CS 1.2: Intro to Data Structures & Algorithms

Hash Table Time Complexity Worksheet

Name: Ayush Jain

Given: Linked List Solutions – implementation and time complexity

The variable ***n*** represents the number of items stored in the list (or equivalently, number of nodes).

Linked List operation	*short summary in pseudocode (English) of the major steps performed in the implementation*	*best case running time*	*worst case running time*
`is_empty`	check if `head` node exists (`None` or `not None`)	O(1)	O(1)
`length`	traverse all nodes; count 1 for each node	O(*n*)	O(*n*)
`append`	add new node to end (after `tail` node); update `tail` property to point to new node	O(1)	O(1)
`prepend`	add new node to beginning (before `head` node); update `head` property to point to new node	O(1)	O(1)
`find`	traverse all nodes until matching `data` is found; if found, return matching `data`; if not, return `None`	O(1)	O(*n*)
`delete`	traverse all nodes until matching `data` is found; if found, set previous node to point to `next` node	O(1)	O(*n*)

New: Hash Table Operations – implementation and time complexity

Use the variable ***n*** for the number of key-value entries stored and ***b*** for the number of buckets.

Hash Table operation	*short summary in pseudocode (English) of the major steps performed in the implementation*	*best case running time*	*average case running time*
`length`	run a loop through the buckets and find the length using a counter	O(n)	O(n)
`items`	go through all buckets and extend your list with each bucket's items	O(n)	O(n)
`contains`	check if the bucket contains the index using key and return true or false based on output.	O(1)	O(L) = O(n/b)
`get`	gets the bucket using the key and returns value if found	O(1)	O(L) = O(n/b)
`set`	gets the index and sets the new key value to the index	O(1)	O(L)
`delete`	deletes the key using the index	O(1)	O(L)