COS126 **Written Exam 2** **Fall** '23

Instructions. This exam has nine (9) questions worth a total of one hundred (100) points. You have eighty (80) minutes.

This exam is preprocessed by computer. Write neatly, legibly, and darkly. If you use a pencil, use extra care to write darkly. Fill in bubbles and checkboxes <u>completely</u>: ⚫ and ⬛ (not ✔ or ✘). Place <u>only</u> your answer <u>inside</u> a box, although you may show work outside a box. Write neatly and legibly.

To change an answer, <u>erase it completely</u> and redo.

Resources. The exam is closed book, except that you are allowed to use a single double-sided reference sheet (8.5-by-11 paper, both sides, in your own handwriting). No electronic devices are permitted.

Honor Code. This exam is governed by Princeton's Honor Code. Discussing the contents of this exam before solutions have been posted is a violation of the Honor Code.

NAME: SOLUTION

NETID ———

PRECEPT

P01 ◯ P02 ◯ P02A ◯ P03 ◯ P04 ◯ P05 ◯ P07 ◯ P08 ◯ P08A ◯

P10 ◯ P11 ◯ P12 ◯ P12A ◯ P13 ◯ P14 ◯ P15 ◯ P16 ◯ P16A ◯

EXAM ROOM McCosh 50 ◯ McCosh 10 ◯ McCosh 62 ◯ McCosh 66 ◯ Other ______

"I pledge my honor that I will not violate the Honor Code during this examination."

Signature

Question 1 — Object-Oriented Programming — **11** points

Write a single letter in each box corresponding to the best matching description (below) for each underlined statement from the Counter class. Letters may be used more than once or not at all.

```
01 public class Counter {
02   private final String name;
03   private final int maxCount;
04   private int count;
05   public Counter(String id, int max) {
06     name = id;
07     maxCount = max;
08     count = 0;
09   }
10   public void increment() {
11     if (count < maxCount) count++;
12   }
13   public int value() {
14     return count;
15   }
16   public String toString() {
17     return name + ": " + count;
18   }
19   public boolean lessThan(Counter that) {
20     int c = this.count;
21     return (this.count < that.count);
22   }
23   public static void main(String[] args) {
24     int n = Integer.parseInt(args[0]);
25     int trials = Integer.parseInt(args[1]);
26     Counter[] hits = new Counter[n];
27     for (int i = 0; i < n; i++) {
28       hits[i] = new Counter(i + "", trials);
29     }
30     for (int t = 0; t < trials; t++) {
31       int index = StdRandom.uniformInt(n);
32       hits[index].increment();
33     }
34     for (int i = 0; i < n; i++) {
35       String s = "Counter " + hits[i];
36       StdOut.println(s);
37     }
38   }
39 }
```

`02 private final String name;`	A	
`04 private int count;`	A	
`05 public Counter(String id, int max) {`	C	B
`14 return count;`	G	
`17 return name + ": " + count;`	G	
`20 int c = this.count;`	B	F G
`23 public static void main(String[] args) {`	E	B
`28 hits[i] = new Counter(i + "", trials);`	L	F
`32 hits[index].increment();`	I	F
`35 String s = "Counter " + hits[i];`	K	F B
`36 StdOut.println(s);`	J	

- **A.** Instance variable declaration
- **B.** Local variable declaration
- **C.** Signature of constructor
- **D.** Signature of instance method
- **E.** Signature of static method
- **F.** Refers to the object whose instance method or constructor is being called
- **G.** Directly accesses an instance variable
- **H.** Defines a collection of related methods
- **I.** Directly calls an instance method
- **J.** Directly calls a static method
- **K.** Implicitly calls `toString()`
- **L.** Calls a constructor

Question 2	**Searching & Sorting**	**10** points

For each statement 1-7, write the letter of the one-word characterization that best describes the order of growth of the **worst-case** running time.

