Algorithm 1 Karatsuba Algorithm

```
Input: $a, b$ - two numbers represent as strings
Input: $n$ - length of the two strings
Output: $product$ - the product of two numbers
procedure Karatsuba($a$, $b$)
    PadZeroes(a, b)                                    ▷ make sure strings are of equal length
    if a and b are single digit numbers then           ▷ base case
        return a * b
    end if
    $al \leftarrow a[0, n/2)$                           ▷ left of a
    $ar \leftarrow a[n/2, n)$                           ▷ right of a
    $bl \leftarrow b[0, n/2)$                           ▷ left of b
    $br \leftarrow b[n/2, n)$                           ▷ right of b
    $p1 \leftarrow Karatsuba(al, bl)$
    $p2 \leftarrow Karatsuba(al + ar, bl + br)$
    $p3 \leftarrow Karatsuba(al, bl)$
    return $(10^n * p1) + 10^{n/2} * (p2 - p1 - p3) + p3$
end procedure
```

Algorithm 2 Graphics

```
Input: $x, y$ - arrays representing points on x and y axis
Input: $n$ - the number of $x$ and $y$ points given
Input: $p$ - point given
Output: how many lines OP intersects
procedure Graphics($x$, $y$)
    QuickSort($x$)
    QuickSort($y$)
    for i = 0 to n - 1 do
        if $(0 - x[i]) * (p.y - 0) - (y[i] - 0) * (p.x - x[i]) > 0$ then    ▷ ToLeft algorithm
            return $i$
        end if
    end for
    return n - 1;                                      ▷ points intersect all numbers
end procedure
```

Algorithm 3 Long Multiplication

Input: a, b - two numbers represented as strings
Input: m, n - the length of two strings
Output: the product of the two numbers

```
procedure LongMult($a$, $b$)
    Prod ← array[m + n]                              ▷ result is at most size m + n
    Carry ← 0                                        ▷ keep track of carry
    if $n < m$ then Swap($a, b$)                     ▷ smaller number on bottom
    end if
    for i = m - 1 to 0 do                            ▷ multiply starting from end of smaller number
        for j = n - 1 to 0 do                        ▷ multiply digit by digit of larger number
            TempProd ← Prod[i + j + 1] + (a[i] * a[j])   ▷ store ones digit
            Prod[i + j + 1] ← TempSum mod 10;
            Prod[i + j] ← TempSum / 10               ▷ store sum carry in previous element
        end for
    end for
    return RemoveLeadingZero(Sum)                    ▷ Remove leading zero
end procedure
```