In [ ]:
```
(a) Pseudo recursive code
m = 2
DIVIDE2(v, low, high):
    if low == high:
        return
    mid = floor((low + high) / 2)
    DIVIDE2(v, low, mid)
    DIVIDE2(v, mid + 1, high)
```

In [ ]:
```
general m
DIVIDEM(v, low, high, m):
    if low == high:
        return
    n = high - low + 1
    for i = 0 to m-1:
        start = low + floor(i * n / m)
        end   = low + floor((i+1) * n / m) - 1
        if start <= end:
            DIVIDEM(v, start, end, m)
```

In [ ]:
```
(b) Complexity analysis
For full decomposition:
m = 2: T(n)=2T(n/2)+O(1)
m = 3: T(n)=3T(n/3)+O(1)
general m: T(n)=mT(n/m)+O(1)
Master theorem:
a = m, b = m
n^(log_b a) = n
Therefore:
T(n) = Theta(n)
```

In [ ]:
```
(c) Collect the unique 1

After full decomposition there are n subproblems of size 1.

Cost of collection:
f(n) = Theta(n)

Binary recurrence:
T(n) = 2T(n/2) + Theta(n)

Since a = 2, b = 2 and f(n) = Theta(n),
T(n) = Theta(n log n)
```

In [ ]:
```
(1) Multiplication in base 10 (school method)

Represent:
x = sum of X[i] * 10^i
y = sum of Y[j] * 10^j
```

In [2]:
```
def school_multiply(a: str, b: str) -> str:
    X = [int(d) for d in a][::-1]
    Y = [int(d) for d in b][::-1]
    res = [0] * (len(X) + len(Y))

    for i in range(len(X)):
        carry = 0
```

```python
        for j in range(len(Y)):
            t = X[i] * Y[j] + res[i + j] + carry
            res[i + j] = t % 10
            carry = t // 10
        res[i + len(Y)] += carry

    while len(res) > 1 and res[-1] == 0:
        res.pop()
    return "".join(map(str, res[::-1]))
```

In [ ]:
```
(2) Large integers

The algorithm uses digit arrays only and does not rely on built-in integer limit
```

In [ ]:
```
(3) Divide and conquer vs Karatsuba

Split:
x = x1 * 10^(n/2) + x0
y = y1 * 10^(n/2) + y0

Standard divide and conquer:
T(n) = 4T(n/2) + O(n)
=> T(n) = Theta(n^2)

Karatsuba:
T(n) = 3T(n/2) + O(n)
=> T(n) = Theta(n^(log_2 3))
```

In [ ]:
```
(4) Sum from 1 to n using one multiplication

Sum = n(n + 1) / 2

Let:
v = n
w = n + 1

Compute one school multiplication n(n + 1),
then divide the result by 2.
```