Haohua Feng

50288502

WA2

Problem 1.

1(a). The best-case execution for insert occurs when numSorted is less than capacity because it just needs to insert a new entry directly without remove.

$T(n) = O(1)$

1(b). A scenario that causes the worst-case execution for insert, insert method is called when capacity reach its max. It needs to remove the oldest entry which at index 0, and move everything in the array 1 index left / forward. So that the second oldest becomes the oldest, free the highest index for the insertion of the new entry.

$T(n) = \Theta(n) + O(1)$

1(c). Base on the implementation of remove provided in hand out, it uses a for loop to match the entry needs to remove from the highest index to index 0. The best-case execution for remove happens when the entry that needs to remove is not in the list. In this scenario, remove only go though the whole list by the for loop, and doesn't remove anything since the condition statement of dataArray(i) == entry can never be true.

$T(n) = O(n) * O(1)$

1(d). The worst-case execution for remove, the entry is going to be removed is at index 0. After remove the oldest entry, every entry from index 1 to the highest index need to move 1 index left.

$T(n) = O(n) * O(1) + \Theta(n)$

Problem 2.

2(a).

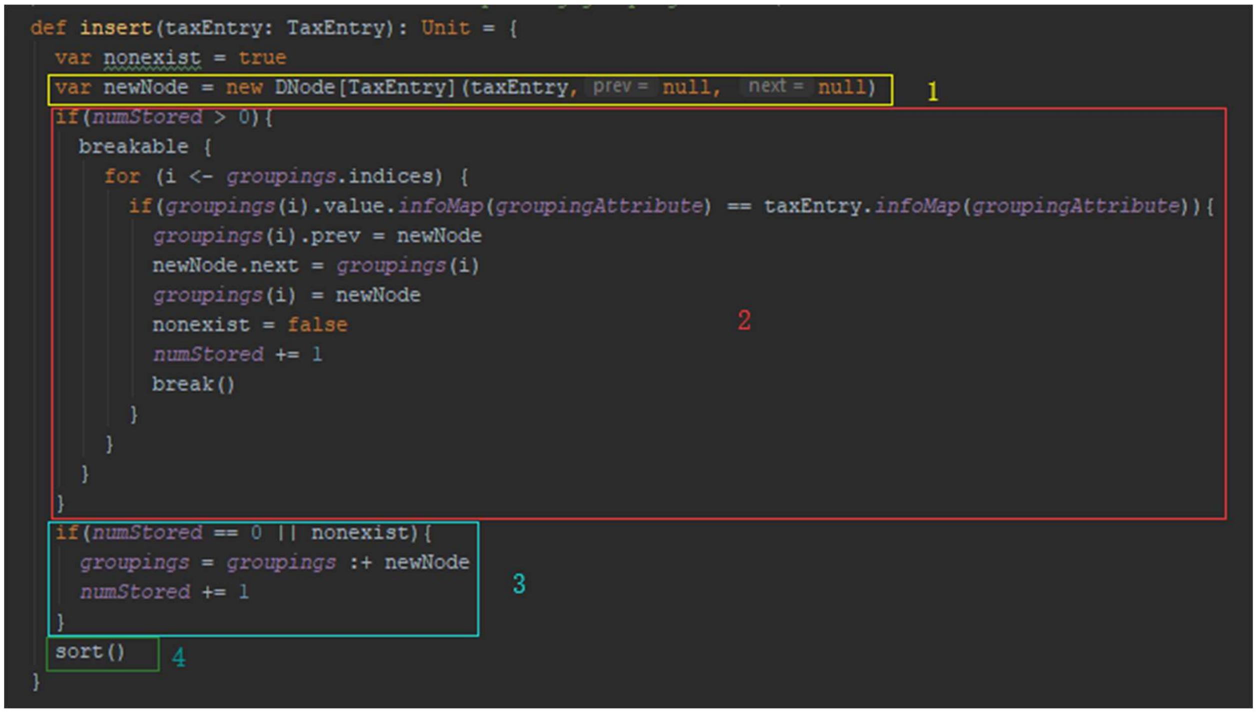
Insert:

1. Create a new DNode, set everything in side is null, and the value equal to the input.
2. If the array is non-empty, check is there already having the same goupingAttribute inside the list. If true, make linked connection between the old head and the new node. And then, replace the old head by new node at that index and update number of stored elements.
3. Else if the array is empty or there is no such goupingAttribute. Append that new node into array and update numStored.
4. Call the helper function 'sort' to keep the array organized.

Regroup:

