COS 126	Princeton University	Spring 2025

Written Exam 1

This exam has 8 questions worth a total of 100 points. You have 80 minutes.

Instructions. This exam is preprocessed by computer. Write neatly, legibly, and darkly. Put all answers (and nothing else) inside the designated spaces. *Fill in* bubbles completely, like this: ●. To change an answer, erase it completely and redo. Do not remove or unstaple any pages.

Resources. The exam is closed book, except that you are allowed to use a one-page reference sheet (8.5-by-11 paper, one side, 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:

NetID:

Exam room: ◯ McCosh 46 ◯ McCosh 50 ◯ Other

Precept:

P01	P02	P03	P03A	P04	P04A	P05	P05A	P06
◯	◯	◯	◯	◯	◯	◯	◯	◯

P10	P10A	P10B	P12	P14	P15
◯	◯	◯	◯	◯	◯

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

Signature

1. **Initialization. (2 points)**

   In the designated spaces on the front of this exam,

   - *Print* your name.
   - *Print* your Princeton NetID (6–8 characters, like "`cs0126`").
   - *Fill in* the bubble corresponding to the room in which you are taking this exam.
   - *Fill in* the bubble corresponding to your precept.
   - *Write* and *sign* the Honor Code pledge.

2. **Java expressions and types. (12 points)**

For each Java expression below, write the letters corresponding to the TYPE and VALUE of the expression. If the expression would result in a compile-time error or run-time exception, write "E" for both TYPE and VALUE. You may use each letter once, more than once, or not at all.

TYPE	VALUE	Expression
		`7 + 3 / 2`
		`true == (12 < 6)`
		`3.0 * 2 / 5`
		`4 - 1 * 3 - 2`
		`5 / 0`
		`!(true && false || true)`
		`Integer.parseInt("1.2")`
		`Math.sqrt(16) / Math.pow(2, 2)`
		`2.0 == (2 * 1.0)`
		`(int)(4.4 - 3.2)`
		`"COS" + 4 * 3 + 6`
		`Math.min(1.2, Math.max(2, 8))`

TYPE Options:

A. `boolean`

B. `int`

C. `double`

D. `String`

E. *Error*

VALUE Options:

E. *Error*

F. `-1`

G. `0`

H. `1`

I. `1.0`

J. `1.2`

K. `2`

L. `8`

M. `16`

N. `COS18`

O. `COS126`

P. `false`

Q. `true`

3. **StdIn, Data Types, and Arrays (16 points)**

true	*false*	
◯	◯	Standard input (`StdIn`) has no limit on its length.
◯	◯	`StdIn` cannot be used in any program that uses command-line arguments.
◯	◯	`StdIn` provides a program with the capability to accept input at any time during its execution.
◯	◯	Calls to `StdIn` methods may be interleaved (mixed in any order) with calls that write to standard output.
◯	◯	The public and private methods in `StdIn` form the API for this library.
◯	◯	Finding the largest value in a list of integers read from `StdIn` requires using an array.
◯	◯	The values of any Java data type can be stored as a sequence of 0s and 1s.
◯	◯	A data type is defined as a set of values and operations on those values.
◯	◯	All Java variables and function parameters must have a data type that is known at compile time.
◯	◯	Type casting in Java is the process of converting one data type to another.
◯	◯	In Java, variables of all data types use the same size in memory.
◯	◯	Arrays can be used to store a sequence of values that have different types.
◯	◯	Arrays facilitate processing large amounts of data.
◯	◯	Arrays are primarily used to avoid using `for` loops.
◯	◯	A two dimensional array can be used to store a math matrix.
◯	◯	A three dimensional array can be used to store all the pixels in a *video* (which can be thought of as a sequence of images).

4. **Standard input, loops, and conditionals. (16 points)**

Consider the Java program `Test.java` and the text file `input.txt`:

```
public class Test {
    public static void main(String[] args) {
        int result = -1;
        while (!StdIn.isEmpty()) {
            int a = StdIn.readInt();

            // SUBSTITUTE HERE EACH
            // CODE FRAGMENT INDEPENDENTLY

        }
        StdOut.println(result);
    }
}
```

```
% more input.txt
5 3 4 4 1 1 6 3
```

Substitute each code fragment below into the designated place in the program above. What is the result of executing the program with `java-introcs Test < input.txt`?

