COS 126	Princeton University	Fall 2024

Written Exam 2 Solutions

This exam has 1010_2 questions worth a total of 64_{16} points. You have 80_{10} minutes.

Instructions. This exam is preprocessed by computer. Write neatly, legibly, and darkly. Put all answers (and nothing else) inside the designated answer spaces. *Fill in* bubbles and checkboxes completely: ● and ■. To change an answer, erase it completely and redo.

Resources. The exam is closed book, except that you are allowed to use a one page 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 are posted is a violation of the Honor Code.

Please complete the following information now.

Name: Ada Lovelace

NetID: `alovelace`

Exam room: ○ McCosh 10 ● McCosh 50 ○ McCosh 66 ○ Other

Precept:

P01	P01A	P02	P03	P03A	P03B	P04	P04A	P05	P05A
○	○	○	●	○	○	○	○	○	○

P06	P10	P10A	P11	P12	P13	P14	P14A	P15
○	○	○	○	○	○	○	○	○

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

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

Ada Lovelace

Signature

1. **Object-oriented programming. (10 points)**

For each Java keyword or construct on the left, write the letter of the best-matching object-oriented concept from the right. Use each letter at most once.

B	`class`	**A.**	API
C	`final`	**B.**	Data type
J	`null`	**C.**	Immutability
D	`private`	**D.**	Encapsulation
A	`public`	**E.**	Fail-fast principle
E	`throw`	**F.**	Object creation
F	*constructor*	**G.**	Object destruction
K	*instance methods*	**H.**	State of an object
H	*instance variables*	**I.**	Identity of an object
I	*object reference*	**J.**	Identity of *no* object
		K.	Behavior of an object

2. **Designing data types. (10 points)**

(a) Which of the following data types used in this course are *immutable*?
Fill in all checkboxes that apply.

- ■ `String`
- ■ `Vector`
- □ `String[]`
- □ `Perceptron`
- □ `StringBuilder`
- □ `GuitarString`
- □ `Picture`
- □ `LapTimer`

(b) Which of the following are primary reasons for *encapsulating* a data type?
Fill in all checkboxes that apply.

- □ To use less memory.
- □ To make program faster.
- ■ To make it easier to reuse code.
- ■ To make it easier to reason about code.
- ■ To develop client code and implementation code independently.
- ■ To ensure that a client can modify a data type's value only through the API.

3. **Creating data types and debugging. (12 points)**

Consider the following partial implementation of a data type:

```
public class Mystery {

    // declare array[]     (SEE FACING PAGE)

    // define constructor  (SEE FACING PAGE)

    public void set(int i, String s) {
        array[i] = s;
    }

    public String toString() {
        String result = "";
        for (int i = 0; i < array.length; i++)
            result += array[i];
        return result;
    }
}
```

Also, consider the following client program:

```
public class MysteryClient {
    public static void main(String[] args) {
        String[] a = { "A", "B", "C" };
        String[] b = a;
        String[] c = { "X", "Y", "Z" };
        Mystery x = new Mystery(a);
        Mystery y = new Mystery(b);
        Mystery z = new Mystery(c);
        a[0] = "D";
        y.set(1, "E");
        StdOut.println(x);
    }
}
```

Substituting each code fragment on the facing page to declare `array[]` and define the constructor of `Mystery`, what will be printed to standard output?

For each code fragment on the left, write the letter of the best-matching description from the right. Use each letter once, more than once, or not at all.

A

```
private String[] array;

public Mystery(String[] a) {
    array = new String[a.length];
    for (int i = 0; i < a.length; i++)
        array[i] = a[i];
}
```

I

```
private String[] array;

public Mystery(String[] a) {
    String[] array = new String[a.length];
    for (int i = 0; i < a.length; i++)
        array[i] = a[i];
}
```

D

```
private final String[] array;

public Mystery(String[] a) {
    array = a;
}
```

E

```
private static String[] array;

public Mystery(String[] a) {
    array = new String[a.length];
    for (int i = 0; i < a.length; i++)
        array[i] = a[i];
}
```

A. `ABC`

B. `AEC`

C. `DBC`

D. `DEC`

E. `XEZ`

F. `XYZ`

G. *empty string*

H. *array index out-of-bounds exception*

I. *null pointer exception*

J. *compile-time error*

```
                    TOY REFERENCE CARD


INSTRUCTION FORMATS

              | . . . . | . . . . | . . . . | . . . .|
  Format RR:  | opcode  |    d    |    s    |    t    |  (1-6, 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


 16 16-bit registers:        R[0]  to R[F]
256 16-bit memory locations: M[00] to M[FF]
  1  8-bit program counter:  PC

R[0] always reads as 0000.
Loads from M[FF] come from stdin.
Stores to  M[FF] go to stdout.
```

4. **TOY programming. (10 points)**

(a) Set the program counter to `10` and run the following TOY program. *How many* values are printed to standard output?

