AP Computer Science A
1999 Sample Student Responses

The materials included in these files are intended for non-commercial use by AP teachers for course and exam preparation; permission for any other use must be sought from the Advanced Placement Program. Teachers may reproduce them, in whole or in part, in limited quantities, for face-to-face teaching purposes but may not mass distribute the materials, electronically or otherwise. These materials and any copies made of them may not be resold, and the copyright notices must be retained as they appear here. This permission does not apply to any third-party copyrights contained herein.
2. 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.

```c
void BigInt::Div2()
// precondition:  BigInt ≥ 0
{
    int carryDown = 0, d;
    for (int i = myNumDigits - 1; i >= 0; i--)
        d = GetDigit(i);
        ChangeDigit(i, (d/2) + carryDown);
        carryDown = (d%2) * 5;
    Normalize();
}
```

GO ON TO THE NEXT PAGE
(b) Write a definition to overload the / operator, as started below. 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:

\[
\begin{array}{ccc}
\text{bigNum1} & \text{bigNum2} & \text{bigNum1 / bigNum2} \\
18 & 9 & 2 \\
17 & 2 & 8 \\
8714 & 2178 & 4 \\
9990 & 999 & 10 \\
\end{array}
\]

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

Let low and high represent a range in which the quotient is found.

Initialize low to 0 and high to dividend.

For each iteration, compute \( \text{mid} = (\text{low} + \text{high} + 1) \), divide \( \text{mid} \) by 2, and compare \( \text{mid} \times \text{divisor} \) with \( \text{dividend} \) to maintain the invariant that \( \text{low} \leq \text{quotient} \) and \( \text{high} \geq \text{quotient} \).

When \( \text{low} == \text{high} \), the quotient has been found.

In writing function \( \text{operator/} \) you may call function \( \text{Div2} \) specified in part (a). Assume that \( \text{Div2} \) 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 \( \text{operator/} \) below. Assume that \( \text{operator/} \) is called only with parameters that satisfy its precondition.

\[
\begin{align*}
\text{BigInt operator/ (const BigInt & dividend, const BigInt & divisor) } \\
\quad \text{// precondition: dividend > 0, divisor > 0} \\
\quad \text{BigInt low (0)} > \text{high (dividend), mid } \\
\quad \text{while (low != high)} \\
\quad \quad \text{mid = (low + high + 1) } > \\
\quad \quad \text{mid.Div2 ()} > \\
\quad \quad \text{if(mid*divisor > dividend) } \\
\quad \quad \quad \text{high = mid-1} > \\
\quad \quad \text{else } \\
\quad \quad \quad \text{low = mid} > \\
\quad \quad \text{return low } > \\
\end{align*}
\]

GO ON TO THE NEXT PAGE
2. 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 \( (d / 2) + \text{carryDown} \)
   
   2.2 let \text{carryDown} be \( (d \% 2) \times 5 \)

3. Normalize the result.

Complete member function Div2 below.

```cpp
void BigInt::Div2()
// precondition: BigInt >= 0
{
    int carryDown(0);
    int d;
    for (int i = 0; i < NumDigits(); i++) {
        d = GetDigit(i);
        ChangeDigit(i, (d/2) + carryDown);
        carryDown = (d % 2) * 5;
    }
    Normalize();
}
```
(b) Write a definition to overload the / operator, as started below. 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:

<table>
<thead>
<tr>
<th>bigNum1</th>
<th>bigNum2</th>
<th>bigNum1 / bigNum2</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>8714</td>
<td>2178</td>
<td>4</td>
</tr>
<tr>
<td>9990</td>
<td>999</td>
<td>10</td>
</tr>
</tbody>
</table>

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

Let low and high represent a range in which the quotient is found.

Initialize low to 0 and high to dividend.

For each iteration, compute mid = (low + high + 1) / 2, divide mid by 2, and compare

\[ \text{mid} \times \text{divisor} \text{ with dividend to maintain the invariant that low} \leq \text{quotient and high} \geq \text{quotient.} \]

When low == high, the quotient has been found.

In writing function operator/ 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 operator/ below. Assume that operator/ is called only with parameters that satisfy its precondition.

```
BigInt operator/ (const BigInt & dividend, const BigInt & divisor)
// precondition: dividend > 0, divisor > 0
{
    BigInt low(0); high(dividend);
    BigInt mid, quotient(1);
    while (low != high) {
        mid = (low + high + 1) / 2;
        mid = Div2();
        quotient = mid + divisor;
        if (dividend > quotient) low = mid;
        if (dividend < quotient) high = mid;
    }
    return low;
}
```
2. 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.

```cpp
void BigInt::Div2()
// precondition: BigInt \geq 0
{
    int carryDown = 0;
    for (int i = 0; i < numDigits; i++)
    {
        ChangeDigit(i, (d / 2) + carryDown)
        carryDown = (d % 2) * 5
    }
    Normalize();
}
```

(b) Write a definition to overload the `/` operator, as started below. 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`:

<table>
<thead>
<tr>
<th><code>bigNum1</code></th>
<th><code>bigNum2</code></th>
<th><code>bigNum1 / bigNum2</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>8714</td>
<td>2178</td>
<td>4</td>
</tr>
<tr>
<td>9990</td>
<td>999</td>
<td>10</td>
</tr>
</tbody>
</table>

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

Let `low` and `high` represent a range in which the quotient is found.

Initialize `low` to 0 and `high` to `dividend`.

For each iteration, compute `mid = (low + high + 1) / 2`, divide `mid` by 2, and compare

`mid * divisor` with `dividend` to maintain the invariant that `low <= quotient` and `high >= quotient`.

When `low == high`, the quotient has been found.

In writing function `operator/` 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 `operator/` below. Assume that `operator/` is called only with parameters that satisfy its precondition.

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

uint64_t high = dividend;
while (low <= quotient && high >= quotient)

mid = (low + high + 1) / 2;
mid = Div2();
if (low == high)
    break;
BigInt quot(mid);
return quot;
```

GO ON TO THE NEXT PAGE