Name ……

ID: ……

COMP 103

Mid Trimester Test

17 August 2006

Model Answers

Time: 90 minutes

Marks: 90

Answer all questions.

Non programmable calculators permitted

Unannotated foreign language dictionaries permitted

1. Collection Types	16	
2. ArrayList	12	
3. Sorting	30	
4. Linked Lists	17	
5. Using Collections	15	
	90	

Question 1. Collection Types. [16 Marks]

(a) [4 marks] Which collection type would be a good choice for representing the collection of people waiting to be served at a supermarket checkout? Explain why.

***Queue** of Person. People at a supermarket checkout follow the queue discipline of adding themselves at the back, and removing themselves from the front.*

Real life is always a little more complicated, so that sometimes people waiting in the middle of the queue leave it because they see another queue that's shorter, or they remember some other item they have forgotten. To model this behaviour, you would need to be able to remove an item from the middle of the queue. Also, sometimes, people with just a few items are "let in" at the front of the queue. To handle this well, you would need a List, rather than a Queue. A standard ArrayList is inefficient for adding or removing at the front, but supermarket checkout queues are sufficiently small that we probably don't care.

(b) [4 marks] Which collection type would be a good choice for storing information about which employees are currently inside a building? Explain why.

***Set** of Person or Set of Employee. You can't have duplicates of a person – there is only one of each of them, and just recording who is in the building and who is not involves no structure. Therefore, a Set is appropriate.*

If you wanted to represent where people are in the building, then you might use a Map from Person to Room, where the null room represents a person not in the building at all. Alternatively, you might have a Set of Person, but include the location of a person as a field in the Person object . If you also wanted to represent which people were in each room, you might have a second Map from Room to Set of Person. If you track the exact location of all the employees all them, then you might want a Quad-tree for indexing the people in each region of the building by their coordinates, which is an efficient representation of a Map from continuous 2D coordinates to Sets of Objects.

(c) [4 marks] Which collection type would be a good choice for storing the information needed to retrieve automobile ownership records based on number plate? Explain why.

***Map** from String to OwnershipRecord. The number plates can be represented by strings, and are unique. You want to access the OwnershipRecord given the number plate string, so you need a Map.*

(d) [4 marks] Name a collection type that has constrained access and state the ways in which the access is constrained.

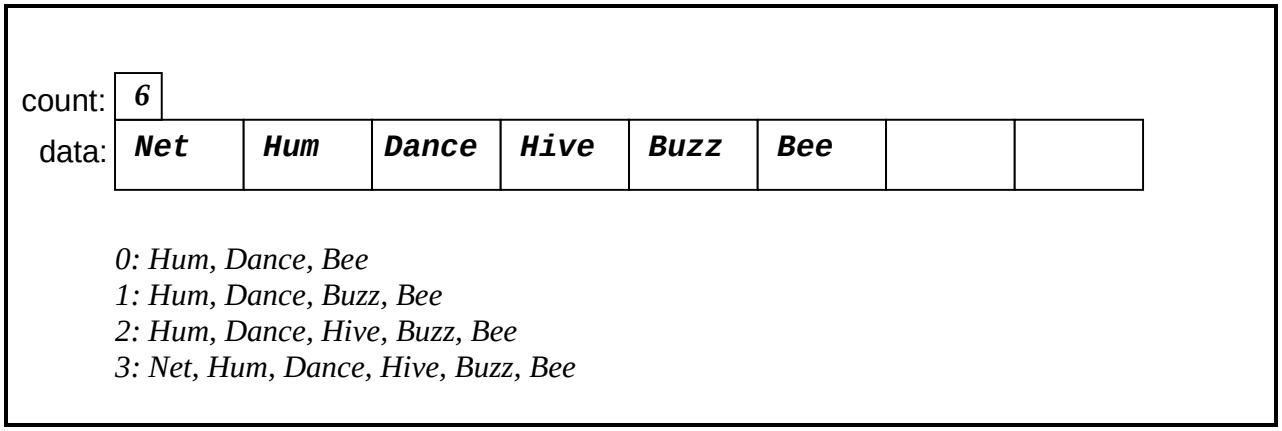
***Stack**: items can only be added or deleted from one end (the top of the stack).*
***Queue**: items can only be added at one end (the back) and can only be removed from the other end (the front)*
***Deque**: items can only be added to or deleted from an end (either end), not the middle.*
***Priority Queue**: Only the highest priority item can be accessed or removed. Items added will be placed according to their priority.*

ID: ……..