Fill in the bubble of the answer.

```
10: 7A10
11: 7101
12: CA16
13: 9AFF   write R[A]
14: 2AA1   R[A] = R[A] - R[1]
15. C012
16: 0000   halt
```

○ *none* ○ *one* ○ *nine* ○ *ten* ● *sixteen* ○ 31,248 ○ *infinite*

(b) Set the program counter to `10` and run the following TOY program. What are the *first* and *last* values printed to standard output?

Write each of the four hexadecimal digits in the provided boxes.

```
10: 7101   R[1] = 0001
11: 7220   R[2] = 0020
12: 1421   R[4] = R[2] + R[1]
13: A302   R[3] = M[R[2]]
14: A204   R[2] = M[R[4]]
15: 93FF   write R[3]
16: D212   if (R[2] > 0) goto 12
17: 0000   halt

20: 6666
21: 0028
22: 0000
23: 0000
24: 1111
25: 0020
26: 5555
27: 0000
28: 2222
29: 0026
2A: 0000
```

values printed:
6666
2222
5555

5. **TOY machine. (10 points)**

For each description on the left, write the letter of the best-matching integer from the right. Use each letter once, more than once, or not at all.

Answer	Description
D	Number of 0s in the binary representation of −142. *Assume 16-bit two's complement integer.*
L	Sum of the hexadecimal integers `02A0` and `0160`. *Hint: add in hex.*
N	Number of distinct values representable by a TOY register.
G	Number of multiway-*OR* gates in a 16-bit adder (using the ripple–carry design from lecture).
R	Value of the Java expression `32768 * 65536`. *Hint: you do not need a calculator.*

A. 0

B. 2^0 (1)

C. 2^1 (2)

D. 2^2 (4)

E. 2^3 (8)

F. 2^4 (16)

G. 2^5 (32)

H. 2^6 (64)

I. 2^7 (128)

J. 2^8 (256)

K. 2^9 (512)

L. 2^{10} (1,024)

M. 2^{15} (32,768)

N. 2^{16} (65,536)

O. 2^{31} (2,147,483,648)

P. 2^{32} (4,294,967,296)

Q. -2^{32} (-4,294,967,296)

R. -2^{31} (-2,147,483,648)

6. **Machine learning. (8 points)**

   (a) The *New Jersey Forest Fire Service* seeks to develop a machine-learning model to predict whether to issue a wildfire warning on a given day. The model will rely upon using historical data, including daily wind speed, humidity, temperature, and past wildfire warnings. Which kind of machine learning problem is this?

