Dylan Shane

Comp220

Lab09 Write Up

Fall 2017

Time Efficiency

* + Name two methods in the IntList class that have the same big-O for both ArrayList and IntLinkedList. State the specific big-O of each method, and provide specific reference to your code to clarify why this is the correct efficiency.
    - isEmpty() | O(1)
      * This method is O(1) because it is only one line that returns the boolean resulting from the conditional (currItemCount == 0)
    - itemCount() | O(1)
      * This method is O(1) because it only has one line of code that returns the value stored in currItemCount
  + Name two methods in the IntList class where ArrayList has better big-O than IntLinkedList. State the specific Big-O of each method in each class (a total of 4 big-O notations), and provide specific reference to your code to clarify why this is the correct efficiency.
    - Find | **ArrayList** = O(logn) : IntLinkedList = O(n)
      * This is faster for ArrayList because it uses a binary search algorithm to find the item giving it a big O of O(logn), whereas IntLinkedList has to possibly check each node giving it a big O of O(n) since the entire list may have to be iterated through.
    - getValueAt | **ArrayList** = O(1) : IntLinkedList = O(n)
      * This is faster for ArrayList because we can directly access the index passed in so the method is O(1), whereas in IntLinkedList we have to iterate through the nodes possibly until the end of the list giving this method for IntLinkedList a big O of O(n)
  + Name two methods in the IntList class where IntLinkedList has better big-O than ArrayList. State the specific Big-O of each method in each class (a total of 4 big-O notations), and provide specific reference to your code to clarify why this is the correct efficiency.
    - insertAtFront | ArrayList = O(n2) : **IntLinkedList** = O(1)
      * This is faster for IntLinkedList because we only have to create the new node, set its next to the front, then change front to be pointing at our new node. Therefore, the big O is only O(1), whereas in ArrayList we have to move everything over and possibly double the arrays capacity. Moving everything over is O(n) since it must go through the array once, then if the size has to be doubled that is another O(n) method since the array has to be copied which gives the whole insertAtFront method O(n2) for ArrayLists.
    - insertAtEnd | ArrayList = O(n) : **IntLinkedList** = O(1)
      * This is faster for IntLinkedList because we only have to create the new node, set the current end’s next value to point at our new node, then change end to be pointing at our new node. Therefore, the big O is only O(1), whereas in ArrayList we have to possibly double the arrays capacity. Doubling the array’s capacity is an O(n) method since the array has to be copied with a doubled size therefore it must be fully iterated through making it an O(n) method. Thus, to insertAtEnd in an ArrayList is O(n).
  + Understanding the above time efficiency considerations, describe a real-world scenario where you would choose a LinkedList, and a real-world scenario where you would choose an ArrayList. (Be specific, not generic. Give context to clarify why one structure would be better than the other in each scenario, considering the above big-O for methods).
    - LinkedList would be more time efficient for a scenario like people waiting to receive customer support. This is because adding someone to the end of the list is an O(1) function while doing the same for an ArrayList is O(n). Furthermore, if in this scenario we were working in a FIFO basis it would also be O(1) to remove the first person using a LinkedList while an ArrayList would be O(n) since it has to shift all of the values after the removed item over one.
    - An ArrayList would be more useful if we were managing an inventory of books that could be searched by clients. This is because the find method for an ArrayList is O(logn) while it is O(n) for a LinkedList. This is under the assumption that there would be many searches for books to justify the sorting that is necessary for a binary find to be useful. Therefore, given there will be at least a few thousand searches in this hypothetical scenario, an ArrayList would be more time efficient than a LinkedList.
* Space Efficiency
  + Name a situation where an ArrayList is more space efficient than a LinkedList. Provide evidence to support your claim including a memory diagram of that specific situation.
    - ArrayList is more space efficient when the size of the list is fixed to a certain value. This is because an ArrayList is one singular object with an array of that size(n) that you don’t have to worry about doubling since it is a fixed size. A LinkedList would be have n objects on the heap which would take up more space than the array of size n.
  + Name a situation where a LinkedList is more space efficient than an ArrayList. Provide evidence to support your claim including a memory diagram of that specific situation.
    - LinkedList would be more space efficient in a situation where we don’t how large the list is going to be. This is because in an ArrayList we would have to double the size which leaves a lot of unused space for only adding a single item. A node can be added to a LinkedList easily without having to add extra unnecessary space.
  + Understanding the above space efficiency considerations, describe a real-world scenario where you would choose a LinkedList, and a real-world scenario where you would choose an ArrayList.(Be specific, not generic. Give context to clarify why one structure would be better than the other in each scenario considering the above space considerations).
    - A LinkedList would be more space efficient in a situation like a order tracker for a fast food drive thru. In this case, we don’t know how many people are going to order on any given night so it would be more space efficient than an ArrayList. This is because an ArrayList would have to double its capacity when it hit a certain threshold of orders, and the restaurant may only get a few more orders leaving unnecessarily allocated memory. Furthermore, to hold the information of each order this would have to be an array of order objects which would then heavily skew space efficiency towards LinkedList.
    - An ArrayList is more space efficient in a scenario like keeping track of each player on a basketball team’s points scored. This is because we have a fixed number of values in our array--the roster size. Therefore, we won’t have any unused allocated memory. This is more space efficient than a LinkedList because this would need the roster size-amount of nodes. Each node would take up its own block of memory on the heap which would take up more space than each of the integer blocks of memory in an ArrayList.