COS 126 Written Exam 2 Fall 2019

There are nine questions on this exam, weighted as indicated. The questions are ordered corresponding to the lectures, *not in order of difficulty*. If a question seems difficult to you, skip it and come back to it. You will have 50 minutes to complete the exam.

This exam is preprocessed by computer. You answer questions by filling in circles *completely* with a dark pencil. If you change your mind, you must erase *completely* and fill in another circle!

Do this ● not this  or this  or this  .

Resources. You may reference your optional two-sided 8.5-by-11 handwritten "cheat sheet" during this exam. You may not use the textbook, your notes, or any electronic devices. You may not communicate with anyone except the course staff during this exam.

Discussing this exam. Due to travel for extracurriculars and sports, some of your peers will take this exam later. Do not discuss its contents with anyone who has not taken it.

This page. Do not remove this exam from the exam room. Fill in this page now, but do not start the exam until you are told.

Name	
NetID	
Precept	
Exam Room	

"I pledge my honor that I have not violated the Honor Code during this examination."

_____

_____

[*copy the pledge here*]

_____

[*signature*]

Q1. Data Type Terminology (2 points).

Consider the following Java code.

```
public class HelloWorld
{
   public static void main(String[] args)
   {
      StdOut.println("Hello, World");
   }
}
```

In the boxes below, write the letter corresponding to the best matching description on the right. Use each letter *at most once*.

☐	`class`	**A**	Identifies the method that writes to standard output.
☐	`public`	**B**	Identifies the method as shared by the class, rather than one associated with each object instantiated from the class.
☐	`static`	**C**	Identifies the method as available for use by any other program.
☐	`void`	**D**	Identifies the method as not returning any value.
☐	`main`	**E**	Identifies the method that is automatically invoked when you run the program.
☐	`args`	**F**	An array of strings containing the command line arguments.
☐	`StdOut`	**G**	A group of related methods and variables.
☐	`println`	**H**	Identifies the library.

Q2. Performance (2 points).

Q2a. A Java programmer experiences the approximate running times shown in the table at left for a program that reads digital photos, for various file sizes. Fill in the one circle in the table at right that gives the best hypothesis for the order of growth of the running time of this program as a function of the file size.

file size	*approximate running time*
1 megabyte	30 seconds
5 megabytes	half a day
50 megabytes	7 weeks

- logarithmic ○
- linear ○
- quadratic ○
- cubic ○
- *none of these* ○

Q2b. Another program does some image processing that involves a constant amount of time per pixel. The aspect ratio is 4:3, so the program represents each photo as a two-dimensional pixel array of size 4R by 3R (one entry per pixel), where the parameter R depends on the resolution of the photo. Fill in the one circle that best describes the order of growth of the running time of this program as a function of R.

- logarithmic ○
- linear ○
- quadratic ○
- cubic ○
- *none of these* ○

Q3. Divide and Conquer (2 points).

Consider the following Java code. The `mystery()` method is *not* mergesort, but has a similar recursive structure. Recall that given a string `s`, the expression `s.substring(i, j)` will produce the substring `s[i]s[i+1]...s[j-1]`, for example, if `s` is the string `"Hello, World"` then `s.substring(2,8)` will return `"llo, W"`.

```
public class Q3
{
   public static String mystery(String s)
   {
      if (s.length() < 2) return s;
      int m = s.length()/2;
      String lh = mystery(s.substring(0, m));
      String rh = mystery(s.substring(m, s.length()));
      s = rh + lh;
      return s;
   }
   public static void main(String[] args)
   {
      StdOut.println(mystery(args[0]));
   }
}
```

In the box to the left of each command, write the letter corresponding to the output it produces. A letter may be used once, more than once, or not at all.

☐ `java-introcs Q3 st`

☐ `java-introcs Q3 ess`

☐ `java-introcs Q3 sse`

☐ `java-introcs Q3 stressed`

A `st`

B `ts`

C `ses`

D `ese`

E `ess`

F `sse`

G `desserts`

H `deerssst`

I `stressed`

J `ssedstre`

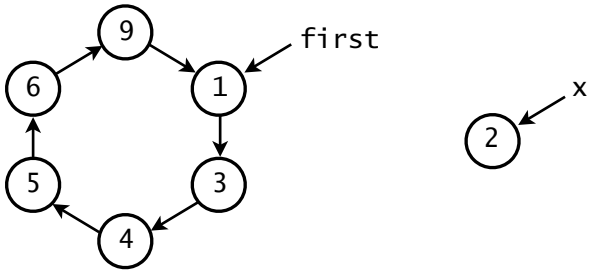
K *none of the above*

Q4. Linked Structures (2 points).

Suppose that a `Node` data type is defined as

```
private class Node
{
   private int item;
   private Node next;
}
```

and that `first` and `x` are variables of type `Node`, where `first` refers to one node in a circular linked list, as in this example.