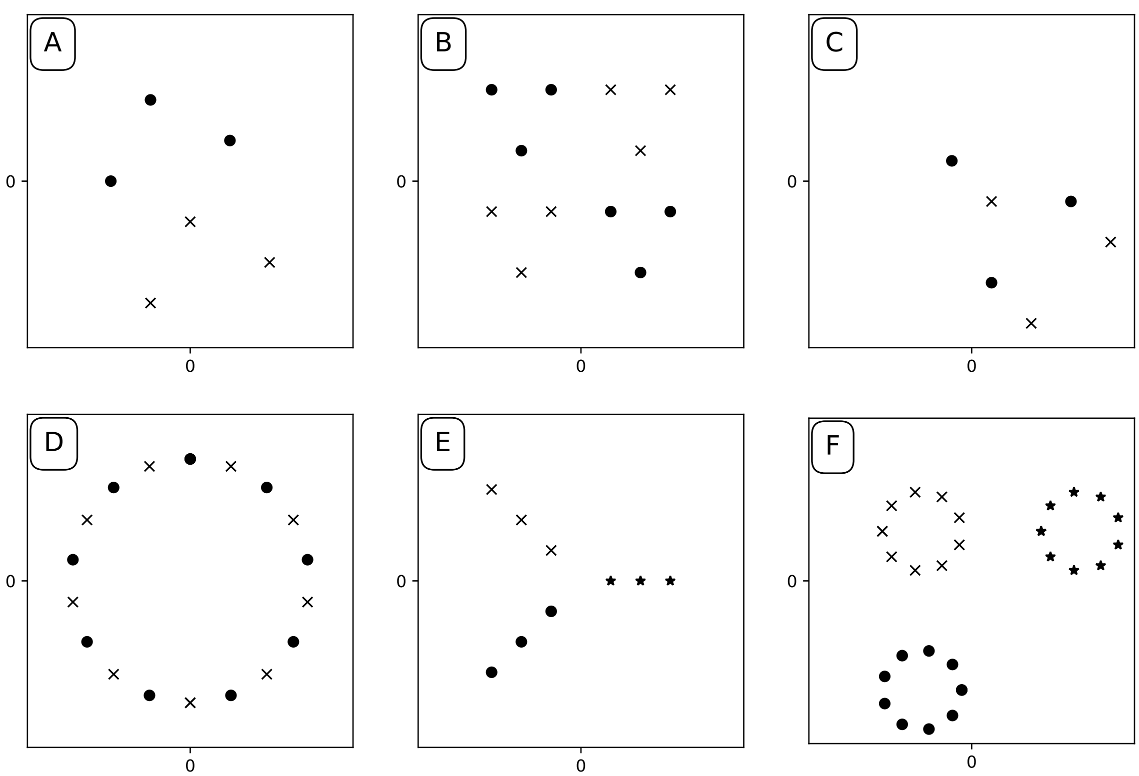
   Fill in the bubble of the best-matching answer.

   ○ *reinforcement learning*
   ● *supervised learning*
   ○ *transfer learning*
   ○ *unsupervised learning*
   ○ *zero-shot learning*

   (b) Consider the following scatter plots, where the x- and y-axes represent numerical *features* of the data; and the circles, crosses, and stars represent different *classes* of data points. Which of the following datasets are *linearly separable* (i.e., there exists a linear model that can perfectly separate the classes)?

   Fill in all checkboxes that apply.

   ■ A
   ☐ B
   ☐ C
   ☐ D
   ■ E
   ■ F



7. **Insertion sort and mergesort. (10 points)**

The leftmost column contains an array of 24 integers to be sorted; the rightmost column contains the integers in sorted order; the other columns are the possible contents of the array at some intermediate step of *insertion sort* or *mergesort* (as implemented in the lecture and textbook). By *intermediate step*, we mean at the very end of some iteration of insertion sort or immediately after some call to `merge()` in mergesort. *Hint: think about the properties of insertion sort and mergesort.*

Consider each column independently. Write the letter of the best-matching description under the corresponding column. You may use each letter once, more than once, or not at all.

54	11	11	28	54	11
57	28	28	54	57	22
89	39	39	57	63	28
63	52	52	63	89	29
28	54	54	89	28	39
79	57	57	79	52	47
74	63	63	74	74	52
52	70	70	52	79	54
70	74	74	70	11	57
11	79	79	11	39	58
39	89	89	39	70	63
91	91	91	91	91	64
72	22	72	72	22	68
47	29	47	47	47	70
22	47	22	22	72	72
81	58	81	81	81	73
84	64	84	84	64	74
68	68	68	68	68	79
88	72	88	88	84	80
64	73	64	64	88	81
58	80	58	58	29	84
80	81	80	80	58	88
29	84	29	29	73	89
73	88	73	73	80	91
A	C	D	B	E	F

A. Original array

B. Insertion sort only

C. Mergesort only

D. Both insertion sort and mergesort

E. Neither insertion sort nor mergesort

F. Sorted array

8. **Data structures. (10 points)**

Each of the following code fragments reads n real numbers from standard input and uses a data structure to shuffle them in uniformly random order. Assuming each data structure performs as expected (and that `uniformInt()` takes constant time), determine the *worst-case running time* as a function of n.

For each code fragment on the left, write the letter of the best-matching term from the right. You may use each letter once, more than once, or not at all.