Question 2 ArrayList [12 Marks]

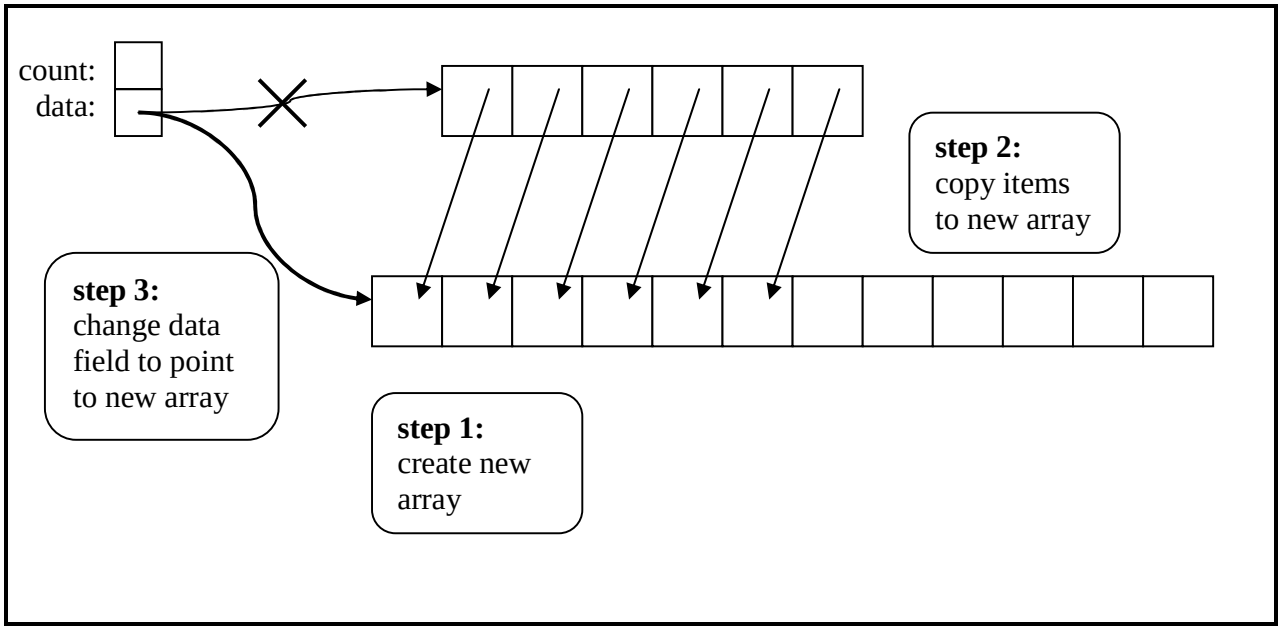
(a) [3 Marks] Draw a diagram showing the result of inserting the following three items into an ArrayList that initially contains [“Hum”, “Dance”, “Bee”]

insert “Buzz” at index 2,
insert “Hive” at index 2,
insert “Net” at index 0.



0: Hum, Dance, Bee
1: Hum, Dance, Buzz, Bee
2: Hum, Dance, Hive, Buzz, Bee
3: Net, Hum, Dance, Hive, Buzz, Bee

(b) [4 Marks] Draw a diagram showing how ensureCapacity increases the capacity of an ArrayList when the array becomes full. Label the steps on your diagram to make the sequence clear.



(c) [5 Marks] Consider the following clear method for the ArrayList class that removes all the items in an ArrayList. It uses the remove method, which removes the item at a given index). Assume that the count field stores the number of items in the ArrayList.

```
public void clear(){
    int max = count;
    for (int j = 0;  j < max;  j++)
        remove(0);
}
```

What is the cost of this method, expressed in terms of *n* = the initial size of the list? Justify your answer.

O(n^2) *The loop will be performed n times. Each time, it removes the first item in the list, and therefore has to move all the other items down one. The cost of this is the size of the list -1, which starts at n-1 and goes down to 0.*
Therefore, the cost is

(n-1) + (n-2) + (n-3) … + 2 + 1 + 0 = n(n-1)/2 = O(n^2)

ID: ……..

