

## 1 Looping

What output is produced by the following non standard for loop? Record it **exactly**. We strongly suggest that you use a table to keep track of the value of k.

```
1 for (int k = 1; k <= 10; k++) {  
2     k = k + 1;  
3     System.out.print(k + " ");  
4 }
```

Output:

## 2 While to For

Translate each of the following two while loops into for loops.

*Note: You must get both parts correct to get credit for this question.*

### Part One

```
1 int k = 0;  
2 while (k < 10) {  
3     System.out.println(k);  
4     k = k + 1;  
5 }
```

Select the letter corresponding to the for loop below that is equivalent to the while loop above.

**A**

```
1 for (int i = 0; i < 10; i++) {  
2     System.out.println(i);  
3 }
```

**B**

```
1 for (int i = 0; i < 10; i++) {  
2     i = i + 1;  
3     System.out.println(i);  
4 }
```

**C**

```
1 for (int i = 1; i < 10; i++) {  
2     System.out.println(i);  
3 }
```

**D**

```
1 for (int i = 1; i < 10; i++) {  
2     i = i + 1;  
3     System.out.println(i);  
4 }
```

A

B

C

D

### Part Two

```
1 int k = 0;  
2 while (k < 10) {  
3     k = k + 1;  
4     System.out.println(k);  
5 }
```

Select the letter corresponding to the for loop below that is equivalent to the above while loop.

**A**

```
1 for (int i = 0; i <= 10; i++) {  
2     System.out.println(i);  
3 }
```

**B**

```
1 for (int i = 0; i <= 10; i++) {  
2     i = i + 1;  
3     System.out.println(i);  
4 }
```

**C**

```
1 for (int i = 1; i <= 10; i++) {  
2     System.out.println(i);  
3 }
```

**D**

```
1 for (int i = 1; i <= 10; i++) {  
2     i = i + 1;  
3     System.out.println(i);  
4 }
```

A

B

C

D

### 3 Arrays

Refer to the following classes:

```

1 public class Point {
2     public int x;
3     public int y;
4     //implementation
5 }
6 public class Line {
7     public Point left_endpoint;
8     public Point right_endpoint;
9     public int slope;
10
11     public Line(Point one, Point two) {
12         this.left_endpoint = one;
13         this.right_endpoint = two;
14     }
15     //implementation
16 }

```

What is printed by the program below? Record it **exactly**.

```

1 public class Test {
2     public static void main(String [] ars) {
3         Line[] line = new Line[2];
4         Point p = new Point();
5         Point q = new Point();
6         Line pq = new Line(p, q);
7         line[0] = pq;
8         line[0].left_endpoint.x = 1;
9         pq = new Line(q, p);
10        line[1] = pq;
11        line[0].left_endpoint.y = 2;
12        line[1] = line[0];
13        line[0] = pq;
14        line[0].left_endpoint.x = 2;
15        line[0].left_endpoint.y = 1;
16        System.out.print(line[0].right_endpoint.x + " ");
17        System.out.print(line[0].left_endpoint.x + " ");
18        System.out.print(line[0].right_endpoint.y + " ");
19        System.out.print(line[0].left_endpoint.y);
20    }
21 }

```

Output:

## 4 Build Code

## Part One

Note: You must get both parts correct to get credit for this subpart.

You are writing a program `boolean isPrime(int n)` which is designed to determine if a number is prime. You know that the definition of a prime number is one that is divisible only by 1 and itself. Assume that  $n$  is a positive number and is greater than 1 throughout this entire question.

Which of the following boolean conditions is true for any number  $1 < x < n$  if the number  $n$  is prime.  `n % x == 1`

`n % x == x`

`n % x != 0`

None of the above

Which of the following would provide an inefficient but accurate implementation of the boolean `isPrime(int n)` method. Note that at least one of the below is correct.

Select the letter corresponding to the correct answer(s).