For each code fragment below, fill in one of the circles to its right to indicate whether it does the job of inserting `x` into the circular linked list immediately after the node that `first` is pointing to. Assume that the initial circular linked list is not empty. *Hint*: You will find this question easier if you start by drawing the desired result.

	does the job	*does not do the job*
`x.next = first.next;` `first.next = x;`	◯	◯
`first.next = x;` `x.next = first.next;`	◯	◯

`x.next = first;`
`x.next.next = x;`
`first.next = x;`
	◯	◯

`Node z = first;`
`x.next = first.next;`
`z.next = x;`
	◯	◯

Q5. BSTs (1.5 points).

Q5a. Suppose that we create a BST by inserting integers into an initially empty tree. In the blank to the right of each of the following insertion orders write the height of the tree produced (max number of links on any path from the root to a leaf node) when keys are inserted in that order into an initially empty tree. The first answer is provided for you.

	3	4	5	6	7
1 2 3 4 5	○	●	○	○	○
1 500 600 700 527	○	○	○	○	○
605 256 490 300 527	○	○	○	○	○
10 860 523 602 599 610 527	○	○	○	○	○

Q5b. Suppose that you are searching for the key 527 in a binary search tree built with integer keys. In the following table, fill in one circle in each row to indicate whether it is possible or impossible for keys given to be examined during the search in the order given below.

	possible	*impossible*
605 256 490 300 527	○	○
10 860 523 602 525 527	○	○
10 860 523 602 599 610 527	○	○

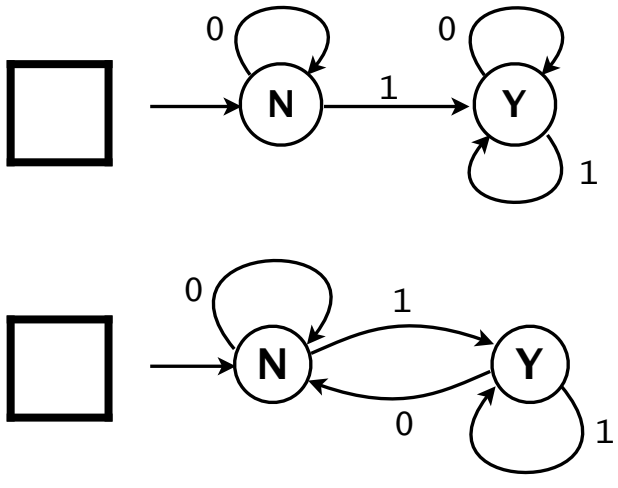
Q6a. Regular Expressions (1 point)

By filling in one of circles to the right of each regular expression, indicate whether it does the job of describing the language of all binary strings that begin with 1.

	does the job	*does not do the job*
`1(0 | 1)*`	◯	◯
`1(0* | 1*)*`	◯	◯
`1(0 | 1)*(0 | 1)*`	◯	◯
`1 | 1((0 | 1)(0 | 1))*`	◯	◯

Q6b. DFAs (1 point).

Match each of the DFAs below with the language that it recognizes, by writing a letter in the box to its left.



A Bitstrings that end in 1

B Bitstrings with an equal number of occurrences of 01 and 10

C Bitstrings with more 1s than 0s

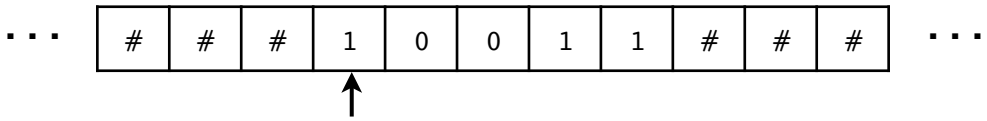
D Bitstrings with an equal number of occurrences of 0 and 1

E Bitstrings with at least one 1

Q7. Turing Machines (2 points).

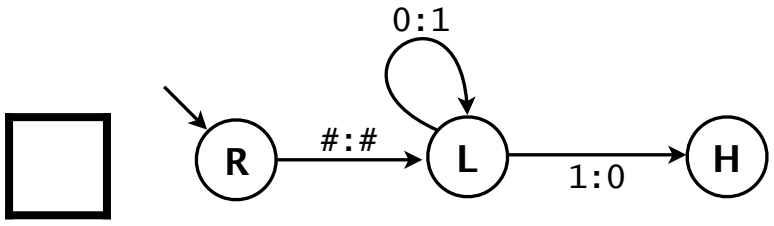
A *binary Turing machine* has just two tape symbols (`0` and `1`) in addition to the marker symbol `#`. Otherwise, it has all the properties of any other Turing machine.

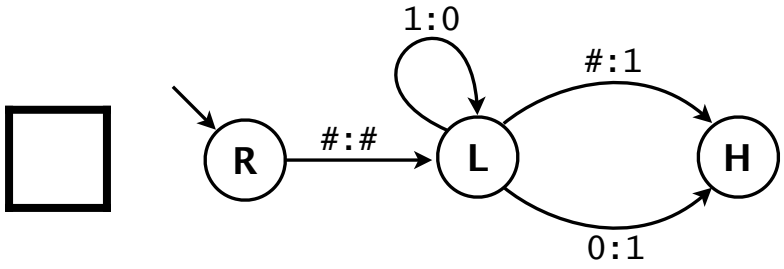
- The initial tape consists of a nonempty string of `0`s and `1`s with infinitely many `#` symbols on both sides, the initial head location is the leftmost bit of the binary string, and the *value* of the tape at any point is the binary number on the tape. For example, this diagram depicts a tape of value 19.