Question 3 Sorting [30 Marks]

(a) [8 marks] For each of the following sort algorithms, state its average case cost and worst case cost (using big-O notation), whether it is stable, and whether it is in-place.

	Average case	Worst case	Stable?	In-place?
Insertion Sort:	$O(n^2)$	$O(n^2)$	*yes*	*yes*
MergeSort:	$O(n \log(n))$	$O(n \log(n))$	*yes*	no
QuickSort:	$O(n \log(n))$	$O(n^2)$	*no*	*yes*
Radix Sort:	$O(n)$	$O(n^2)$ or $O(n \log(n))$	*yes*	*no*

(b) [4 marks] Show how Bubble Sort sorts the following list of six items by showing the state of the list after each swap.

[A , B , D , F , E , C]

[A , B , D , E , F , C] *pass 1: swap F and E*

[A , B , D , E , C , F] *pass 1: swap F and C*

[A , B , D , C , E , F] *pass 2: swap E and C*

[A , B , C , D , E , F] *pass 3: swap D and C*

passes 4 and 5: no swap

(c) [7 marks] Show how Quicksort sorts the following list of 9 items by showing the state of the list after each call to partition

You may use the partition algorithm described in the lectures. You may also use another standard partition algorithm, but you must then describe it very briefly.

[J , H , G , V , U , K , L , H , U]

0..8: partition on J => swap H-H, G-G, H_2-V ; finally J-H_2 return 3 => recurse 0..2 & 4..8

[H , H , G , J , U , K , L , V , U]

0..2: partition on H => swap H-G ; finally H –G return 1 => recurse 0..0 & 2..2

[G , H , H | J , U , K , L , V , U]

0..0 no action,
2..2 no action
4..8 partition on U => swap K-K, L-L finally U-L return 6 => recurse 4..5 & 7..8

[G , H , H , J | L , K , U , V , U]

4..5: direct swap (doesn't need a call to partition)

[G , H , H , J | K , L | U , V , U]

7..8: direct swap (doesn't need a call to partition)

[G , H , H , J , K , L , U | U , V]

(d) [7 marks] Outline the Insertion sort algorithm using pseudo code.

```
To insertion sort a list from positions 0 to n-1:
    For each index  i  in list, from 1  to  n-1
        copy item at i  to temp
        for each index  j  from i down to 1,
            if item at j-1 is larger than temp,
                move item at j- 1 to j
            otherwise
                break
        copy temp to position j
```

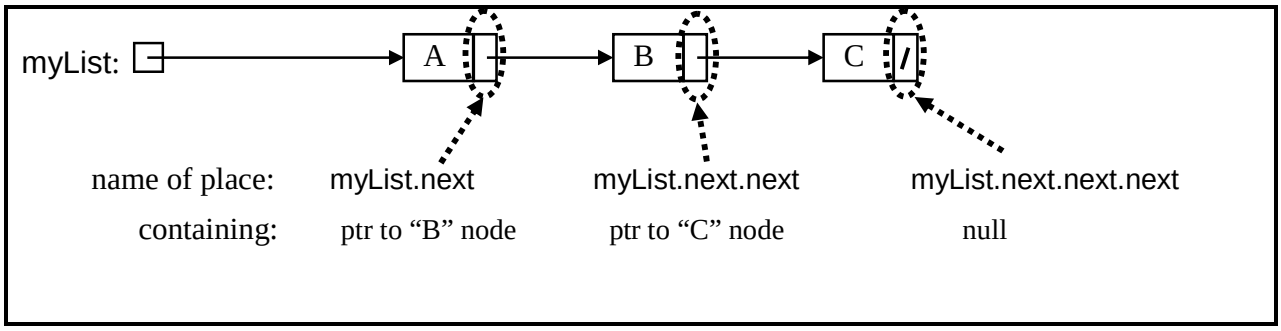
(e) [4 marks] Explain why InsertionSort is more efficient than QuickSort or MergeSort if you know that the list is almost sorted.

If a list is almost sorted, then almost every element is very close to its correct position, and only a very small number of items are far from their correct position. Insertion sort only makes enough compare-and-moves to move each element from its original position to its correct position. If the list is almost sorted, then this will be small. In particular, if at most k items are far from their correct place and the other items are at most k steps from their correct place, then the cost will be O(kn).

