COS 126 **General Computer Science** **Spring 2005**

Midterm 2

This test has 10 questions worth a total of 50 points. You have 120 minutes. The exam is closed book, except that you are allowed to use a one page cheatsheet, one side only, handwritten by you. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. **Write out and sign the Honor Code pledge before turning in the test.**
"I pledge my honor that I have not violated the Honor Code during this examination."

Signature

Problem	Score	Possible
0		1
1		8
2		6
3		6
4		3
5		5
6		8
7		4
8		3
9		6
Total		50

Name:

NetID:

Preceptor: Dan Mona

Phil Benedict

Zafer Donna

0. **Miscellaneous. (1 point)**

   (a) Write your name and Princeton NetID in the space provided on the front of the exam, and circle the name of your preceptor.

   (b) *Write* and sign the honor code on the front of the exam.

1. **Encapsulation, ADT (8 points)** In this problem, we will define a simple ADT, Account, for maintaining bank accounts. Each account has a balance, and your job is to write a constructor, as well as methods to (1) deposit money into an account (2) withdraw money from an account (3) transfer money between accounts, and (4) get balances. Assume all arguments sent to the methods are ≥ 0. Make sure your methods match the comments and the given output.

```
public class Account {
   private int balance;

   //constructor, initializing account balance to init
   public Account(int init) {




   }

   // deposit amt into account
   public void deposit(int amt) {




   }

   // withdraw amt from account if there is enough balance
   // otherwise, print an error message and withdraw nothing
   public void withdraw(int amt) {








   }
   // transfer amt to the account b if there is enough balance
   // otherwise, print an error message and transfer nothing
   public void transfer(int amt, Account b) {










   }

   // get current balance
   public int getBalance() {



   }

   public static void main(String[] args) {
      Account princeton = new Account(100000000);
      Account student1 = new Account (1000);
      Account student2 = new Account (1000);
      System.out.println("Student1 account has " + student1.getBalance());
      System.out.println("Student2 account has " + student2.getBalance());
      System.out.println("Princeton account  has " + princeton.getBalance());
      student1.withdraw(100);
      student2.withdraw(500);
      student1.transfer(800, princeton);
      student2.transfer(800, princeton);
      System.out.println("Student1 account  has " + student1.getBalance());
      System.out.println("Student2 account has " + student2.getBalance());
      System.out.println("Princeton account has " + princeton.getBalance());
   }
}
```

The test client given in `main` outputs:

```
Student1 account has 1000
Student2 account has 1000
Princeton account has 100000000
Insufficient funds
Student1 account has 100
Student2 account has 500
Princeton account has 100000800
```

2. **Regular Expressions, Deterministic Finite State Automata (6 points)**

   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
```

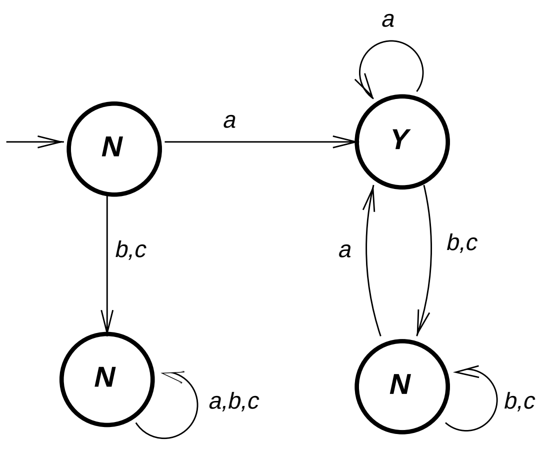
   a) Which one of these Regular Expressions generates all strings that start and end with a? Circle the roman numeral that goes with your answer.

