Karatsuba Algorithm

Integer Multiplication

Sasank Tadepalli, 10/24/2022

Agenda

- Naive Integer Multiplication
  - Technique
  - Pseudo Code
  - Time Complexity
- Karatsuba Algorithm
  - Technique
  - Pseudo Code
  - Time Complexity
- Summary

Naive Integer Multiplication

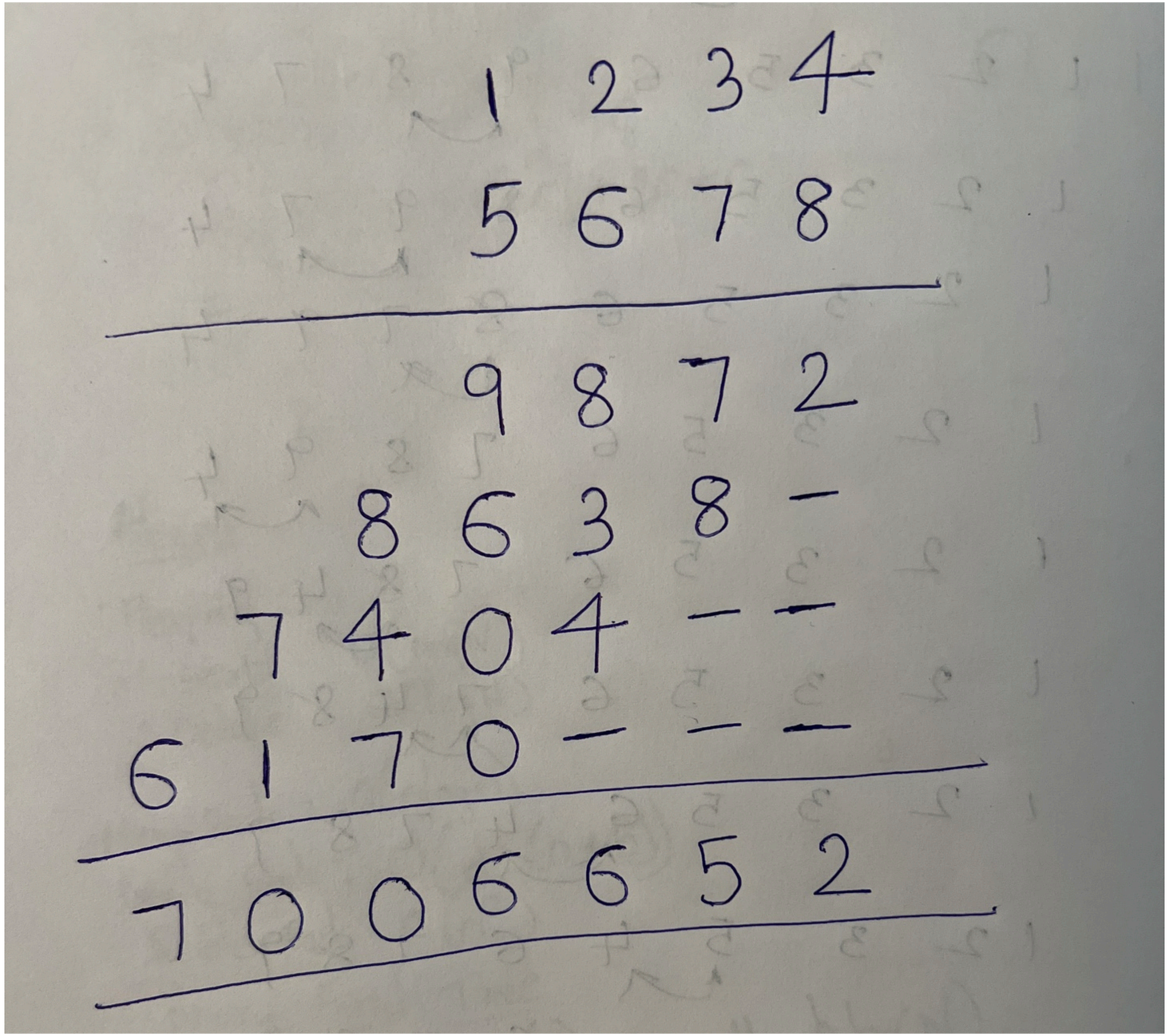
Technique

- The naive way to multiply two n-digit numbers is commonly taught in elementary schools.

- The naive multiplication algorithm to multiply two n-digit numbers x, y is to multiply each digit of first number with each digit of second number by shifting the result of each multiplication by (k-1) digits to left where k is the position of digit from right end of number and adding all those intermediate results.

- Let’s look at an example:

  X = 1234 and Y = 5678


1 2 3 4
5 6 7 8
9 8 7 2
8 6 3 8 –
7 4 0 4 – –
6 1 7 0 – – –
7 0 0 6 6 5 2

Pseudo Code

```
def mutliply(x, y):
    finalProduct = 0
    shift = 1
    yCopy = y
    while x != 0:
        tempProduct = 0
        multiplier = 1
        carry = 0
        y = yCopy
        xLast = x % 10
        x = x // 10
```

Pseudo Code

```
        while y != 0:
                yLast = y % 10
                product = xLast * yLast
                product += carry
                carry = product // 10
                product = product % 10
                tempProduct += product * multiplier
                multiplier *= 10
                y = y // 10
        tempProduct += carry * multiplier
        finalProduct += tempProduct * shift
        shift *= 10
    return finalProduct
```

Time Complexity Analysis

- We perform n multiplications for each digit of x. Since x has n digits, in total we perform n^2 multiplications.

