Assignment 1.3

(a) After n operations, the maximum size of DLL can be O(n). Now the worst case time complexities of these function can be :-

1.) Insert :- O(1) . Since only 4 operations are done.

2.) getFirst :- O(n) Since we iterate back till the head sentinel node from current node.

3.) GetNext :- O(1) Just returns next or null if next is tail sentinel.

4.) Delete :- O(n) calls the getFirst function which itself has O(n) complexity. After this iterates in DLL, if last element is the one needed then ~~complexity will~~ n operations Hence complexity is O(n).

5.) ~~f~~ Find :- O(n) getFirst of O(n) is called. Also whole DLL is being searched & thus n operations in worst case. So O(n) complexity.

6.) Sanity :- O(n) Because for Loop Detection Floyd's Algorithm will take atleast n/2 iterations if there is no loop. Also the process of checking curr. next. prev =? curr will also take O(n) time.

So ~~ext~~ Insert & getNext take O(1) time and the rest take O(n) time, if n is the size of DLL.

(b) * AllocateBlk worst size can be O(n), for example in all the n operations I just kept doing Allocate 1.

* FreeBlk can have a worst size of O(n) also. Let's say we did Allocate O(1) for n/2 operations. Now for the remaining n/2 we can keep freeing these allocated blocks.

1) Allocate → we first search for a Block(node) of appropriate size in the whole freeBlk DLL. For that we use Find function. It takes O(n) time to run. Thus O(n) complexity.

2) Free → Again using Find we search in the AllocBlk for a particular key. Thus since Find takes O(n) time, free function will also take O(n) time.

A3.2 After 'n' operations the maximum size of allocBlk & freeBlk can be O(n). Hence in our further analysis we will consider size is O(n).

complexity of BST functions. if n is the total no. of nodes & 'h' is the height.

(a) Insert → O(h). At each step we go either left or to right (or even null if we are inserting a duplicate). Hence complexity = O(h). h = O(n). So worst case becomes O(n).

(b) Delete → O(h). Because we search for key. Now getNext() returns successor, so it is also O(h).

(c) find → we have only 2n, right or left until found. Hence maximum nodes we check are O(h). Thus time can is O(h).

~~(d) getfirst() →~~

(d) getRoot() → Travels till the Root & hence O(h).

(e) getFirst() → travels only left & hence O(h) complexity.

(f) getNext() → finds the successor & hence either travels up or travels left. Thus complexity is O(h).

(g) compareNode or compare-dict is a O(1) complexity function.

(h) Sanity → Sanity is a O(n) function.

In all these for a BST h can be worst case O(n). Hence they are O(n) functions.

Now in FreeBlk and AllocBlk

(i) AllocBlk → O(h) since we are using the find, insert & delete functions a constant no. of times. Each of which takes O(h) time. Hence the TC is O(h) or O(n) of AllocBlk when we use BST tree to implement the dictionary.

(ii) FreeBlk → O(h). Again we are just using the insert, find and Delete functions. Each of them works in O(h). Hence the time complexity is O(h). In worst case the TC is O(n).

(iii) Defragment → first we copy the whole freeBlk into an address indexed. This Requires inorder traversal & also at each step insert is being done. Insert is O(h). So this Requires O(n·h) time. After if we do an inorder traversal & in that we do the insert, delete operations during each iteration of traversal. Hence each step of traversal Requires O(n) time & the travel takes O(n) time. So total time is O(nh) Hence, time complexity will be O(nh) & in the worst case it becomes O(n²).

A3.3 AVL tree is an Extension of BST Tree. In this maintain the height property which ensures that $h = O(\log n)$ for AVL Tree.

Now substituting this in the time complexities we analysed in BST Tree.

(i) Insert → putting the node takes $O(\log n)$ time (just like in BST). Now we call the updateHeight function which starts a while that runs maximum $O(\log n)$ time, since travels only to root through parents. In this it can call Rebalance function. Now Rebalance does a lot of operations but they independent of 'n'. Hence TC of Rebalance is $O(1)$. So TC of updateHeight is $O(\log n)$. Thus the time complexity of Insert remains $O(\log n)$ even after maintaining the height property. (ie max diff in height of left & right child is max 1). TC of Insert $= O(\log n)$

(ii) Delete → First we find the Node to be deleted like BST & it takes $O(\log n)$ time. In the end we call the updateHeight which takes $O(\log n)$ time. Hence TC of Delete is $O(\log n)$.

(iii) Find → $O(\log n)$

(iv) getRoot() → $O(\log n)$

(v) getFirst() → $O(\log n)$

(vi) getNext() → $O(\log n)$

(vii) Sanity() → $O(n)$

(viii) compareDict & compareNode → $O(1)$

Thus the TC of :-

(a). AllocBlk → O(logn). In BST it was O(h) and h = O(logn). Thus TC is O(logn)

(b) FreeBlk → O(logn). In BST it was O(h) and h = O(logn). Thus TC is O(logn).

(c) Defragment → O(n logn). Because in BST we saw it was O(n·h) & here h = O(logn). Thus TC of Defragment is O(n logn).