```
  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
```

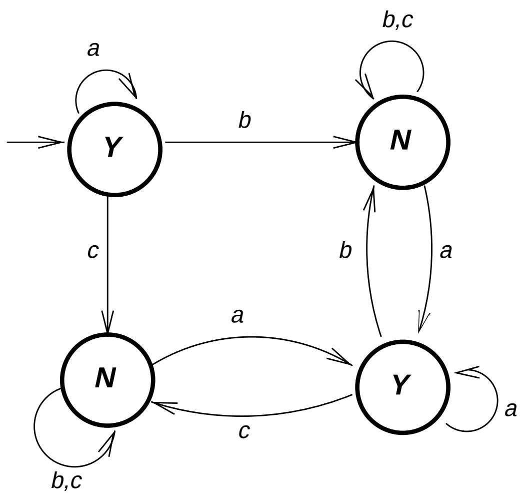
2. **RE, DFA continued**

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.

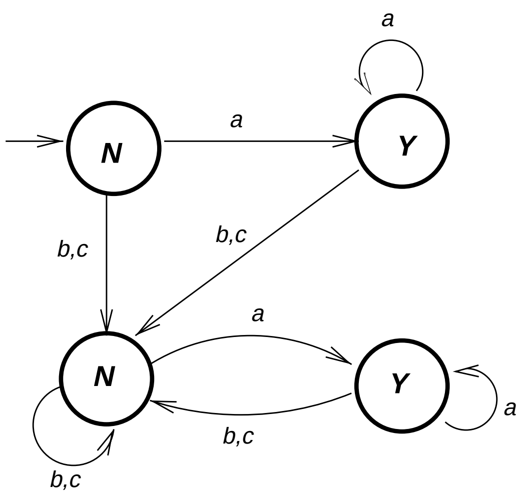
i)



ii)



iii)



3. **Linked Lists (6 points)** Assume you have access to the private Node class:

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

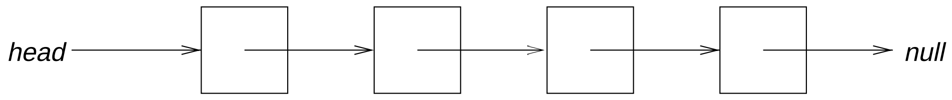
Consider the following method which operates on linked lists:

```
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) $$\begin{tikzcd} \textit{head} \arrow[r] & \textit{null} \end{tikzcd}$$

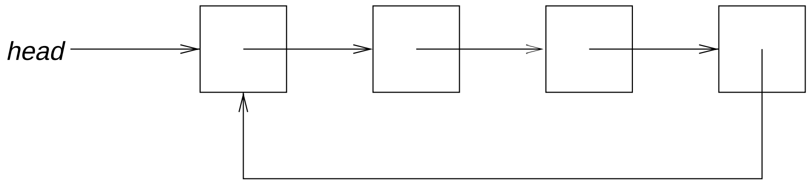
returns true   returns false   does not return

ii) 

returns true   returns false   does not return

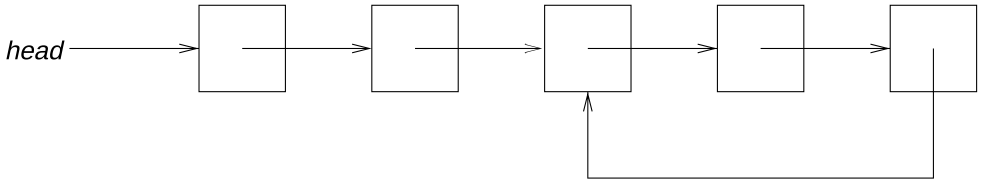
3. **Linked Lists continued**

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. **Analysis of Algorithms (3 points)**

Each of the three Java methods below takes a positive integer N as input and returns an integer equal to N^N. Circle the complexity of the `method1`, `method2`, and `method3` functions. (Assume operations like + and * take a constant amount of time.) Don't worry about the numbers getting too big to fit in an `int`.

(a)
```
public static int method1(int N) {
     int count = N;
     int toAdd = 1;
     int pow = 0;

     for (int i = 0; i < N; i++) {
         pow = 0;
         for (int j = 0; j < N; j++) {
            pow = pow + toAdd;
         }
         toAdd = pow;
     }
     return pow;
}
```

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

(b)
```
public static int method2 (int N) {
   int pow = 1;
   for (int i = 0; i < N; i++) {
         pow = pow * N;
   }
   return pow;
}
```

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

(c)
```
public static int method3 (int N) {
   return method3Help(N, N);
}

public static int method3Help (int x, int N) {
   if (N == 0) return 1;
   int pow = method3Help (x, N/2);
   pow = pow * pow;
   // one more multiple of x for an odd power
   if (N%2 == 1) pow = pow * x;
   return pow;
}
```

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

5. **QuickSort (5 points)**

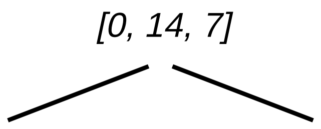
   Computers at Hogwarts School of Magic use a variant of QuickSort named MagicSort, which works like QuickSort except for its pivot selection strategy. Given an array containing a range of numbers to sort, MagicSort magically "knows" the best pivot to choose. It chooses the element that will be in the middle when the range is sorted (or the one before the middle if there are an even number of elements).

