SPRING 2019 CS 445 MIDTERM EXAM WED FEB 27 (Instructor Tim Hoffman)

A PITT USER NAME IS AN EMAIL ADDY w/out @PITT.EDU. LIKE THIS: **JFK63**

PRINT –**YOUR**- PITT USERNAME ➔ ________________

PRINT FIRST & LAST NAME LEGIBLY ➔ _____________________________

There are 22 questions on this exam. 1 – 20 worth 5pts each
Questions 21 & 22 are Extra Credit 1 point each.

YOU MUST HAND IN YOUR ANSWERS ONLINE BY GOING TO THE HANDIN SITE AND UPLOADING TO THE MIDTERM LINK. THE HANDIN WILL ONLY ACCEPT A FILE NAMED Midterm.java

AFTER HANDING IN ONLINE YOU SHOULD GET FEEDBACK JUST LIKE ANY OTHER ASSIGNMENT. ALSO YOU MUST SEE YOUR NAME IN GREEN ON THE PROJECTOR. IF YOUR NAME IS IN RED YOUR ANSWER FILE HAS MISSING ANSWERS OR INCORRECT FORMAT.

AS SOON AS YOU SEE YOUR NAME IN GREEN ON THE PROJECTOR YOU MUST HAND ME YOUR PAPER EXAM. THEN YOU MAY LEAVE THE ROOM.

IF YOU DON'T HAND ME YOUR SIGNED PAPER EXAM BEFORE LEAVING THE ROOM, YOU WILL GET A 0 FOR THE MIDTERM

Section #1   ARRAY BASED BAG ADT

Assume the following ArrBag objects of type T where type T is actually String type and assume those ArrBags contain the following elements respectively.

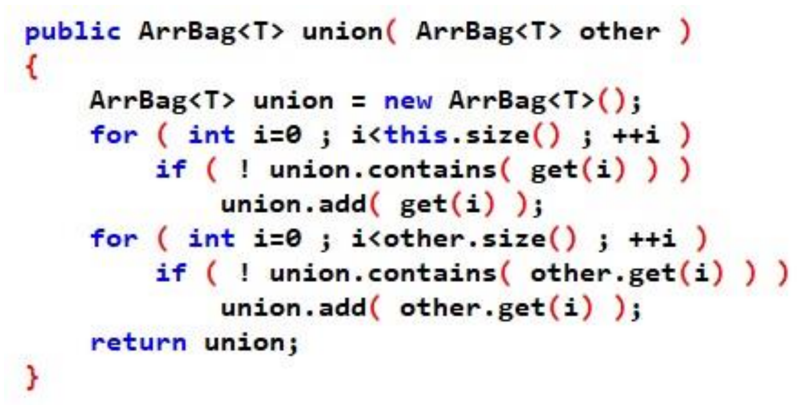
This ArrBag class is the one you wrote for your project.

ArrBag A --> [bravo][delta][alpha][golf][india][hotel][echo] // elements not in sorted order

ArrBag B --> [foxtrot][baker][xray][zebra][charlie][victor]   // elements not in sorted order

You are to assume that size() is O(1). It just returns the count
You are to assume the elements on the backing array are not necessarily sorted.
Assume that .size() .get() .contains() .add() methods are correctly written.

Look at this union method for the ArrBag class:



```
public ArrBag<T> union( ArrBag<T> other )
{
    ArrBag<T> union = new ArrBag<T>();
    for ( int i=0 ; i<this.size() ; ++i )
        if ( ! union.contains( get(i) ) )
            union.add( get(i) );
    for ( int i=0 ; i<other.size() ; ++i )
        if ( ! union.contains( other.get(i) ) )
            union.add( other.get(i) );
    return union;
}
```