A

```

1 int x = 2;
2 while (x < n) {
3     if (!condition) {
4         return false;
5     }
6 }
7 return true;

```

B

```

1 for (int x = 2; x < n; x++) {
2     if (!condition) {
3         return false;
4     }
5 }
6 return true;

```

C

```

1 for (int x = 2; x < n; x++) {
2     if (condition) {
3         return true;
4     }
5 }
6 return false;

```

D

```

1 for (int x = 2; x < n; x++) {
2     if (!condition) {

```

6  
`6 % 2 == 0`  
`6 % 3 == 0`

7  
`7 % x != 0`  
 2  
 3  
 4  
 5  
 6

`n % x != 0`

`!condition`  
`n % x == 0`

9

9 4

```

3     return true;
4     } else {
5     return false;
6     }
7 }

```

- A
- B
- C
- D

**Part Two**

All of the implementations above are a tad inefficient, so let's make them slightly more efficient! To do so, we will change the loop's **stopping condition**. The stopping condition for *all* the implementations above is  $x < n$ . Which of the following stopping conditions are *correct* and would make the code *more efficient*?

→ Hint: You only need to check the factors until the square root of  $n$

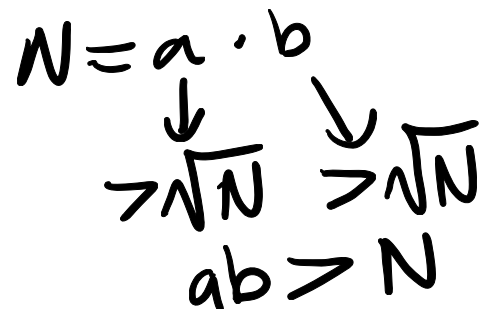
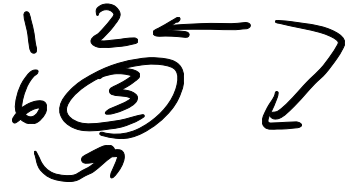
- $x \leq n * n$
- $x * x \leq n$
- $x * x * x \leq n$

**Factors (pairs of factors)**

**Prime Number:** one pair of factors (1 and itself)

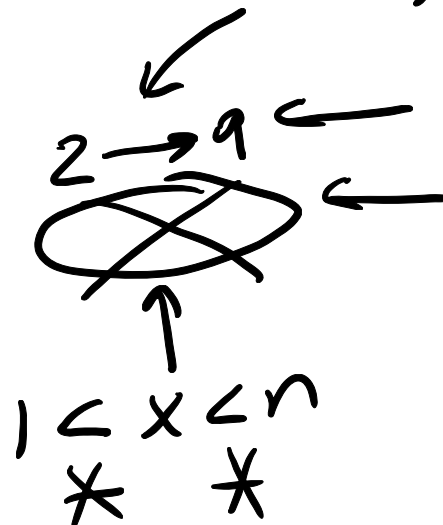
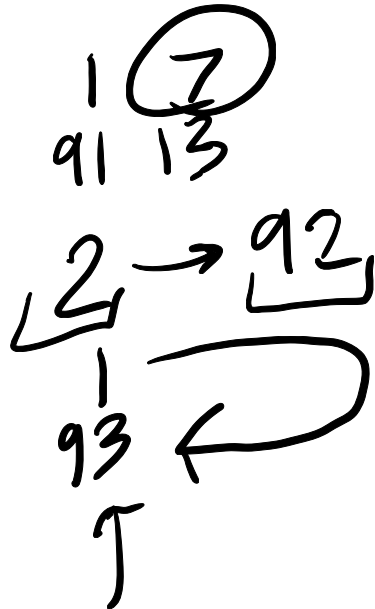
6: two different pairs  
2 and 3, 1 and 6

7: one pair  
1 and 7



$4 < 6 < 9$   
 $2 < \text{sqrt}(6) < 3$

$81 < 91 < 100$   
 $9 < \text{sqrt}(91) < 10$



$81 < 93 < 100$   
 $9 < \text{sqrt}(93) < 10$