C

```
double[] a = new double[n];
for (int i = 0; i < n; i++) {
    double x = StdIn.readDouble();
    int r = StdRandom.uniformInt(i+1);
    a[i] = a[r];
    a[r] = x;
}
```

E

```
Queue<Double> queue = new Queue<Double>();
for (int i = 0; i < n; i++) {
    double x = StdIn.readDouble();
    int r = StdRandom.uniformInt(i+1);
    queue.enqueue(x);
    for (int j = 0; j < r; j++)
        queue.enqueue(queue.dequeue());
}
```

D

```
ST<Integer, Double> st = new ST<Integer,Double>();
for (int i = 0; i < n; i++) {
    double x = StdIn.readDouble();
    int r = StdRandom.uniformInt(i+1);
    st.put(i, st.get(r));
    st.put(r, x);
}
```

E

```
LinkedList<Double> list = new LinkedList<Double>();
for (int i = 0; i < n; i++) {
    double x = StdIn.readDouble();
    int r = StdRandom.uniformInt(i+1);
    list.add(r, x);  // insert at position r in list
}
```

A. $\Theta(1)$ *constant*

B. $\Theta(\log n)$ *logarithmic*

C. $\Theta(n)$ *linear*

D. $\Theta(n \log n)$ *linearithmic*

E. $\Theta(n^2)$ *quadratic*

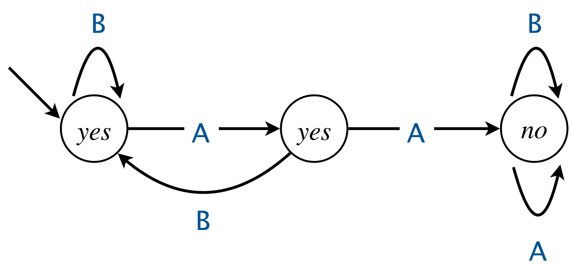
F. $\Theta(n^3)$ *cubic*

G. $\Theta(2^n)$ *exponential*

9. **Theory of computing. (10 points)**

(a) Describe the set of strings that the following DFA matches.

Fill in the bubble corresponding to the best-matching description.



- ○ starts with AA
- ○ ends with AA
- ○ contains AA
- ○ does *not* start with AA
- ○ does *not* end with AA
- ● does *not* contain AA

(b) Identify each statement below as *known to be true*, *known to be false*, or *unknown*.

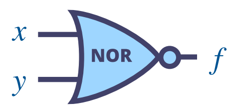
true	*false*	*unknown*	
●	○	○	Any computational problem that can be solved on a TOY machine, can also be solved on a Turing machine.
○	●	○	It is possible to write a Java program to determine whether two Turing machines always produce the same output (when given identical starting tapes).
○	○	●	It is possible to harness the power of general relativity to build a physical device that can solve the halting problem.
○	●	○	Given a Java function with no arguments, *ChatGPT-4* can determine whether the function will go into an infinite loop.
●	○	○	A *universal Turing machine* can simulate the behavior of any individual Turing machine.
○	●	○	Any Turing machine with n states that halts must halt after at most 2^n steps.
○	●	○	A Turing machine *without* the ability to *write* to the tape is a *universal* model of computation.

10. **Digital circuits. (10 points)**

A *NOR* gate is defined by the following truth table, Java function, and schematic symbol:

x	y	f
0	0	1
0	1	0
1	0	0
1	1	0

```
public static boolean nor(boolean x, boolean y) {
    return !(x || y);
}
```

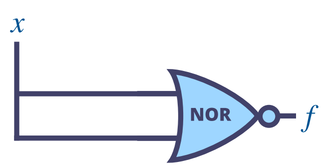


For each Java expression or circuit on the left, write the letter of the best-matching function on the right. Use each letter at most once.

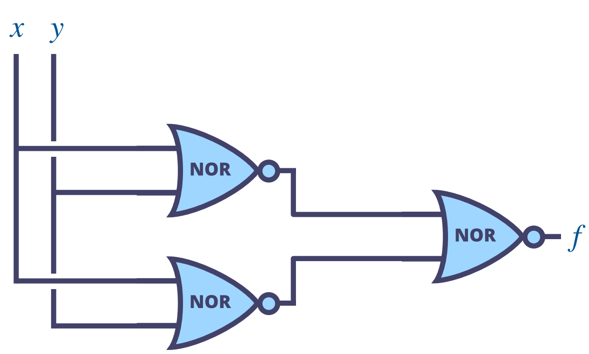
A — `nor(x, y)`

B — `nor(nor(x, x), nor(y, y))`

D —



C —



A. $NOR(x, y)$

B. $AND(x, y)$

C. $OR(x, y)$

D. $NOT(x)$

E. $XOR(x, y)$

F. $NAND(x, y)$

G. *none of the above*