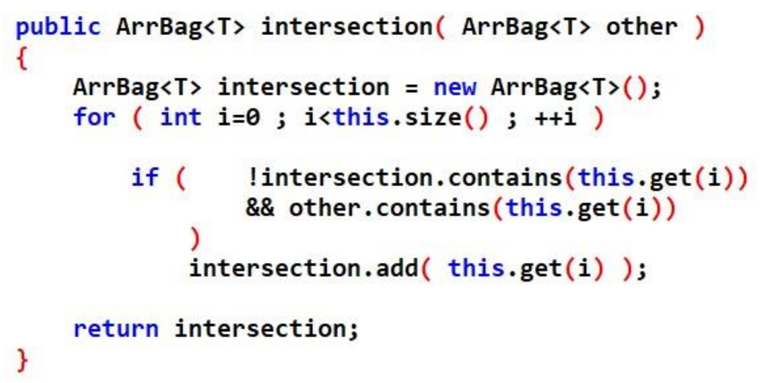
#1	Is the union code **above** completely correct? i.e. does it produce the correct union of these 2 sets with no dupes introduced into the result set?

A	YES, it correctly produces the union of those 2 sets with no dupes
B	It works on these two sets **but** if either set had dupes, it could put a dupe into the result
C	NO, it misses the last element of both sets
D	YES but inefficient. Should add all elements from both then remove dupes from result

#2	What is the” big Oh” runtime of the **above** union code?

A	O(1)
B	O(log2N)
C	O(N*log2N)
D	O(N*N)
E	O(2 to the N)
F	O(N!)

Look at this intersection method for the ArrBag class:



```
public ArrBag<T> intersection( ArrBag<T> other )
{
    ArrBag<T> intersection = new ArrBag<T>();
    for ( int i=0 ; i<this.size() ; ++i )

        if (    !intersection.contains(this.get(i))
                && other.contains(this.get(i))
           )
           intersection.add( this.get(i) );

    return intersection;
}
```

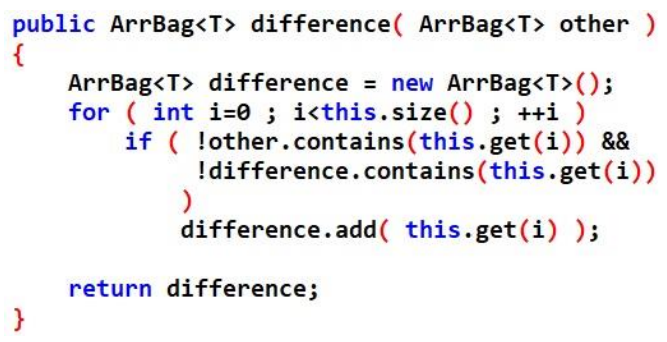
#3	Is the intersection code as written completely correct? i.e. does it produce the correct intersection of these 2 sets with no dupes introduced into the result set?

A	YES, it correctly produces the intersection of those 2 sets with no dupes
B	It works on these two sets **but** if either set had dupes, it could put a dupe into the result
C	NO, it makes the wrong if test. Should be: ***`&& !other.contains( this.get(i) )`***
D	YES but inefficient. Should add all elements from both then remove dupes from result

#4	What is the" big Oh" runtime of the **above** intersection code?

A	O(1)
B	O(log2N)
C	O(N*log2N)
D	O(N*N)
E	O(2 to the N)

Look at this difference method for the ArrBag class:



```
public ArrBag<T> difference( ArrBag<T> other )
{
    ArrBag<T> difference = new ArrBag<T>();
    for ( int i=0 ; i<this.size() ; ++i )
        if ( !other.contains(this.get(i)) &&
             !difference.contains(this.get(i))
           )
           difference.add( this.get(i) );

    return difference;
}
```

#5	Is the difference code **above** completely correct? i.e. does it produce the correct difference of these 2 sets with no dupes introduced into the result set?

A	YES, it correctly produces the difference of those 2 sets with no dupes
B	It works on these two sets **but if** either set had dupes, it could put a dupe into the result
C	NO, it misses the last element or both sets
D	YES but inefficient. Should add all elements from both then remove dupes from result

#6 What is the" big Oh" runtime of the **above** difference code?

A O(1)
B O(log2N)
C O(N*log2N)
D O(N*N)
E O(2 to the N)
F O(N!)

