1. **Combinational Logic**

   Consider a **3-bit** binary number X represented in 3-bit 2's complement format.

   (a) Write out the truth table for the following Boolean function of X: the absolute value of X is greater than 2

   (b) Write out the sum-of-products form of this Boolean function.

   (c) Now, using AND, OR, and NOT gates, draw a combinational circuit of the same function. Your AND and OR gates may have any number of inputs.

2. **Regular Expressions, Deterministic Finite State Automata**

   We have the three letter alphabet { a, b, c } and the language of all strings that start and end with a.

   Here are some examples of strings, and whether they are in the language:

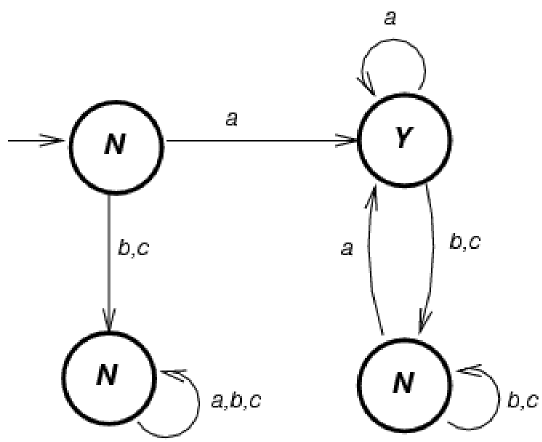
```
  Yes             No

-------     ---------------
  a           empty string
 abca             abc
abacaa            baca
```

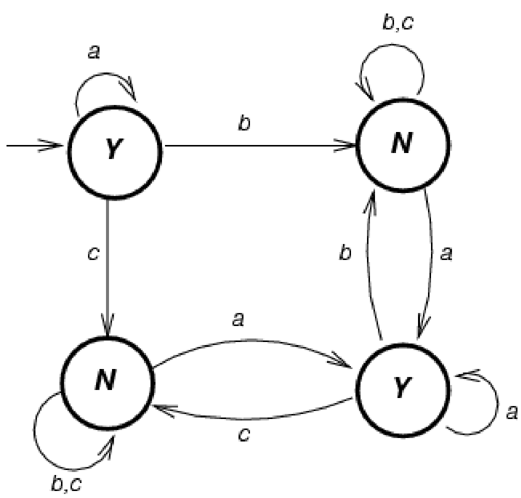
   a) Which one of these Regular Expressions generates all strings that start and end with a? Circle the roman numeral that goes with your answer.

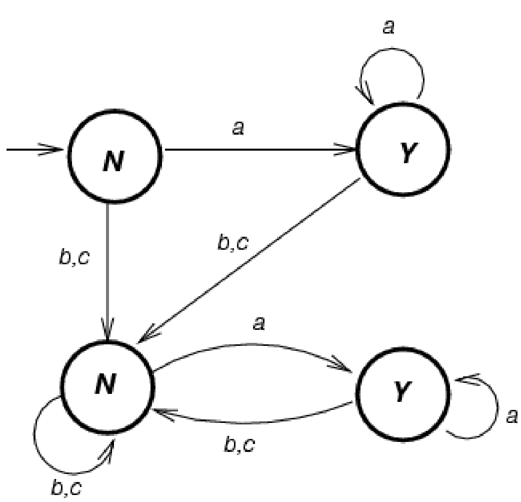
```
  i)    a* ( a | b | c )* a

 ii)    a ( a | b | c )* a

iii)    a ( ( b | c )* a )*

 iv)    a* ( ( b | c )* a )*

  v)    a ( b | c )* a
```

2. **RE, DFA continued**

b) Which one of the following DFA accepts all strings that start and end with a? Circle the roman numeral that goes with your answer.

i) 

ii) 

iii) 

3. **Linked Lists** Assume you have access to the private Node class:

```
private class Node {
  double value;
  Node next;
}
```

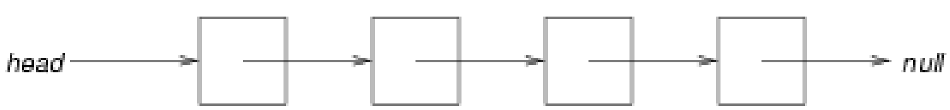
Consider the following method which operates on linked lists:

```
public boolean linky_dink (Node head) {
   Node a,b;
   a = head;
   if (a == null) return true;
   b = a.next;

   while ( b != null && b != a ) {
      b = b.next;
      if (b == null) return true;
      b = b.next;
      a = a.next;
   }

   return (b == null);
}
```

(a) What does `linky_dink` return on the following lists? Circle your answer.

