3. This question involves reasoning about the code from the Large Integer case study. A copy of the code is provided as part of this examination.

   (a) Write a new `BigInt` member function `Div2`, as started below. `Div2` should change the value of the `BigInt` to be the original value divided by 2 (integer division). Assume the `BigInt` is greater than or equal to 0. One algorithm for implementing `Div2` is:

   1. Initialize a variable carryDown to 0.

   2. For each digit, d, starting with the most significant digit,

      2.1 replace that digit with (d / 2) + carryDown

      2.2 let carryDown be (d % 2) * 5

   3. Normalize the result

   Complete member function `Div2` below.

```
void BigInt::Div2()
// precondition:  BigInt ≥ 0
```

(b) Write function `DivPos`, as started below. `DivPos` returns the quotient of the integer division of `dividend` by `divisor`. Assume that `dividend` and `divisor` are both positive values of type `BigInt`.

For example, assume that `bigNum1` and `bigNum2` are positive values of type `BigInt`:

`bigNum1`	`bigNum2`	`DivPos(bigNum1, bigNum2)`
18	9	2
17	2	8
8714	2178	4
9990	999	10

There are many ways to implement division; however, you must use a binary search algorithm to find the quotient of `dividend` divided by `divisor` in this problem. You will receive no credit on this part if you do not use a binary search algorithm.

One algorithm for implementing division using binary search is as follows:

1. Initialize `low` to `0` and `high` to `dividend`.
2. For each iteration,

   2.1 compute `mid = (low + high + 1)`

   2.2 divide `mid` by `2`

   2.3 if `mid * divisor` is larger than `dividend` (`mid` is too large to be the quotient) then set `high` equal to `mid - 1` else set `low` equal to `mid`.

3. When `low == high` the search terminates, and you should return `low`.

In writing function `DivPos`, you may call function `Div2` specified in part (a). Assume that `Div2` works as specified, regardless of what you wrote in part (a). You will receive no credit on this part if you do not use a binary search algorithm.

Complete function `DivPos` below. Assume that `DivPos` is called only with parameters that satisfy its precondition.

```
BigInt DivPos(const BigInt & dividend, const BigInt & divisor)
// precondition: dividend > 0, divisor > 0
```