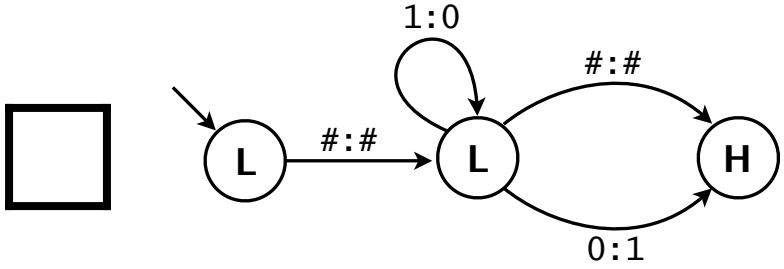


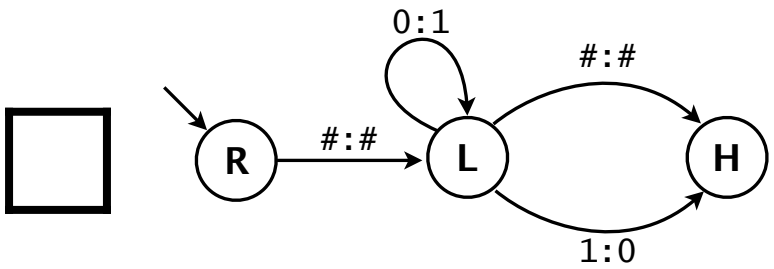
- The initial state of each machine is the leftmost in the diagram. On entering a new state, the head location moves left (`L`) or right (`R`) as indicated, or halts (`H`). The next state is determined by following the arrow whose label before the colon matches the symbol at the head location on the tape. The label after the colon is written on the tape. Then the process continues.

Match each of the TMs below with its function, by writing a letter in the box to its left.









A Binary incrementer

B Binary decrementer

C Might not halt

D Has no effect

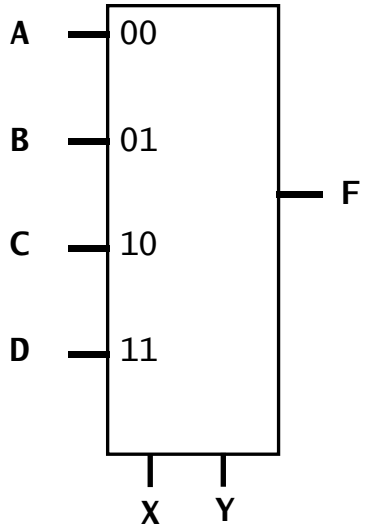
Q8. Theory of Computation (2 points).

In the box to the left of each term on the left, write the letter corresponding to the best matching description on the right. Use each letter at most once.

☐	Universal	**A**	The set of all search problems.
☐	Decidable	**B**	A simple, universal model of computation.
☐	Church-Turing thesis	**C**	The set of all search problems that can be solved in polynomial time.
☐	Satisfiable	**D**	A problem that can be solved by a Turing machine.
☐	Turing machine	**E**	A Turing machine, TOY, or your laptop.
☐	P	**F**	If you can solve a problem in this class in polynomial time, then P = NP.
☐	NP	**G**	Anything computable in this universe can be computed by a Turing machine.
☐	NP-complete	**H**	A logical equation that has a solution.

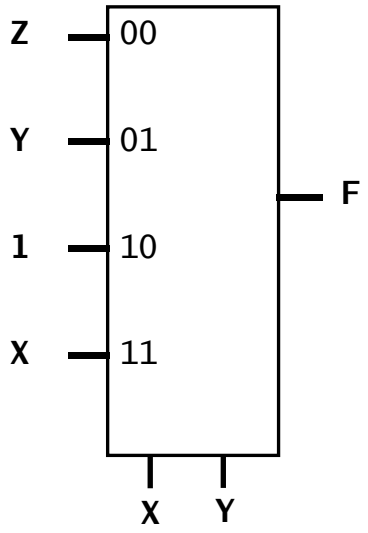
Q9. Combinational Circuits (2 points).

A *two bit multiplexer* circuit (drawn at left below) uses two *selection* input lines (X and Y) to choose among four *data* input lines (A, B, C, and D) for the output value (F) as shown in the table at right. For example, if X is 1 and Y is 0, then they address the input line labeled 10 so that the output value is C.



X	Y	F
0	0	A
0	1	B
1	0	C
1	1	D

Now, suppose that a two-bit multiplexer is wired up as shown at left below, with three input values and a constant 1. Find the output value (0 or 1) for all possible input values (fill in the truth table at right).



X	Y	Z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	