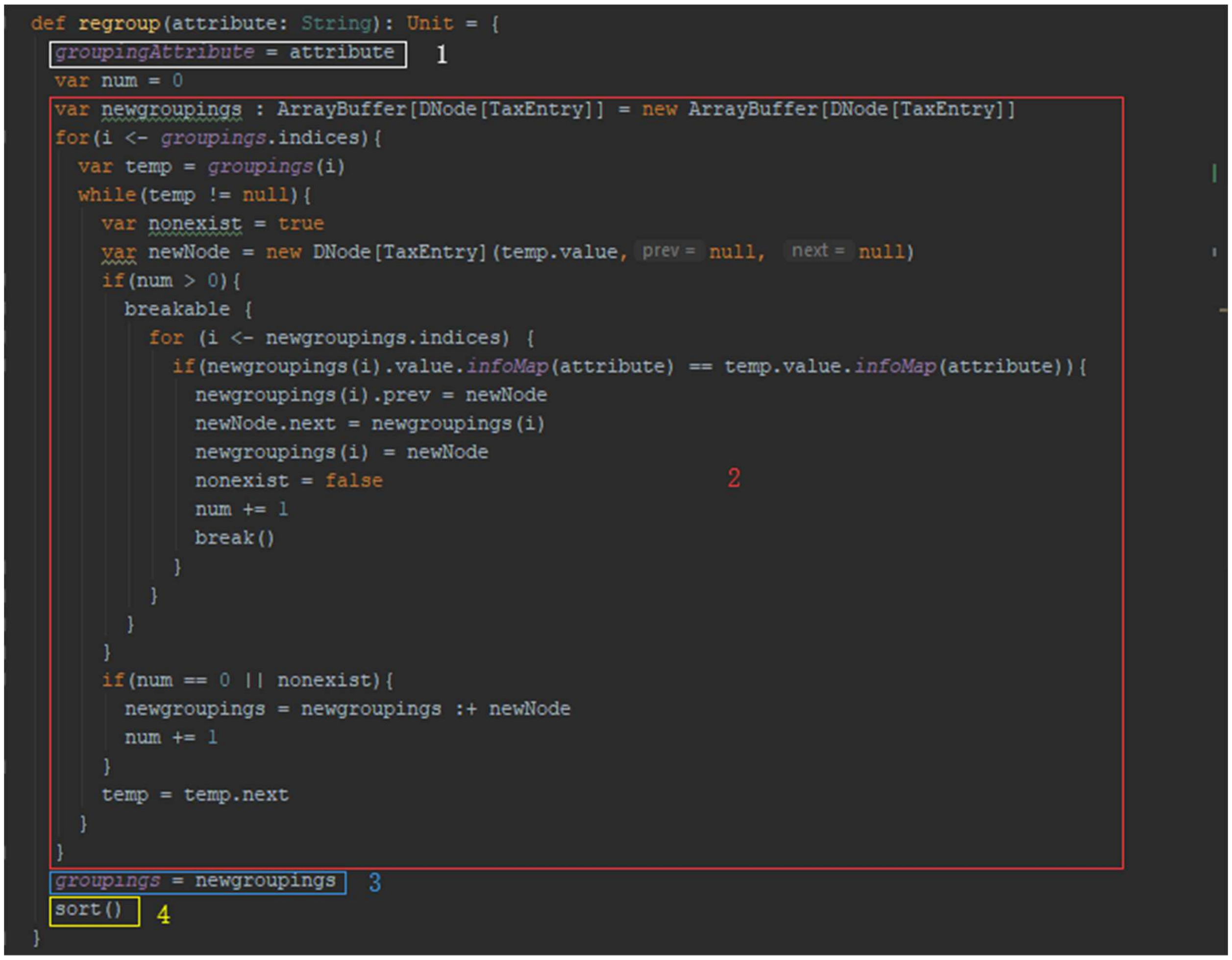
1. Update groupingAttribute.
2. Create a new arraybuffer to regroup inserted elements by inserting them one by one from arraybuffer goupings. The process of insertion is similar to method insert which describe above.
3. Set grouping equal to this new arraybuffer.
4. Call the helper function 'sort' to keep the array organized.

2(b). Insert:



```scala
def insert(taxEntry: TaxEntry): Unit = {
  var nonexist = true
  var newNode = new DNode[TaxEntry](taxEntry, prev = null, next = null)   1
  if(numStored > 0){
    breakable {
      for (i <- groupings.indices) {
        if(groupings(i).value.infoMap(groupingAttribute) == taxEntry.infoMap(groupingAttribute)){
          groupings(i).prev = newNode
          newNode.next = groupings(i)
          groupings(i) = newNode
          nonexist = false                                    2
          numStored += 1
          break()
        }
      }
    }
  }
  if(numStored == 0 || nonexist){
    groupings = groupings :+ newNode
    numStored += 1                                            3
  }
  sort()   4
}
```

Regroup:



```
def regroup(attribute: String): Unit = {
  groupingAttribute = attribute
  var num = 0
  var newgroupings : ArrayBuffer[DNode[TaxEntry]] = new ArrayBuffer[DNode[TaxEntry]]
  for(i <- groupings.indices){
    var temp = groupings(i)
    while(temp != null){
      var nonexist = true
      var newNode = new DNode[TaxEntry](temp.value, prev = null, next = null)
      if(num > 0){
        breakable {
          for (i <- newgroupings.indices) {
            if(newgroupings(i).value.infoMap(attribute) == temp.value.infoMap(attribute)){
              newgroupings(i).prev = newNode
              newNode.next = newgroupings(i)
              newgroupings(i) = newNode
              nonexist = false
              num += 1
              break()
            }
          }
        }
      }
      if(num == 0 || nonexist){
        newgroupings = newgroupings :+ newNode
        num += 1
      }
      temp = temp.next
    }
  }
  groupings = newgroupings
  sort()
}
```

2(c).

Insert:

If array is empty: T(n) = O(1) + O(n^2)

Non-empty but no such groupingAttribute: T(n) = O(1) + O(n^2)*O(1) + O(1) + O(n^2)

Non-empty and groupingAttribute exists: T(n) = O(1) + O(n^2)*O(1) + O(n^2)

Regroup:

T(n) = Θ(1) + O(n^2)*O(1) + O(n^2)

Problem 3.

If TaxEntry objects were stored in arraybuffer objects instead of in a linked list, insertion or deletion will not be that convenience, O(n). But random access can be done on arraybuffer, O(1). For insertion, we can simply use prepend or append to insert new entries instead of update the head nodes. Two iterator methods can be simplified because there are built in functions. In this case, this will perform slower in method insert, increase the run time.