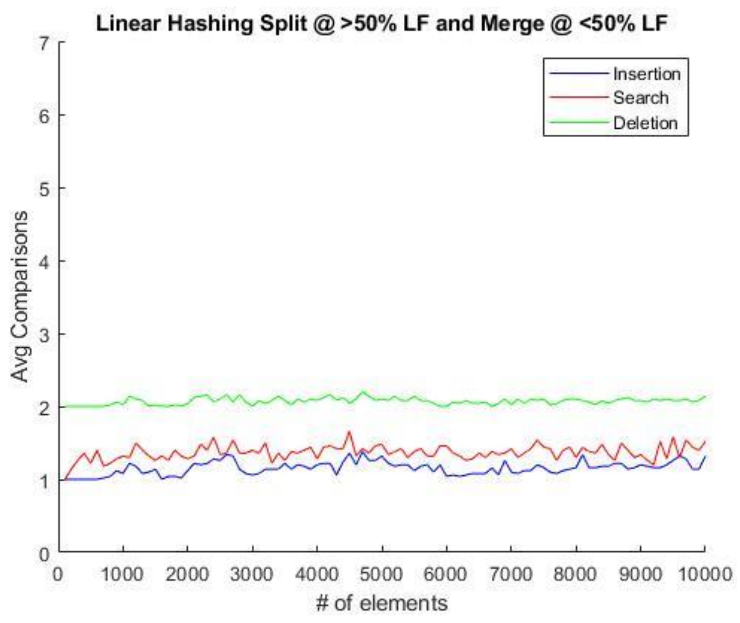
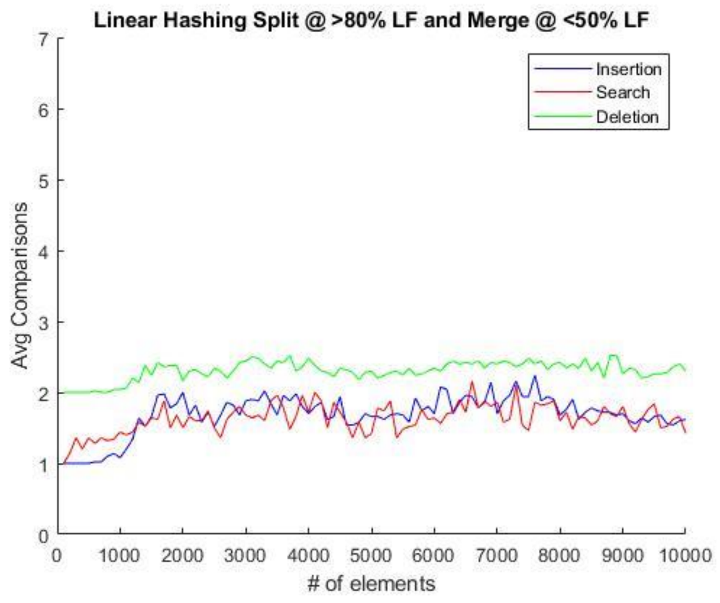
Results:

Linear Hashing Split Load Factor > 50% Merge Factor < 50%:



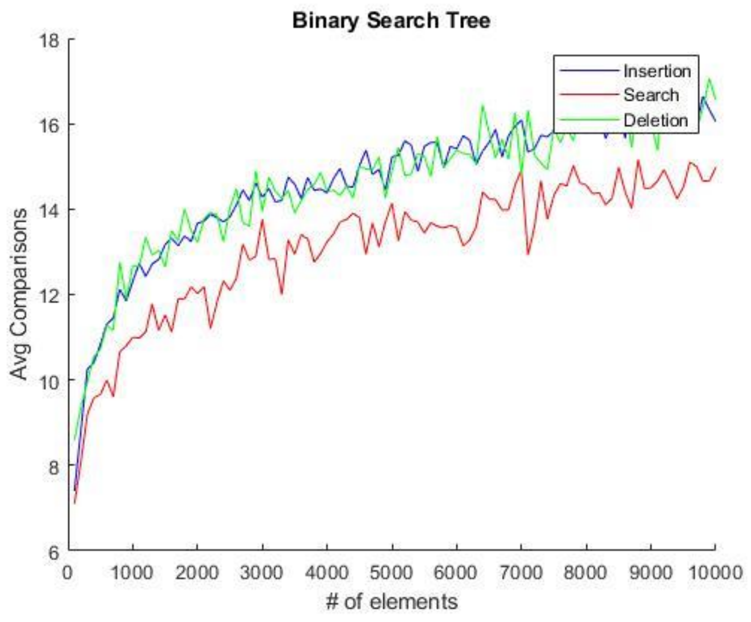
Linear Hashing Split Load Factor > 80% Merge Factor < 50%:



Observations and Comparisons:

- In both graphs the numbers of average comparisons for Insertion/Search/Deletion remains constant and independent of the number of elements. Thus, confirming the complexity of a HashTable which is O (1).
- Increasing the limit of the Split Condition allows more overflow Buckets to be created, slightly increasing the comparisons of the Insertion/Search and marginally the comparisons for a deletion, whilst still remaining constant.
- The numbers are really small because I'm using Java's built in methods for checking if an Element is present in a Vector, deleting an Element in a specific index, clearing the Vector, etc.
- One thing that's not apparent from the graph is the memory Usage of the two implementations. The implementation with the lowest Split Condition uses more memory because it prevents the creation of overflows. On the other side the large Load Factor has more overflows, thus it takes

more time to search for a key that Hashes on that specific Bucket. A Load Factor at 70% is proven to be a great balance between speed and memory use.



Linear Hashing Vs Binary Search Tree:

- Observing the graphs of the BST we can confirm the Complexity of all 3 functionalities to be O (log n).
- We need much more computations to complete each task required due to the "tree" structure (always starting from root and working our way downwards).
- The BST doesn't use excess memory because it allocates as much as it needs.
- The overall construction and maintenance of a BST is easier. On the HashTable every time we delete a key, we need to rearrange the vectors and re-align them for the next insertion (cons of using Vectors). On the BST is a matter of distinguishing one of three cases with minor difference between them.
- BST's are great for algorithms and large Datasets where sorting is required or ranged queries.
- While HashTables are faster, we need continuous ram blocks in order to store them, because of their "Linear" like structure (Just the table not the overflows).