Section #2 ONE WAY LINKED LIST ADT

Assume the following LinkedList objects of type T where type T is actually String type and assume those Linked Lists contain the following elements respectively.

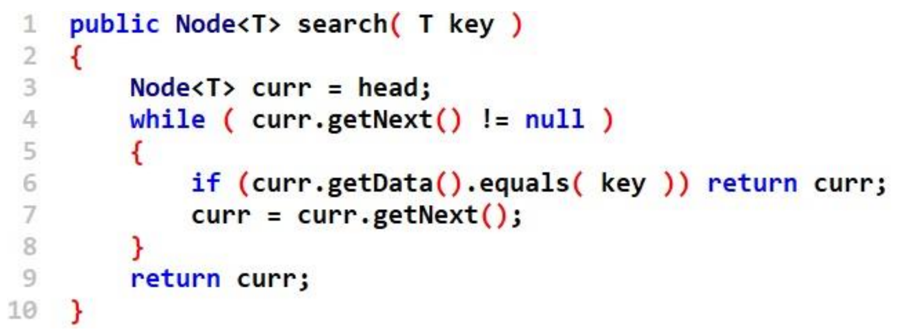
This LinkedList class is the one you wrote for your project.

LinkedList A --> bravo -> delta -> alpha -> india -> hotel -> echo // again elements not in sorted order

LinkedList B --> foxtrot -> baker -> xray -> zebra -> Charlie -> victor // not in sorted order

You are to assume that size() is O(1). It just returns the count
You are to assume the elements in the list are not necessarily ordered.
Assume that .size() .get() .contains() .add() methods are correctly written.

Look at this search method for the LinkedList class:
Assume incoming key is NOT null



```
public Node<T> search( T key )
{
    Node<T> curr = head;
    while ( curr.getNext() != null )
    {
        if (curr.getData().equals( key )) return curr;
        curr = curr.getNext();
    }
    return curr;
}
```

#7 Is the search code **above** completely correct? i.e. does it always return the reference to the node containing the specified key (or null if not found) in all cases?

A YES, it correctly returns the reference the node continuing key (or null if not in List)
B NO, it fails on head == null
C NO, it misses the last element
D YES but inefficient. Should add all elements from both then remove dupes from result

Look at this pair of methods you could have written for your linked list project. Assume incoming key is NOT null

```
// PUBLIC SEARCH THAT MAIN CALLS
public Node<T> search( T key )
{
    return searchHelper( key, this.head );
}

// PRIVATE HELPER METHOD DOING THE REAL WORK
private Node<T> searchHelper( T key, Node<T> head )
{
    if ( head==null || head.getData().equals(key) ) return head;
    return searchHelper( key, head.getNext() );
}
```

#8 Does the **above** pair of methods (working together) correctly return the reference to the node that contains key (or null if not in list) ?

A YES, completely correct and does not fail on any edge cases
B NO, it is close but fails on an edge case.
C NO, it has a flaw such that it only gets it right on the base case
D NO, it has a flaw such that it never gets any cases right .

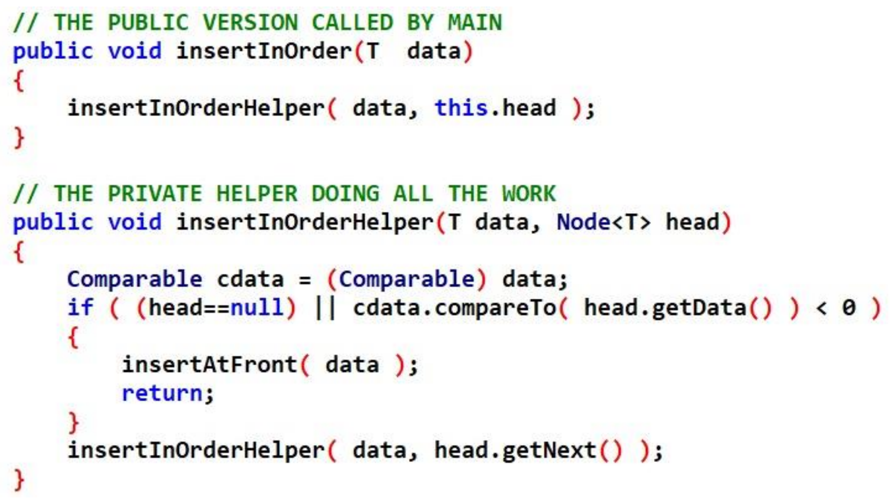
#9 Assuming the **above** search is called on a list that has N elements in it, what is the maximum number of clones of the private searchHelper() method that will ever be on the call stack at any time. ***Be clear – I am asking for maximum number of clones of the private helper on stack at any time. Do not count the public search().***