   (a) Given that MagicSort(A, i, j) sorts the range i...j (inclusive) in array A, complete the recursion tree below for MagicSort(A, 0, 14) on

   $$A = \{7, 1, 3, 6, 0, 2, 5, 12, 9, 11, 14, 4, 8, 13, 10\}$$

   Each node in your tree should contain [i, j, pivot value].

   You may omit nodes where i equals j.



   (b) If picking the pivot element takes constant time, what is the complexity of MagicSort? Circle your answer.

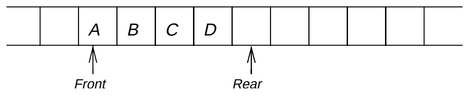
   $\log N$ N $N \log N$ N^2 2^N

6. **Queue (8 points)**

If the maximum capacity for a queue is known beforehand, it can be implemented using a "wrap-around" array. Two indices in the array are stored: `front` and `rear`. During queue operations, either index only moves "forward" in a cyclic fashion through the array; i.e., if it needs to move forward when at the very end, it goes back to the beginning.

When the queue is empty or full, `front` equals `rear`, so it is necessary to keep a count of the items in order to distinguish between the empty and full cases.

Here is a diagram of the queue with four elements, A, B, C, and D. `front` is the index of the first element in the queue. `rear` is the index where a new element should be put.



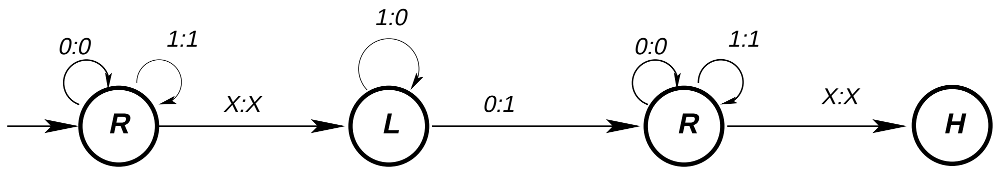
Here is a class `ArrayQueue`. The instance variables and constructor are complete. Write the code for the `ArrayQueue` methods `enqueue`, `dequeue`, `isEmpty`, and `isFull`.

```
public class ArrayQueue {
   private double[] queue; // Array that holds the queue elements
   private int capacity;   // maximum size of the queue
   private int numItems;   // number of items currently in queue
   private int front;      // index of front of queue
   private int rear;       // index of rear of queue

   // Constructor
   public ArrayQueue(int maxSize) {
      queue = new double[maxSize];
      capacity = maxSize;
      numItems = 0;      // nothing in queue yet
      front = 0;        // index of front of queue
      rear = 0;          // index of rear of queue
   }


   // Add an element to the rear of this queue if there is room.
   // If there is no room left on the queue, just return.
   public void enqueue(double item) {









   }
```

6. **Queue continued**

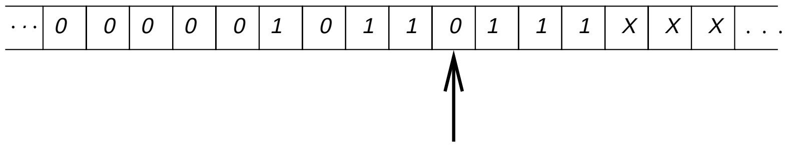
```
    // Remove and return the element from the front of this queue
    // If there are no elements on the queue, return 0.
    public double dequeue() {















    }

    // Check if this queue is empty
    public boolean isEmpty() {






    }

    // Check if this queue is full
    public boolean isFull() {






    }
}
```

7. **Turing Machine (4 points)**



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.



···	0	0	0	0	0	1	0	1	1	0	1	1	1	X	X	X	. . .

b) What computation does this Turing Machine perform?

8. **Data Structures (3 points)** Circle your answer.

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

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

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

   (c) Represent the relationships between the professors, their classes, and the students in those classes.
   Graph   Binary Search Tree   Parallel Arrays   Circular Linked List

9. **True or False (6 points)** Circle your answer.

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

T F (b) NP is the set of decision 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 (f) If Bob wants to send a message to Alice using RSA encryption, he would first encrypt his message with his own public key, and then encrypt the result with Alice's public key.

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 (i) In a symbol table implementation using a hash table, a good hash function would distribute the keys more or less evenly over the symbol table positions.

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.

T F (l) The Turing Test is a test of whether a problem can be solved by a Turing Machine.