INFSCI 2591: Algorithm Design
Assignment #1
Due: September 13, 2016

1. Write a pseudocode for multiplying large numbers using the ala carte multiplication method.

Problem: multiply two large numbers (*x* and *y*) using the rectangle Ala Carte method.
Input: two integers *x*, *y*
Output: *result*, the product of the two numbers (*x* and *y*).

```
int ala_carte_multi(int x, int y)
{
   int result = 0;

   // get rid of the negative sign,
   // and change the corresponding flag to add them later.
   bool is_x_neg = false;
   bool is_y_neg = false;
   if(x < 0)
   {
      x_neg = true;
      x = x * -1;
   }
   if(y < 0)
   {
      y_neg = true;
      y = y * -1;
   }

   while(x > 0)
   {
      if(x % 2 != 0)
      {
         result += y;
      }
      x = floor(x / 2);
      y = y * 2;
   }

   // make sure the negative sign exists if needed
   if((is_x_neg && !is_y_neg) || (!is_x_neg && is_y_neg))
   {
      result = result * -1;
   }

   return result;
}
```

2. Write a pseudocode for multiplying large numbers using the rectangle multiplication method.

Problem: multiply two large numbers (*x* and *y*) using the rectangle multiplication method.
Input: two integers *x*, *y*
Output: *result*, the product of the two numbers (*x* and *y*).

```
int rectangle_multi (int x, int y)
{
   int result = 0;
   // get rid of the negative sign,
   // and change the corresponding flag to add them later.
   bool is_x_neg = false;
   bool is_y_neg = false;

   if(x < 0)
   {
      is_x_neg = true;
      x = x * -1;
   }
   if(y < 0)
   {
       is_y_neg = true;
       y = y * -1;
   }

   index i, j;
   int x_len = digits count of (x);
   int y_len = digits count of (y);
   int x_digits[1..x_len];
   int y_digits[1..y_len];

   // split x into digits
   for(i = x_len; i >= 1; i--)
   {
      x_digits[i] = x % 10;
      x /= 10;
   }
   // split y into digits
   for(i = y_len; i >= 1; i--)
   {
      y_digits[i] = y % 10;
      y /= 10;
   }

   // multiplying each two-corresponding digits
   // and storing the result in a table (2D array)
   int table_mult[1..y_len][1..(x_len * 2)];

   for(i = 1; i <= y_len; i++)
      for(j = 1; j <= x_len; j++)
      {
         int tmp_mult = y_digits[i] * x_digits[j];
         table_mult[i][(j*2)] = tmp_mult / 10;
         table_mult[i][(j*2)+1] = tmp_mult % 10;
      }
```

```
// summing up and store the result in array.
int result_arr_len = (x_len + y_len);
int result_arr[1..result_arr_len];
int sum;
index row, col;
bool iteration;

// first loop (zigzag traversal) in the sum operation
for(i = y_len; i >= 1; i--)
{
   sum = 0;
   row = i;
   col = (x_len * 2);
   iteration = false;

   while(row <= y_len && col >= 0)
   {
      sum += table_mult[row][col];

      if(!iteration)
         row++;
      col--;
      iteration = !iteration;
   }

   result_arr[(x_len + i)] = sum;
}

// second loop (zigzag traversal) in the sum operation
for(j = x_len; j >= 1; j--) {

   sum = 0;
   col = (j * 2);
   row = 0;
   iteration = true;

   while(row <= y_len && col >= 0)
   {
      sum += table_mult[row][col];

      if(!iteration)
         row++;
      col--;
      iteration = !iteration;
   }

   result_arr[j] = sum;
}

// summing up the carryout/leftover with the next number.
for(i = result_arr_len; i >= 1; i--)
{
   if(result_arr[i] > 10)
   {
      result_arr[i-1] += (result_arr[i] / 10);
      result_arr[i] = result_arr[i] % 10;
   }
}
```

```
   // preparing the numbers to final stage
   // by giving them its actual value (weight).
   for(i = 1; i <= result_arr_len; i++)
      for(j = i; j <= (result_arr_len - 1); j++)
         result_arr[i] *= 10;

   // sum all items inside the result array
   for(i = 1; i <= result_arr_len; i++)
      result += result_arr[i];

   // make sure the negative sign exists if needed
   if((is_x_neg && !is_y_neg) || (!is_x_neg && is_y_neg))
   {
       result = result * -1;
   }

   return result;
}
```