CS304 Assignment 5 – due Tuesday April 12th at 11:59 PM

**Comparing map performance with different underlying data structures**

In this assignment, you will implement the **map** ADT using the following underlying data structures:

1) BST (binary search tree)
2) Hash table (separate chaining)
3) Hash table (linear probing)
4) Skip list

You may use the hash table implementations from the notes (lecture 16 and 17). Base your BST code on the pseudocode in lecture 14 notes. Base your skip list code on the pseudocode from lecture 18 notes.

The operations supported by map ADT are below:

**M[k]:** Return the value v associated with key k in map M, if one exists; otherwise raise a KeyError. In Python, this is implemented with the special method __getitem__.

**M[k] = v:** Associate value v with key k in map M, replacing the existing value if the map already contains an item with key equal to k. In Python, this is implemented with the special method __setitem__.

**del M[k]:** Remove from map M the item with key equal to k; if M has no such item, then raise a KeyError. In Python, this is implemented with the special method __delitem__.

**len(M):** Return the number of items in map M. In Python, this is implemented with the special method __len__.

**iter(M):** The default iteration for a map generates a sequence of *keys* in the map. In Python, this is implemented with the special method __iter__, and it allows loops of the form, **for** k **in** M.

Your goal is to implement these operations using the 4 data structures listed above (BST, hash tables, and skip list). you can omit the iter operation because we did not cover this in class. Also, this is not an *ordered* map ADT, so the hash tables *should* perform better (skip list and BST can be used for unordered and ordered map implementations, while hash table is used exclusively for unordered map).

The hash table should provide $O(1)$ performance for all map operations. Skip list and BST should provide $O(logn)$ performance for $M[k]$, $M[k] = v$, and $del\ M[k]$. Your job is to verify these time complexities by 1) implementing the map using the 4 data structures listed above and 2) performing timing experiments like you did in assignment 4 and plotting the resulting curves.

In addition to your code in .py format, you should provide a pdf showing the results of your timing experiments.

**Timing experiments:** Generate 10,000 random key-value pairs (keys can be anything that is comparable (float, string, int, etc.), value can be anything) and insert them into your map, getting the time for each insertion (before the first insertion, your map will have 0 elements, before the last insertion, your map will have 9,999 elements). Provide three plots in your pdf:

1) Number of elements on x-axis, time in ms on y-axis for **M[k] = v operation** (insertion). The plot should have 4 curves (red = BST, blue = separate chaining hash table, green = linear probing hash table, magenta = skip list).
2) Same as (1) but for **M[k]** operation (return value). Perform each M[k] operation after inserting the element (get the time it takes to access an element directly after inserting it), so you can have number of elements on the x-axis like in figure 1.
3) **del M[k].** after inserting all your elements so your map has 10,000 elements, delete them one by one by using del M[k]. the x-axis of this plot is the number of elements and the y-axis is time, similar to figure 1 and 2 (use same color scheme also).

Comment on your plots. Do they show the expected O(log n) and O(1) for the BST/skip list and hash tables, respectively? Which hash table implementation (separate chaining or linear probing) is faster?