Quiz Section 1: Specifications and Coverage

Task 1 – The Same Ol' Strong and Dance [12 pts]

We plan to provide the following method:

```
/** Returns the best menu item for the given price.
 * ...
 */
public static MenuItem findBest(int price, MenuItem[] items);
```

To do so, we need to fill in the rest of the specification. For example, we need to explain exactly which item will be returned. The term "best" is far too vague.

We are considering the following alternatives:

```
@requires price > 0 and items is not null                          // Spec A
@return an item T in items with T.price <= price <= T.price + 0.5
    or null if none exists

@requires items is not null                                        // Spec B
@throws IllegalArgumentException if price <= 0
@return an item T in items with T.price <= price <= T.price + 0.5
    or null if none exists

@requires price > 0 and items is not null                          // Spec C
@return the item in items whose price is closest to the given price
    but not more than the given price or null if none exists

@requires price > 0                                                // Spec D
@throws NullPointerException if items is null
@return the item in items whose price is closest to the given price
    but not more than the given price or null if none exists

@requires price > 0                                                // Spec E
@return an item T in items with T.price <= price <= T.price + 0.5
    or null if none exists or if items is null
```

a) Fill in the following table explaining the relationships between each pair of specifications. Write an "S" if the spec on left (the row) is stronger than the spec on top (the column), a "W" if the left spec is weaker, and "—" if the specs are incomparable.

	A	**B**	**C**	**D**	**E**
A	X				
B		X			
C			X		
D				X	
E					X

b) Now consider a new pair of precondition and postcondition behaviors:

```
@requires price > 0
@return an item T in items with T.price <= price <= T.price + 0.5
```

This specification is not sensible, what is wrong with it? Why shouldn't we use it?

Task 2 – Great Finds Think Alike [12 pts]

For each of the following implementations, state which of the specifications it satisfies. If it does not satisfy some specification, explain (in as few words as possible) why it does not.

a)
```
public static MenuItem findBest(int price, MenuItem[] items) {
  if (price <= 0)
    throw new IllegalArgumentException("bad price");
  for (int i = 0; i < items.length; i++) {
    if (items[i].price <= price && price <= items[i].price + 0.5)
      return items[i];
  }
  return null;
}
```

b)
```
public static MenuItem findBest(int price, MenuItem[] items) {
  items.sort();  // puts items in order by increasing price
  MenuItem best = null;
  for (int i = 0; i < items.length; i++) {
    if (items[i].price <= price)
      best = items[i];
  }
  return best;
}
```

c)
```
public static MenuItem findBest(int price, MenuItem[] items) {
  if (items == null)
    return null;
  if (price <= 0)
    throw new IllegalArgumentException("bad price");
  for (int i = 0; i < items.length; i++) {
    if (items[i].price <= price && price <= items[i].price + 0.5)
      return items[i];
  }
  return null;
}
```

Task 3 – The Test-Laid Plans [12 pts]

For each of the following functions, state the number of tests required to meet our coverage requirements and explain why that is the required number.

Then, describe a specific set of tests to use (with the same number of tests you as said before). Describe each test by giving the input (identify a specific input rather than saying, e.g., "some positive number"), saying what portion of the function it tests, and explaining why our rules require that test.

a)

```
public static int f(int n) {
  if (n < 0) {
    return -2 * n;
  } else {
    return 3 * n;
  }
}
```

b)

```
public static int h(int n) {
  if (n <= 0) {
    return 1;
  } else {
    return 2 + h(n / 3);
  }
}
```