A N
B N-1
C N+1
D log2N + n
E (n squared + n) / 2

#10 Why didn't we just make that searchHelper() method public, and then call it from main?

A You can't call recursive methods from a main method
B It creates a risk of stack overflow
C We could have.
D none of the above

Look at this second pair of methods you could have written for your linked list project. Assume incoming key is NOT null



```
// THE PUBLIC VERSION CALLED BY MAIN
public void insertInOrder(T  data)
{
    insertInOrderHelper( data, this.head );
}

// THE PRIVATE HELPER DOING ALL THE WORK
public void insertInOrderHelper(T data, Node<T> head)
{
    Comparable cdata = (Comparable) data;
    if ( (head==null) || cdata.compareTo( head.getData() ) < 0 )
    {
        insertAtFront( data );
        return;
    }
    insertInOrderHelper( data, head.getNext() );
}
```

#11 Does the **above** pair of methods (working together) correctly insert the key into the list in correct sorted order position??

A YES, completely correct and does not fail on any edge cases
B NO, it is close but fails on certain input orderings
C NO, it has a flaw such that it only gets it right on the base case
D It insert the elements in reverse/descending order instead of ascending

END OF SECTION TWO

#12 Why is insert at tail easier on a CDLL circular double linked list than on a non-circular singly linked list?

A Because you can remove a node in O(1) time
B Tail node is always at head.getPrev()
C There are no base cases
D There are no nulls in a CDLL

#13 If I have an operation X that performs a logarithmic operation and then a linear operation, then operation X is said to be logarithmic.

A TRUE
B FALSE

#14 In an array based ADT, the remove at front operation is faster than a Linked List based remove at front.

A TRUE
B FALSE

#15 What is the advantage of a Linked List over an Array?

A Linked List does not require large chunks of contiguous memory
B Linked List only uses more memory as more elements are added
C Linked List is faster for sorting and searching
D A and B
E A and B and C

#16 Why (the strongest reason) is removal of a node easier for a CDLL circular doubly linked list than for a singly linked (one direction) linked list?

A On a CDLL You don't have to stop before the node you want to delete
B On a CDLL you only have to reassign one single pointer to remove the node
C On a CDLL you can search for it in either direction from the head
D There are no nulls in a CDLL

#17 Which of these operations are O(1)?

A indexing into an array
B comparing primitives
C copying a primitive to another primitive
D A, B & C
E just A and B

#18 Which of these operations are O(N)

A remove at front on an array
B remove at front on a singly linked list
C insert at front on an array
D insert at front on a singly linked list
E A and C
F B and D

#19 The complexity of binary searching a sorted CDLL is the same as on a sorted array

A TRUE
B FALSE

#20 Why does the java language offer us Generics (the <T> stuff) ?

A so that we may write the definition of a container once instead of having to write separate version of it for each data type we might want to store in it
B so that it can enforce compliance on code (programs) that want to use the container.
C It is a form of code re-use – one the fundamental principles of software engineering
D all of the above

#21 OPEN RESPONSE Why do we need a heuristic to solve extremely large boggle boards?

#22 OPEN RESPONSE In our ArrBag project, when we upSized our arrays, we created a new array that was twice the length of the one that got full. Why didn't we just increase it by one - and avoid having an array that has a lot of wasted extra unused capacity when we reach end of file?