Question 1

Recall that Java byte variables are signed two's-complement integers. After the following Java code runs, what are the values of the four variables? Express all answers in base ten.

```java
byte a = (byte)( 56 / 10 );
byte b = (byte)( 20 * 20 );
byte c = (byte)( (byte) 0x80 + 0x40 );
byte d = (byte)( 0x7A - 0x6B );
```

1. a ==
2. b ==
3. c ==
4. d ==

Question 2

Write a static method `straight` that takes an array of integers as an argument and returns `true` if the array contains a contiguous range of integers in any order. For example:

```java
straight(new int[]{ 5, 7, 6 })  // returns true
straight(new int[]{ 1, 3, 4 })  // returns false
straight(new int[]{ -1 })       // returns true
straight(new int[]{0})         // returns true
```

You may use any standard Java library class. Don’t worry about minor details like `set` vs. `put` if your meaning is clear. The argument may be of any length, including zero, but is guaranteed not to be null.

Question 3

Suppose you have a `Node` object, defined like this.

```java
public class Node {
    int value;
    Node a;
    Node b;
}
```

A `Node` has two children, `a` and `b`. Either or both may be null. You can assume that no `Node` will have itself as a child, directly or indirectly. A `Node` is a heap if and only if both of the following are true:

1. It does not have a child with a smaller `value` field. This is trivially true if both `a` and `b` are `null`.
2. All of its non-null children are heaps.

Write a method `isHeap` that returns a `true` if its argument is a heap and `false` if it is not:

```
public static boolean isHeap(Node node) {
    ...
}
```

You can treat `null` arguments however you like.
Question 4

Implement a class that represents an amount of US money in dollars and cents. It should support these operations:

```java
public class Money {
    public Money(int dollars, int cents)
        throws IllegalArgumentException;
    public int getDollars();
    public int getCents();
    @Override public String toString();
    @Override public boolean equals(Object that);
    public static Money add(Money x, Money y);
}
```

The amount should be considered negative if the dollars constructor argument is negative. The cents argument should always be positive; if not, it should throw an IllegalArgumentException. add should return a newly-allocated Money representing the sum of the two arguments. For example, the following code:

```java
Money z = Money.add(new Money(5, 10), new Money(-3, 50));
System.out.println(z.toString());
```

should output “$1.60”. To prevent rounding errors, you should not use floating-point numbers in your solution.

Question 5

Write a static method reverseComparator that, given a Comparator, returns another Comparator that represents the opposite order. So, if cmp.compare(a, b) returns −1, then reverseComparator(cmp).compare(a, b) should return 1. (The logic here is pretty simple, but how do you write the type for this?)