A. Constant **B**. Logarithmic **C**. Linear **D**. Linearithmic **E**. Quadratic **F**. Cubic **G**. Exponential

1. Performing ***n*** sequential searches on an array of ***n*** elements. E
2. Performing ***n*** binary searches on a sorted array of ***n*** elements. D
3. Insertion sort (to put into increasing order) on a pre-sorted array (already in increasing order) of ***n*** elements. C
4. Insertion sort (to put into increasing order on a pre-sorted array (already in decreasing order) of ***n*** elements. E
5. Mergesort on an array of ***n*** elements. D
6. Mergesort (to put into increasing order) on a pre-sorted array (already in increasing order) of ***n*** elements. D
7. Mergesort (to put into increasing order) on a pre-sorted array (already in decreasing order) of ***n*** elements. D

9. Using **mergesort** to arrange eight (8) characters in alphabetical order, which of the following could be the values of the sub-lists (each in brackets) *just prior to the final merge.* Fill in the bubbles for **all** that apply.

- ○ [B G L] [A D X] [M R]
- ○ [A F Q] [C J M O P]
- ● [Q R Y Z] [A F P V]
- ● [W X Y Z] [A B C D]
- ○ [D A F M] [C O S W]
- ○ [A] [B] [C] [D] [E] [F] [G] [H]
- ○ All of the above
- ○ None of the above

Question 3	**Stacks and Queues**	**12** points

1. Consider the following Java program that uses the COS 126 `Stack` and `Queue` ADTs. The program prints words on three lines on standard output. You can assume that the *foreach* for a Stack uses LIFO order and the *foreach* for a Queue uses FIFO order.

```
public class Q4 {
  public static void main(String[] args) {
    Stack<String> s = new Stack<String>();
    Queue<String> q = new Queue<String>();
    while (!StdIn.isEmpty())
      q.enqueue(StdIn.readString());
    for (String str : q) {
      StdOut.print(str + " ");
      s.push(str);
    }
    StdOut.println();
    for (String str : s) {
      StdOut.print(str + " ");
      q.enqueue(str);
    }
    StdOut.println();
    while (!q.isEmpty())
      StdOut.print(q.dequeue() + " ");
    StdOut.println();
  }
}
```

If standard input contains:

`hello how are you`

For each line, write the letter (in the box) that corresponds to the words printed on standard output.

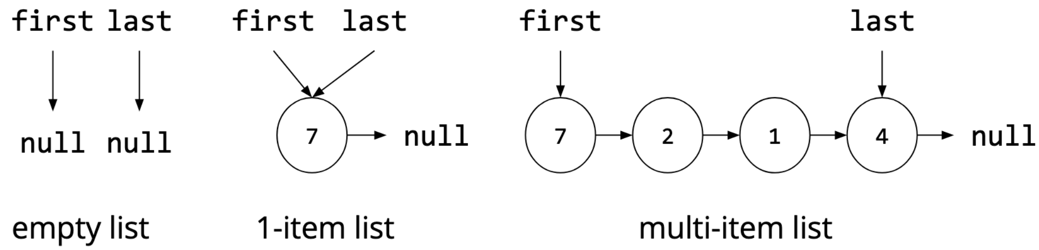
First output line	E
Second output line	G
Third output line	J

A. hello
B. how
C. are
D. you
E. hello how are you
F. are hello how you
G. you are how hello
H. you how hello are
I. hello how are you hello how are you
J. hello how are you you are how hello
K. you are how hello you are how hello
L. you are how hello hello how are you
M. hello hello how how are are you you
N. you you are are how how hello hello
O. hello you are how are you hello how

Question 4 **Linked Lists** **12** points

Suppose that a Node data type is defined (below) and that `first` and `last` are variables of type Node that refer to the *first* and *last* node, respectively, in a singly-linked list. Some examples are shown below.

```
public class Node {
  private int key;
  private Node next;
  public Node(int key) {
    this.key = key;
    this.next = null;
  }
}
```



Each code snippet below updates the linked list and maintains a singly-linked list (null-terminated). Write the letter in the box that corresponds to the best description of the operation of the code snippet. Assume the operation of each code snippet is independent of one another.