Merge sort and quicksort must always do O(n log(n)) moves and/or comparisons, even if the list is already in order, so insertion sort will be faster.

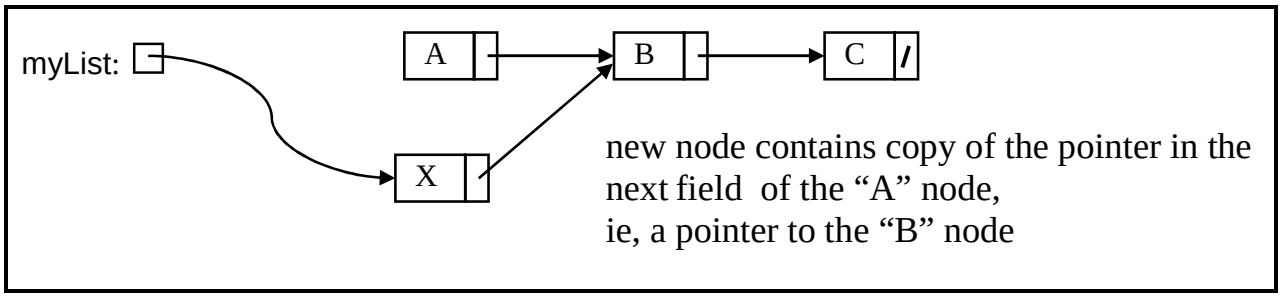
Question 4 Linked Lists [17 Marks]

(a) [2 Marks] Suppose the variable myList contains a linked list of three items ("A", "B", and "C"). Draw a diagram of the linked list.



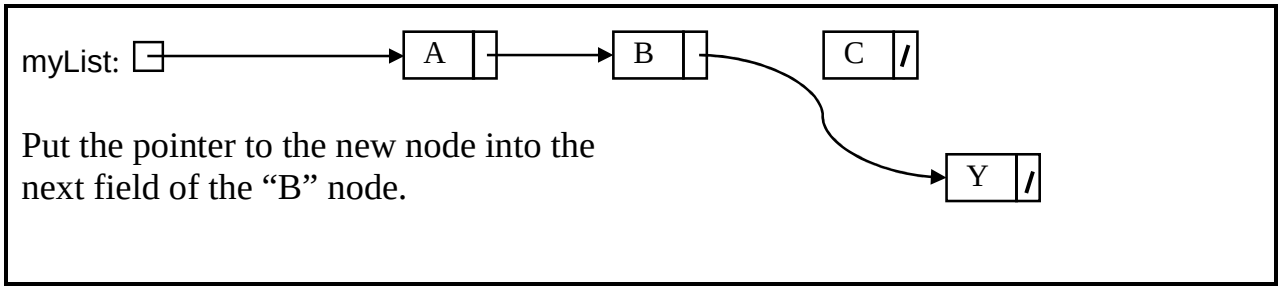
(b) [2 marks] Show the changes to the linked list in (a) after the following statement:

myList = new LinkedNode("X", myList.next);



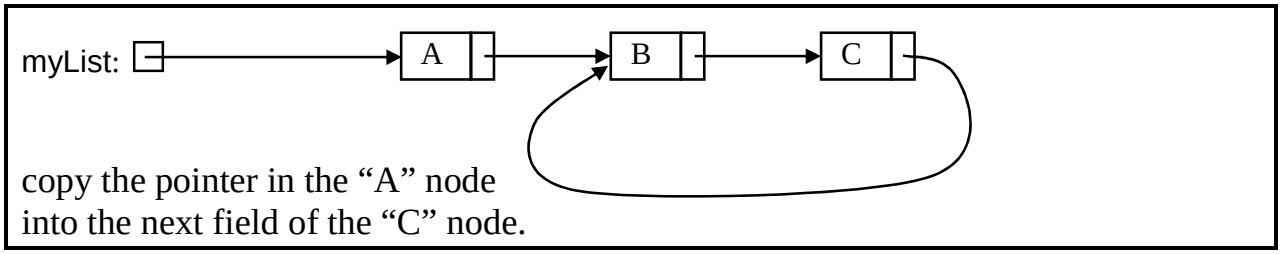
(c) [2 marks] Show the changes to the ***original*** linked list in (a) after the following statement:

myList.next.next = new LinkedNode("Y", null)

