Assignment 1 - COL106

Name - Shrey Patel
Entry No. - 2019CS10400
Group - 9

A 1.3

1.3.1

We assume n operations have already been performed. We need to find the worst case complexity of each of the operations, so we assume maximum size of list, which can be constructed from 'n' elements using 'n' Insert operations.

(1) Insert (at a given node)

→ Insert operation requires us to only update some pointers so that the new node can be added just ~~aft~~ between the given node and its next node in the original list.

So, Time complexity = O(1)

(2) getNext (for a given node)

→ getNext returns the next node or 'null' depending on the 'next' pointer of the given node. So, Time complexity = O(1)

③ Delete (~~a given~~ a node with given key, address and size)

→ Delete operation first requires to search the list for the node with the given key, and then delete it if both the address and size of that node also match. And repeat this process until all such instances of the node in the list have been deleted.

→ Because this operation requires us to traverse the whole list,

Time complexity = O(n)

④ Find (a node with given key)

→ Similar to delete, find requires us to first search the list for a node with the given key and return null if no such node is found. So, as traversing the list takes linear time,

Time complexity = O(n)

⑤ getFirst (given a node)

→ getFirst operation on a node returns the first node of the list of which the given node is a part of or 'null' if the list is empty.

Q-3

→ So, we may need to traverse the whole list backwards, starting from given node to the first node, using 'prev' pointers. So, Time complexity = O(n)

(6) sanity (of a given list)

→ Sanity operation is made up of several sub-operations:

1) checking if head.prev = null → O(1)
2) checking if tail.next = null → O(1)
3) checking if node.prev.next = node and node.next.prev = node for every node in the list → O(n) because we check for each of the n nodes
4) checking presence of loops → O(n) (by two pointer method)

→ At every step, the distance b/w (in forward direction) the 2 pointers increases by one, and if there is no loop, one of the pointers reaches the end of the list.

→ If there is a loop, the 2 pointers become equal and the distance b/w them will be the size of the loop which is at most 'n'. So, no. of steps required are at most 'n'

5) check duplicates

Using Hashing → $O(n)$
Using comparing → $O(n^2)$

Overall time complexity of sanity:
$O(n)$ or $O(n^2)$ depending on the type of check duplicates function

1.3.2

① Allocate (a given size):

→ Allocate operation first searches the free memory blocks list to check if there exists a contiguous memory block of required size or higher. This requires us to traverse the freeBlk list which is of maximum size 'n' (formed from n free operations)

So, Time required = $O(n)$

→ If such block doesn't exist, nothing more to do. But if it does, we check if the required size is less than the found memory block. If it is, we split the found block into the required size (takes $O(1)$ time).

→ The found memory block is then removed from freeBlk and inserted to the allocBlk (this takes O(1) time)

Total time = O(n) + O(1) + O(1)

= O(n)

Split

(2) Free (a given address):

→ Free operation first searches the allocated memory block list to check if there exists a contiguous memory block starting with the given addresses. This requires us to traverse the allocBlk list which is of maximum size 'n' (formed by 'n' allocate operations)

So, Time required = O(n)

→ If such block doesn't exist, nothing more to do. But if it does, the found memory block is removed from allocBlk and inserted into freeBlk (which takes O(1) time)

Total time = O(n) + O(1)

= O(n)