1.
```
Node x = new Node(5);
if (first == null)
  first = x;
else
  last.next = x;
last = x;
```
B

2.
```
if (first == last) {
   first = null;
   last = null;
}
else {
  Node x = first;
  while (x.next != last)
     x = x.next;
  x.next = null;
  last = x;
}
```
F

3.
```
if (first != last) {
  Node x = first;
  while (x.next != last)
     x = x.next;
  x.next = null;
  last.next = first;
  first = last;
  last = x;
}
```
G

4.
```
if (first != last) {
  Node x = first;
  first = x.next;
  last.next = x;
  x.next = null;
}
```
H

A. Adds a new Node to the beginning of the list.

B. Adds a new Node to the end of the list.

C. Adds a new Node before the last Node.

D. Adds a new Node after the first Node.

E. Removes the first Node.

F. Removes the last Node.

G. Moves the original last Node before the original first Node.

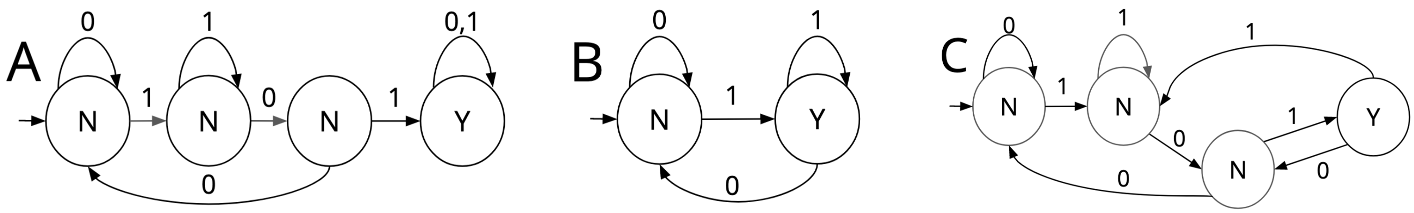
H. Moves the original first Node after the original last Node.

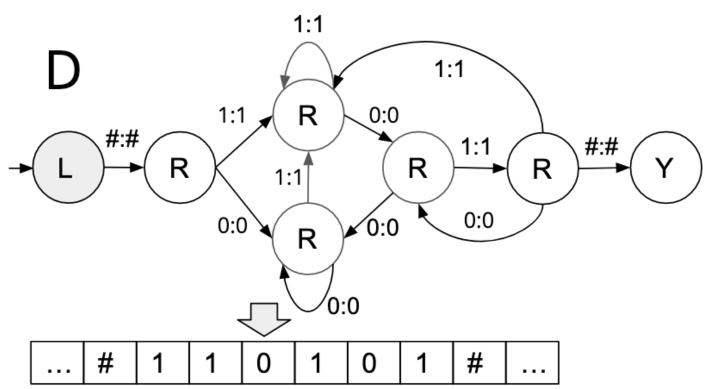
Question 5 **Binary Search Trees** **10** points

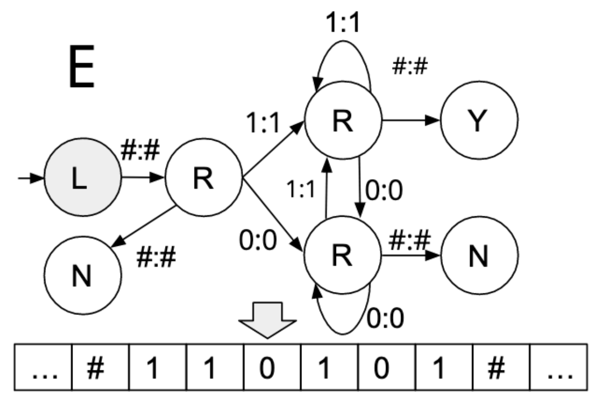
Suppose we have int values between 1 and 100 in a Binary Search Tree and we search for 68. Fill in the YES bubble for any of the following that could be the sequence of keys examined in a search for 68. Fill in the NO bubble for any sequence that could not result.