For each of the four code fragments, write a letter in the box to its left. Choose the letter from the right column corresponding to what would be printed on standard output. You may use each letter once, more than once, or not at all.

☐
```
if (a % 2 == 0 && a < result) {
    result = a;
}
```

☐
```
result = Math.max(result, a);
```

☐
```
result += a;
```

☐
```
if (a > result) {
    int[] arr = new int[a];
    result = arr[result];
}
```

A. `-1`

B. `0`

C. `1`

D. `3`

E. `5`

F. `6`

G. `15`

H. `26`

I. *run-time exception*

5. **Properties of functions (static methods). (14 points)**

Which of the following are properties of Java *static methods*?

Identify each statement as either true or false by filling in the appropriate bubble.

true	*false*	
◯	◯	When a Java program is run, the `main` function of *every* class (including libraries) is automatically called.
◯	◯	A function can call itself within its own body, a technique known as *recursion.*
◯	◯	A Java function with multiple return statements can return multiple values at once.
◯	◯	In Java, a function argument that is a primitive type is always passed by value.
◯	◯	An example of method *overloading* is: *a single Java class that contains two methods with the same name but different return types.*
◯	◯	An example of method *overloading* is: *a single Java function that has two parameters that have the same name but are of different types.*
◯	◯	A `void` function *may* contain zero, one or multiple `return` statements.
◯	◯	A recursive method must always include at least one base case to prevent infinite recursion (or a stack overflow error).
◯	◯	Passing a value of type `double` to a function that takes an argument of type `int` will produce a *compile-time error.*
◯	◯	A function with `return` statements may also have side effects.
◯	◯	A Java client may call a `private` function in a library if it improves performance.
◯	◯	I am feeling pretty good at this point in the exam. *(Hint: either answer is fine. Either way, take a deep breath and then continue...)*
◯	◯	In the *Conjunction Function* assignment, suppose we already have executed the code: `double[] a = StdAudio.read("cow.wav");` Then, evaluating the two expressions `mix(a, a)` and `amplify(a, 2.0)` will give equivalent results.
◯	◯	In the *Conjunction Function* assignment, the function `changeSpeed()` could possibly return an array of the same length as its array argument. For reference, the method header is: `double[] changeSpeed(double[] a, double alpha)`

6. **Arrays and loops. (16 points)**

Each code fragment below works with an array `a[]` *containing* n *integers (where* $n > 1$*). For each code fragment, choose the best-matching letter from the list on the right, and write it in the corresponding box. You may use each letter once, more than once, or not at all.*

☐
```
for (int i = 1; i < n; i++) {
    a[i] = a[i] + a[i-1];
}
```

☐
```
for (int i = 1; i < n; i++) {
    int rand = (int) (Math.random() * i);
    int temp = a[i];
    a[i] = a[rand];
    a[rand] = temp;
}
```

☐
```
int i = 0;
while (i < n - 1) {
    if (a[i] > a[i + 1]) {
        int temp = a[i];
        a[i] = a[i + 1];
        a[i + 1] = temp;
        i = 0;
    } else {
        i++;
    }
}
```

☐
```
int i = 0;
while (i < n) {
    if (i % 2 == 0) {
        i++;
    } else {
        int temp = a[i];
        a[i] = a[i-1];
        a[i-1] = temp;
    }
}
```

A. *elements of* `a[]` *unchanged*

B. *reverse elements in* `a[]`

C. *sort elements of* `a[]` *ascending*

D. *shuffle elements in* `a[]`

E. *compute running sum of* `a[]`

F. *infinite loop*

G. `ArrayIndexOutOfBoundsException`

7. **Performance. (12 points)**

Determine the order-of-growth of running time relative to input n. *For each function below, write in the corresponding box to its left the letter of the best-matching term from the right column. You may use each letter once, more than once, or not at all.*

