CSE12 Fall 16 Practice Midterm Solutions

1. True False

a. False. An Abstract Data Type, or ADT, consists of (a) a specification of the possible values of the data type and (b) a specification of the operations that can be performed on those values in terms of the operations' inputs, outputs, and effects.
b. True
c. False
d. True
e. True
f. False. It is true as n→ infinity, but it is not ALWAYS true. $O(n^2)$ might be faster for smaller values of n
g. True
h. True
i. True
j. True
k. True

2. **Short Answer**

a) There are many different order of assignments. Below is one possible answer:
B.next = C.next
C.prev = B.prev
B.prev.next = C
C.next.prev = B
B.prev = C
C.next = B
TIP: Count how many links need to be updated. Also don't mess up the order!

b) $O(n^2)$ (Give correct explanation with instruction counts)

c) S1: yellow, black, green, red, blue
S2: orange
S3: empty

d) Map both push() and pop() to the end of the array i.e top of the Stack is at index size-1. Explain why this is O(1).

e) Advantages of LinkedLists:

i) Size of an array is fixed, but the size of a LinkedList can dynamically increase as more elements are added, with better efficiency.
ii) Random insertions are costly in an array because room has to be created for the new element to be inserted and to create room the existing elements have to shifted
iii) Similarly, random deletion is also expensive with arrays

Disadvantages

iv) Random access is not allowed in a Linked List. We have to access elements sequentially starting from the first node. So we cannot do a binary search with linked lists as we do with arrays.
v) Extra memory space for a reference to the next (and previous) node is required with each element of the list.

f) iii)

3. **Coding question**

a)

```
public E next( ) {
      if ( nelems==0 ) { // Check if empty.
            throw new NoSuchElementException();
        }
        E nextElement = right.getElement(); // Gets next element.
        left = right; // Moves the iterator.
        right = left.getNext();
         forward = true;
        index = (index + 1)%nelems;
         return nextElement;
}
```

IMPORTANT: Do not forget to use the % operator to increment index. This is essential for making the iterator wrap-around to the beginning of the list and hence make it circular. A simple index = index++ will give you a linear list that goes out of bounds.

b)

```
public int previousIndex() {

      if(index==0)
            return nelems-1;
      return index-1;
```

```
}
```

c)

```
public void removeNNodes(int n){
          if (n > nelems)
               throw new IndexOutOfBoundsException;  //optional
          if (n==nelems) {
              head = tail = null;
              left = right = null;
              index = 0;
              nelems = 0;
          } else {
              for (int i = 0; i < n; i++){
                  left.getPrev().setNext(right);
                  right.setPrev(left.getPrev());
                  left = left.getPrev();
                  index--;
                  if(index<0)
                      index = nelems-1;
                  nelems --;
              }
          }
      }
```

Notes:

1. The iterator moves forward and so you must always remove the left node.
2. Make sure you check what is available to the Iterator class. You do not have access to the Node's remove method according to the question and so you cannot use it in your solution.