	YES	**NO**	**Sequence of keys examined**
1.	●	○	97 13 80 20 60 68
2.	○	●	14 80 40 32 50 68
3.	●	○	12 26 30 48 50 62 68
4.	○	●	41 97 20 28 68
5.	●	○	97 7 53 86 68

The first row (below) shows three DFAs, labeled A, B and C, defined over the alphabet {0, 1}. The second row shows two Turing Machines (TMs), where a tape contains a binary string with infinite #'s on both sides of the string. Example tapes and tape head starting locations are provided. For each language, write the letter of a DFA that recognizes the language in the box in the first column, and write the letter of a TM that recognizes the language in the box in the second column. If there is not a DFA or Turing Machine listed that recognizes the language, write the letter **X** in the corresponding box.







Language	**DFA**	**TM**
1. Contains the binary string 101	A	X
2. Binary string ends with 101	C	D
3. Binary string that ends with 1	B	E

4. For each DFA and TM fill the bubble for *terminates* if the DFA/TM always terminates or *infinite loop* if the DFA/TM can potentially go into an infinite loop. If *infinite loop* is selected, provide one example string that results in an infinite loop.

DFA/TM	**Terminates**	**Infinite Loop**	**Example of a string that produces an infinite loop**
A	●	○	
B	●	○	
C	●	○	
D	○	●	11
E	○	○	

Question 7	TOY	**12** points

For each of the following TOY programs, assume standard input starts with the following sequence of HEX data values. (So each program starts reading the first value **1111**).

1111 2222 2222 1111 2222 7B08 C016 3333 C016 FFFF

Assuming each TOY program starts at memory location **10** what are the values of **R[1]** and **R[2]** after each program finishes? Write one hex digit per box.

1.
```
10: 7100
11: 82FF
12: 1222
13: D216
14: 1102
15: C011
16: 0000
```

R[1]				R[2]			
0	0	0	0	2	2	2	2

2.
```
10: 7100
11: 82FF
12: 2221
13: C216
14: 1102
15: C011
16: 0000
```

R[1]				R[2]			
1	1	1	1	0	0	0	0

3.
```
10: 7100
11: 82FF
12: 9213
13: 0000
14: 1102
15: C011
16: 0000
```

R[1]				R[2]			
7	B	0	8	C	0	1	6

TOY REFERENCE CARD

INSTRUCTION FORMATS

```
              | . . . . | . . . . | . . . . | . . . .|
  Format RR:  | opcode  |    d    |    s    |    t    |  (0-6, A-B)
  Format A:   | opcode  |    d    |       addr        |  (7-9, C-F)
```

```
ARITHMETIC and LOGICAL operations
    1: add              R[d] <- R[s] +  R[t]
    2: subtract         R[d] <- R[s] -  R[t]
    3: and              R[d] <- R[s] &  R[t]
    4: xor              R[d] <- R[s] ^  R[t]
    5: shift left       R[d] <- R[s] << R[t]
    6: shift right      R[d] <- R[s] >> R[t]

TRANSFER between registers and memory
    7: load address     R[d] <- addr
    8: load             R[d] <- M[addr]
    9: store            M[addr] <- R[d]
    A: load indirect    R[d] <- M[R[t]]
    B: store indirect   M[R[t]] <- R[d]

CONTROL
    0: halt             halt
    C: branch zero      if (R[d] == 0) PC <- addr
    D: branch positive  if (R[d] >  0) PC <- addr
    E: jump register    PC <- R[d]
    F: jump and link    R[d] <- PC; PC <- addr
```

Register 0 always reads 0.
Loads from M[FF] come from stdin.
Stores to M[FF] go to stdout.

16-bit registers (two's complement)
16-bit memory locations
 8-bit program counter

Question 8	**Universality, Computability and Intractability**	**10** points

For each statement, fill the bubble for *True*, *False*, or *No one is sure yet* that **best** describes each statement. Fill in the bubble for *I'm not sure* if you don't know the answer and prefer partial credit (.5 points).