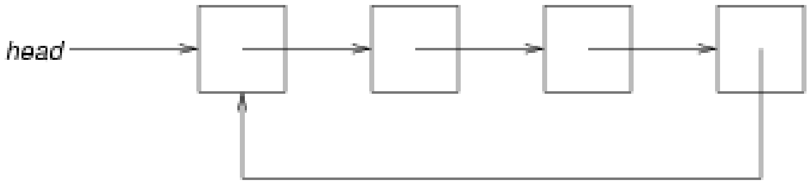
i) 

returns true   returns false   does not return

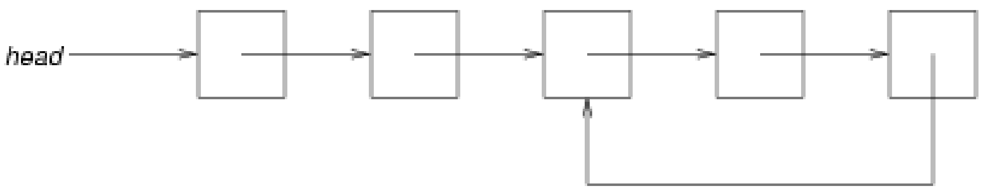
ii) 

returns true   returns false   does not return

3. **Linked Lists continued**

iii) 

returns true   returns false   does not return

iv) 

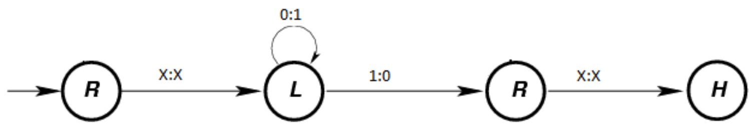
returns true   returns false   does not return

(b) What does `linky_dink` do?

(c) If your linked list has N nodes, what is the complexity of `linky_dink`? Circle your answer.

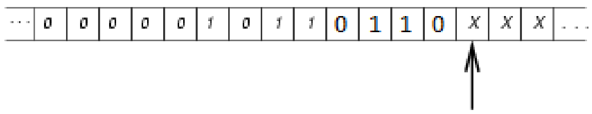
N $N \log N$ N^2 2^N

4. **Turing Machine**



a) The Turing Machine above starts in the leftmost state. If this Turing Machine is run on the tape below, with the tape head starting at the position marked by the arrow, what will be the contents of the tape when it halts, AND where will the head be?

Write your answer in the empty tape below.



b) What computation does this Turing Machine perform?

5. **Data Structures** Circle your answer.

   Circle the data structure that is the most appropriate choice for the described problem.

   (a) Store and retrieve student records, which have unique usernames.
   Array Linked List Queue Symbol Table

   (b) Store all student grades and retrieve all grades higher than 90.
   Linked List Binary Search Tree Symbol Table Stack

   (c) Implement the Back button in a browser
   Queue Binary Search Tree Stack Circular Linked List

6. **Theory True or False** Circle your answer.

T F (a) P is the set of search problems solvable in Polynomial time by a deterministic Turing Machine.

T F (b) NP is the set of search problems not solvable in Polynomial time by a deterministic Turing Machine.

T F (c) For proper encapsulation, instance variables should always be declared public.

T F (d) Because the Halting Problem is unsolvable, it is impossible to tell if *your* TSP program for Assignment 6 has an infinite loop.

T F (e) A Universal Turing Machine can compute anything that any other Turing Machine could possibly compute.

T F (g) If P equals NP, then the Traveling Salesperson Problem can be solved in polynomial time by a deterministic Turing Machine.

T F (h) If P does not equal NP, then there is no case of the Traveling Salesperson Problem for which you can find the optimal tour in polynomial time.

T F (j) Factoring is known to be in NP but has not been proven to be NP-complete, so the discovery of a polynomial-time algorithm for factoring would mean that P equals NP.

T F (k) Factoring is known to be in NP but has not been proven to be NP-complete, so no polynomial-time algorithm for factoring is possible.