COL106: Assignment 1.3

Sayam Sethi (2019CS10399)

November 2020

1 A1.3.1

If n operations are performed on the A1List, the size of the A1List will be $O(n)$ (since atmost n insertions are possible).

1.1 `Insert`

For insertion, the node is inserted after the current node. Performing this operation requires creation of a new node, and pointer updates for the current node, the node after the current node and the newly inserted node. Time taken for creation of a new node is $O(1)$ and total number of pointer updates are 4. Therefore, the complexity of `Insert` is $O(1)$.

1.2 `getFirst`

`getFirst` traverses to the head of the A1List and then checks if the next node is an actual node or a tail node. The worst case time for traversal will be when we are at the node just before the tail sentinel taking $O(n)$ steps. All other operations are $O(1)$ and hence total complexity of `getFirst` is $O(n)$.

1.3 `getNext`

`getNext` performs two checks and return `null` or `this.next` depending on the checks. This is trivially $O(1)$.

1.4 `findNext`

This method performs $O(1)$ check with every nodes that appear at or after the node from which this method is called. In every iteration, `getNext` is called which is $O(1)$. Thus, the total complexity of every iteration is $O(1)$ and in the worst case, the number of iterations will be equal to the size of the list leading to the total complexity of $O(n)$.

1.5 `Find`

In `Find`, `getFirst` is called once which is $O(n)$. After this, in the worst case `findNext` is called which can attain its worst case complexity leading to the final complexity of `Find` to be $O(n)$.

1.6 `isEqual`

This method performs $O(1)$ checks of $O(1)$ each and hence has a complexity of $O(1)$.

1.7 `Delete`

The initial call of `getFirst` is $O(n)$. After this, the total complexity of the while loop is $O(n)$ since `findNext` only moves the pointer forward thus the total complexity of all `findNext` calls is $O(n)$, also the total complexity of all `isEqual` checks is $O(n)$ since it will be called on every node in the worst case. These both together lead to the worst case time complexity of $O(n)$ for the `Delete` method.

1.8 `isCyclic`

This is a direct implementation of Floyd's algorithm for loop detection and hence is $O(n)$.

1.9 `sanity`

`sanity` first calls `isCyclic` which is $O(n)$, then calls `getFirst` (this is a valid call since non-cyclicity is ensured) of $O(n)$. After this, one complete forward iteration of the list happens with $O(1)$ operations in each iteration leading to $O(n)$ total complexity of the iteration. Adding all these, the complexity of `sanity` turns out to be $O(n)$.

2 A1.3.2

After n operations, the size of `freeBlk` and `allocBlk` will be $O(n)$ since every call of `Allocate` leads to an increase in the size of `allocBlk` without changing the size of `freeBlk` in the worst case leading to $O(n)$ size of `allocBlk`. Similarly, every `Free` operation leads to an increase in the size of `freeBlk` and a decrease in the size of `allocBlk`. Hence $\Theta(n)$ `Allocate` calls and $\Theta(n)$ `Free` calls will lead to $O(n)$ size of `freeBlk` and `allocBlk` both.

2.1 `Allocate`

Since $size$ of `freeBlk` is $O(n)$ and `Find` method is $O(size)$, this step takes $O(n)$ time. `Insert` takes $O(1)$ and `Delete` method will also take $O(size)$ which will

be $O(n)$ in this case. This leads to the total complexity of `Allocate` to be $O(n)$.

2.2 `Free`

Since *size* of `allocBlk` is $O(n)$ and `Find` method is $O(size)$, this step takes $O(n)$ time. `Insert` takes $O(1)$ and `Delete` method will also take $O(size)$ which will be $O(n)$ in this case. This leads to the total complexity of `Free` to be $O(n)$.