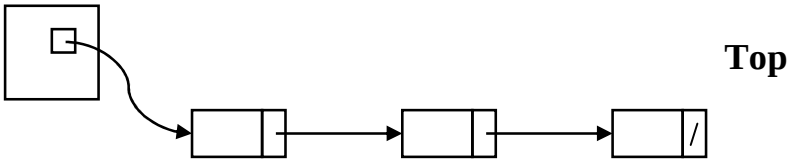


(d) [2 marks] Show the changes to the ***original*** linked list in (a) after the following statement:

myList.next.next.next = myList.next;



(e) [4 Marks] Suppose you implemented a Stack using a linked list with a header node as shown below, and chose to use the last node of the linked list as the top of the stack (where items are pushed and popped). Explain why this is a bad design.

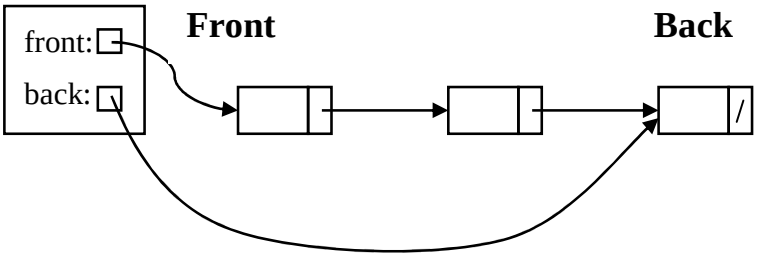


Nodes have to be added and removed from the Top of the stack.
In this design,

- adding a node (push) means replacing the null in the last node of the list with the new node. This requires traversing the whole list to get to the last node, ie, O(n) cost.
- removing a node (pop) means accessing the value in the last node of the list, and replacing the next field of the second to last node by null. Again, this requires traversing the whole list to get to the last node, ie, O(n) cost.

Given that the alternative design (Top at the head of the list) has only O(1) cost for both operations, this is a bad design.

(f) [5 Marks] Suppose you were implementing a Queue using a linked list and a header node containing two fields: front and back. Draw a diagram of how you would implement the queue, showing a queue with three elements. Explain why you chose this implementation.



A queue must add items to the back of the queue and remove items from the front. Items can be added and remove from the head of a list with O(1) cost
If you have a pointer to the last node of a list, then items can be added with O(1) cost, but it still requires O(n) cost to remove the last node (you have to change the 2nd to last node). This design has two pointers, one to the head and one to the end of the linked list, and does the remove (dequeue from the front) at the head of the list, and the add (enqueue at the back) at the end of the list, which are both O(1) operations.

ID: ……..

Question 5. Using Collections [15 Marks]

Most programming languages require that the syntax is properly nested. For example, an if statement cannot start inside a while loop but end outside the loop. A compiler must therefore check that every "opening" item has a matching "closing" item, ***and*** that the constructs are properly nested.
Consider a language that doesn't use brackets or braces, but uses the keywords **if** and **fi** to open and close an if statement, and the keywords **do** and **od** to open and close a loop. The program

`"xxx do xxx xxx if yyy yyy fi xxx od zzz"`

would be properly nested, but the programs

`"xxx do xxx if yyy od xxx fi"` and `"xxx do yyy if yyy od zzz"`

would not be properly nested.

Complete the following method that takes an argument containing a List of the words in a program and returns true if the program is correctly nested, and returns false otherwise. Assume that the only opening and closing words are **if**, **fi**, **do**, and **od**, and that all other words can be ignored.

```
public boolean checkNested(List<String> program){
    Stack<String> stack = new Stack<String>();
    for (String token : program){
        if (token.equals("do") || token.equals("if") )
            stack.push(token);
        else if (token.equals("od")){
            if ( stack.isEmpty() || ! stack.pop().equals("do") )
                return false;
        }
        else if (token.equals("fi")){
            if ( stack.isEmpty() || !stack.pop().equals("if") )
                return false;
        }
    }
    return stack.isEmpty();
}
```

Uses a stack to keep track of all the opening words.
Every closing word must match the opening word on the top of the stack.
All other tokens are ignored.
Since every opening word must have a matching closing word, the stack must be empty at the end.