000
			Unit: I/O counts

table 1