CMPT 225

Lecture 5 – Comparing List implementations - Tables

List ADT class invariant?

- Element duplication allowed?
- For sake of simplicity, we shall allow duplicated elements to be inserted in our List

Comparing various implementations of the position-oriented List ADT class using Big O notation

- Time efficiency of their operations (worst case scenario):

Operations	array-based (heap)	link-based (SHSL list)
getElementCount	**O(1)**	**O(1)**
insert	**O(n)** needs to shift right to make space for newElement	**O(n)** insert at the end (can be O(1) if use DHSL list)
remove	**O(n)** needs to shift left to overwrite (remove) the element to be removed	**O(n)** remove last element
removeAll/clear	**O(1)** if elements have not been stored on heap (e.g. `int`) – otherwise each element needs to be deleted -> **O(n)**	**O(n)** delete each Node object on heap and perhaps each element object on the heap
retrieve/get	**O(1)**	**O(n)** must traverse to reach last element

List ADT class invariant?

- Sorted – yes!
- Element duplication allowed?
- For sake of simplicity, we shall allow duplicated elements to be inserted in our List

Comparing various implementations of the value-oriented List ADT class using Big O notation

- Time efficiency of their operations (worst case scenario):

Operations	array-based (heap)	link-based (SHSL list)
getElementCount	**O(1)**	**O(1)**
insert	**O(n)** needs to shift right to make space for newElement and keep elements sorted	**O(n)***
remove	**O(n)** needs to shift left to overwrite (remove) the element to be removed and keep elements sorted	**O(n)***
removeAll/clear	**O(1)** if elements have not been stored on heap (e.g. `int`) – otherwise each element needs to be deleted -> **O(n)**	**O(n)**
retrieve/get	**O($\log_2$ n)**	**O(n)***

* There are several binary search algorithms (and programs) that successfully search a sorted linked list (SHSL list) for a target element (search key).
* They have different time efficiencies:
  * The binary search algorithm that performs in **$O(\log_2 n)$** has a space efficiency of **$O(n)$**, which is not very efficient. To achieve a time efficiency of **$O(\log_2 n)$**, it uses an array in which the memory address of each of the nodes are stored (hence **$O(n)$**).
    * ***Bottom Line:*** Since this implementation of Binary Search requires an array, perhaps, using an array, as opposed to a linked list, would be better: same time efficiency, but better space efficiency and easier implementation.
  * The binary search algorithm that performs in **$O(n \log_2 n)$** uses the slow/fast pointer algorithm to find the middle node/element. The first middle element is found in n/2 steps (**$O(n)$**), then n/4, then n/8, etc… There are **$O(\log_2 n)$** steps, hence **$O(n \log_2 n)$**
    * ***Bottom Line:*** Using linear search would be better: better time efficiency and easier implementation.