☐
```
public static int f1(int n) {
    int count = 0;
    for (int i = n; i > 0; i = i / 2)
        count++;
    return count;
}
```

☐
```
public static int f2(int n) {
    int count = 0;
    for (int i = 0; i < n; i++)
        for (int j = i; j < n; j++)
            for (int k = i; k < n; k++)
                count++;
    return count;
}
```

☐
```
public static int f3(int n) {
    int count = 0;
    for (int i = n; i >= 0; i++)
        count += Math.min(i, n);
    return count;
}
```

☐
```
public static int f4(int n) {
    int count = 0;
    for (int i = 0; i < 1000; i++)
        count = count + n;
    for (int i = 0; i < 100; i++)
        count = count - n;
    return count;
}
```

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

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

C. $\Theta(\sqrt{n})$ *square root*

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

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

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

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

H. $\Theta(n^4)$ *quartic*

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

J. *infinite loop*

8. **Recursive graphics. (12 points)**

This function draws a T-tree of order n, analogous to the H-tree that we studied in class:

```
    // draw a T-tree of order n, centered at (x, y), of specified size
    public static void draw(int n, double size, double x, double y) {
1      if (n == 0) return;
2      double x0 = x - size/2, x1 = x + size/2;  // x left and right
3      double y0 = y - size/2, y1 = y + size/2;  // y bottom and top
4      StdDraw.line(x0, y1, x1, y1);             // horizontal top bar
5      StdDraw.line(x,  y0, x,  y1);             // vertical middle bar
6      draw(n-1, size/2, x0, y1);                // top left recursive tree
7      draw(n-1, size/2, x1, y1);                // top right recursive tree
    }
```

Suppose we reorder the 7 numbered lines of code in `draw()` and pause execution just after the 24th call to `StdDraw.line()`. This is when the 12th `T` is completed, each using 2 lines. *For each of the following orderings of the 7 numbered lines of code, write the letter corresponding to the picture that results from calling* `draw(5, 0.5, 0.5, 0.5)` *and pausing execution as described above. Scrambled orderings are underlined to help you identify them. The first answer is provided. You may use each letter once, more than once, or not at all.*

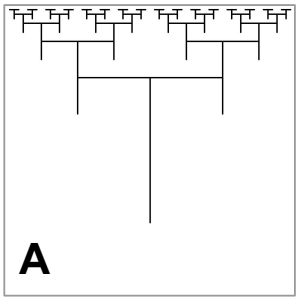
[D] 1 2 3 4 5 6 7

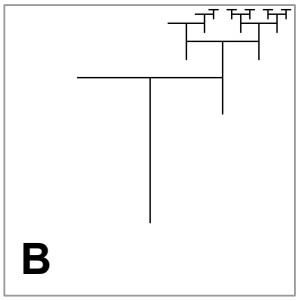
[ ] 1 2 3 4 5 7 6

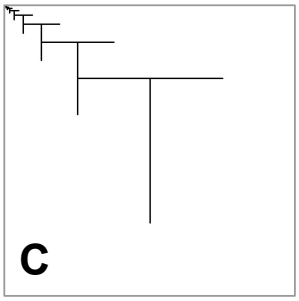
[ ] 1 2 3 5 4 6 7

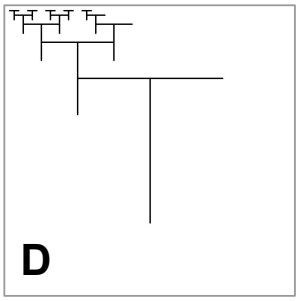
[ ] 1 2 3 6 7 4 5

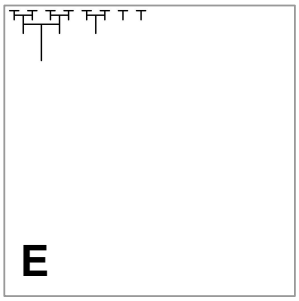
[ ] 2 3 6 7 1 4 5











F

`StackOverflowError`

This page is intentionally left blank. You may use it for scratch work.