Ordered List ADT

Fixed-Size Array
Implementation Version 2

Implementation Option 2

- add() method simply inserts into next open slot at end of the array.

- Which Ordered List ADT methods must be changed? How are they different?

Implementation Option 2

- add() method simply inserts into next open slot at end of the array.
- print() method must sort list before printing.
- retrieve() / delete methods call on sequential search private helper method.

FATicketOrdredList

```
public class FATicketOrderedList implements
TicketOrderedList
{
  /* VERSION 2 OF ADD METHOD – INSERT INTO NEXT OPEN SLOT
   * @param T Given Ticket initialized object
   * @return true if not full and T's key is unique; else,
   * false leaving list unchanged. */
  public boolean add(Ticket T)
  {
  }
}
```

FATicketOrdredList

```
public class FATicketOrderedList implements
TicketOrderedList
{
   /*Inserts Ticket into the list if its key is unique and the list is not full.
     * Returns true if successful; otherwise, false */
   public boolean add(Ticket C)
   {
     // Do nothing if full or duplicate key.
     if(isFull() || retrieve(C.getNum() != null)
       return false;
     myList[myCount++] = C;
     return true;
   }
}
```

FATicketOrdredList

```
public class FATicketOrderedList implements
TicketOrderedList
{
   /*Returns array index of Ticket having key value;
     * otherwise, return -1 if not found.*/
   private int sequentialSearch(int key)
   {


   }
}
```

FATicketOrdredList

```
public class FATicketOrderedList implements
TicketOrderedList
{
   /*Returns array index of Ticket having key value;
     * otherwise, return -1 if not found.*/
   private int sequentialSearch(int key)
   {
       int foundIndex = -1;
       int x = 0;
       while(found == null && x < myCount)
       {
          if(myList[x].getNum() == key)
             foundIndex = x;
          else x++;
       }
       return foundIndex;
  }
}
```

FATicketOrdredList

```
public class FATicketOrderedList implements
TicketOrderedList
{
   /*Returns the object having the specified key value;
     * otherwise, return null if not found.*/
   public Ticket retrieve(int key)
   {
   }
}
```

FATicketOrdredList

```
public class FATicketOrderedList implements
TicketOrderedList
{
  /*VERSION 2 – CALL ON BUBBLE SORT, THEN PRINT */
  public void print()
  {
    bubbleSort(myList);

    // Loop to print list of Tickets, now in sorted order.
  }
}
```

Bubble Sort

```
bubbleSort(int[] data)
{
  for(int i = 0; i < data.length-1; i++)
  {
    for(int j = data.length-1; j > i; j--)
    {
      Swap elements in array slots
      and j and j-1 if they are out of order.
    }
  }
}
```

Bubble Sort

```
public static void bubbleSort(int[] data)
{
  for(int i = 0; i < data.length-1; i++)
    for(int j = data.length-1; j > i; j--)
       if(data[j-1] > data[j])
          swap(data, j-1, j)
}

/* Private helper method swaps array elements at indices
 * index1 and index2 */
private static void swap(int[] data, int index1,
                         int index2)
{
  // Please complete this method.




}
```

Modify for Use with Ticket Objects

```
private static void bubbleSort(Ticket[] data)
{
  for(int i = 0; i < data.length-1; i++)
    for(int j = data.length-1; j > i; j--)
       if(data[j-1] > data[j])
          swap(data, j-1, j)
}

/* Private helper method swaps array elements at indices
 * index1 and index2 */
private static void swap(Ticket[] data, int index1,
                         int index2)
{
  // Please complete this method.









}
```

Bubble Sort Complexity

- Arrange your letter tiles to try to find a best case input for which bubble sort does less work in terms of number of swaps.

- Arrange your letter tiles to try to find a worst case input for which bubble sort does most work in terms of number of swaps.

Bubble Sort Complexity

- Arrange your letter tiles to try to find a best case input for which bubble sort does less work in terms of number of key compares.
  - A[x] > B[y] counts as one key compare

- Arrange your letter tiles to try to find a worst case input for which bubble sort does most work in terms of number of key compares.

Bubble Sort Complexity

- Arrange your letter tiles to try to find a best case input for which bubble sort does less work in terms of number of swaps.

- Arrange your letter tiles to try to find a worst case input for which bubble sort does most work in terms of number of swaps.

Bubble Sort Complexity

- Arrange your letter tiles to try to find a best case input for which bubble sort does less work in terms of number of key compares.
  - A[x] > B[y] counts as one key compare

- Arrange your letter tiles to try to find a worst case input for which bubble sort does most work in terms of number of key compares.