1. A Universal Turing Machine (UTM) can simulate the COS 126 javac-introcs compiler.
   - ● True ○ False ○ No one is sure yet ○ I'm not sure

2. The discovery of a polynomial-time algorithm for TSP would not imply a polynomial-time algorithm for FACTOR.
   - ○ True ● False ○ No one is sure yet ○ I'm not sure

3. No Turing Machine can decide whether a given DFA halts.
   - ○ True ● False ○ No one is sure yet ○ I'm not sure

4. NP is the class of search problems for which, in principle, a Java program could be written to check a given proposed solution in polynomial time.
   - ● True ○ False ○ No one is sure yet ○ I'm not sure

5. 3-SAT can be solved in polynomial time on a deterministic Turing Machine.
   - ○ True ○ False ● No one is sure yet ○ I'm not sure

6. There exists a polynomial time reduction from TSP to FACTOR.
   - ○ True ○ False ● No one is sure yet ○ I'm not sure

7. Showing that FACTOR poly-time reduces to TSP would show that FACTOR is NP-Complete.
   - ○ True ● False ○ No one is sure yet ○ I'm not sure

8. The Church-Turing thesis implies that no computer can solve the halting problem.
   - ● True ○ False ○ No one is sure yet ○ I'm not sure

9. If a quantum computer is successfully built, it could provide a counterexample to the Extended Church-Turing thesis.
   - ● True ○ False ○ No one is sure yet ○ I'm not sure

10. There does not exist a polynomial time algorithm for FACTOR.
    - ○ True ○ False ● No one is sure yet ○ I'm not sure

Question 9	**Circuits**	**11** points

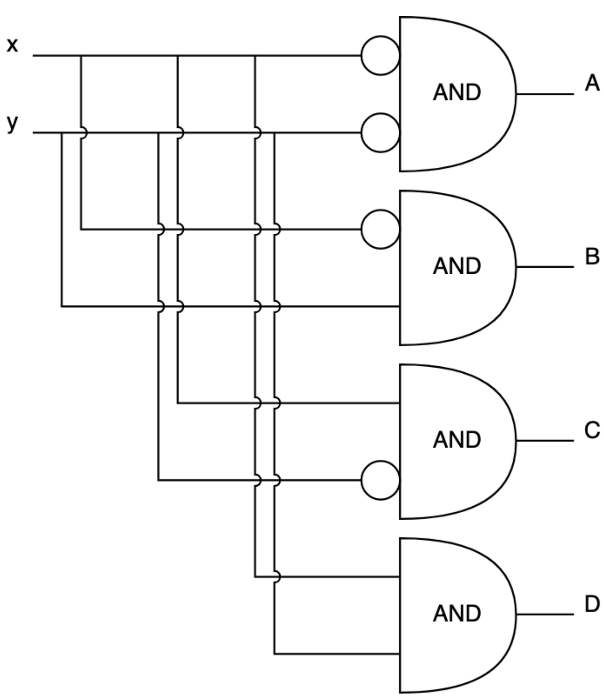
1. Consider the function **F(x, y, z)** that is true if and only if **xyz** is a 3-bit two's complement integer whose **absolute value** is ≥ 3. Here **x** is the most significant (leftmost) bit and **z** is the least significant (rightmost) bit for each integer. Complete the truth table by filling in the values in the column labeled **F(x, y, z).**

x	y	z	F(x, y, z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

2. What is the sum-of-products formula for **F(x, y, z)**? Print your answer in this box. Write legibly and neatly please.

x'yz + xy'z' + xy'z

3. Suppose the circuit below has inputs **x = 0 y = 1**. What are the values of the outputs **A, B, C, D**? Enter one bit per box (below):



A	B	C	D
0	1	0	0

4. In **one** word, what does this circuit implement? Print your answer in the box below:

decoder

BLANK PAGE

BLANK PAGE