- Therefore Time Complexity of this algorithm is $O(n^2)$

Can we solve it using Divide & Conquer?

Yes.

Can you guess the logic to follow?

Let’s understand the logic…

Divide & Conquer Technique

- Let x and y be two n-bit numbers.
- Divide each number into two halves, the high bits H and low bits L
    - $x = xH * 10^{(n/2)} + xL$
    - $y = yH * 10^{(n/2)} + yL$
- Now, $x * y = (xH * 10^{(n/2)} + xL) * (yH * 10^{(n/2)} + yL)$

$$x * y = xH * yH * 10^{n} + (xH * yL + xL * yH) * 10^{(n/2)} + xL * yL$$

- Can anyone guess how many subproblems are there in the given equation?

It’s **4**

Divide & Conquer Technique

- The Subproblems are
    - xH * yH
    - xH * yL
    - xL * yH
    - xL * yL
- So, we have 4 recursive calls in our logic and at each call we reduce n size by half.
- What other things do we need to form our recurrence relation?
    - Base Case
    - Number of basic operations we perform at each call

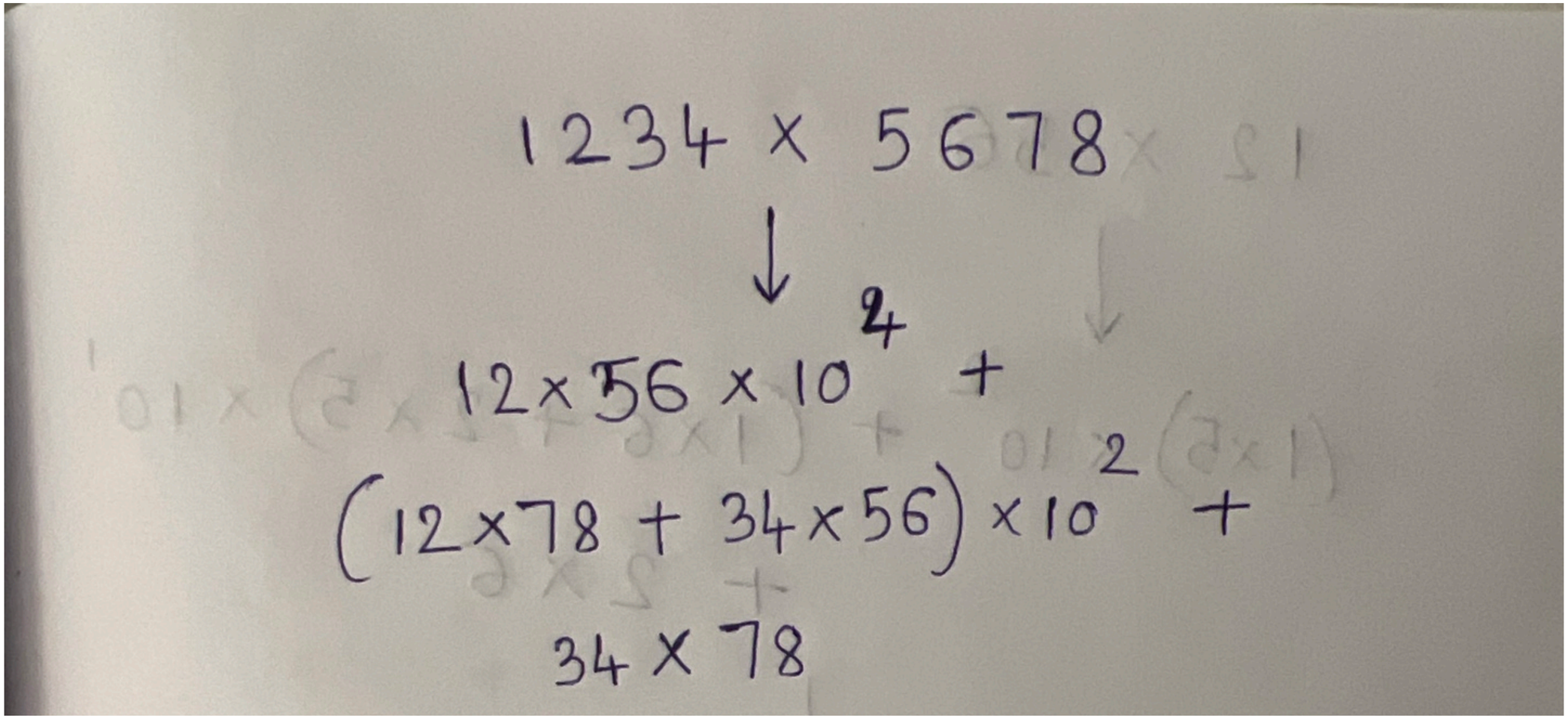
Divide & Conquer Technique

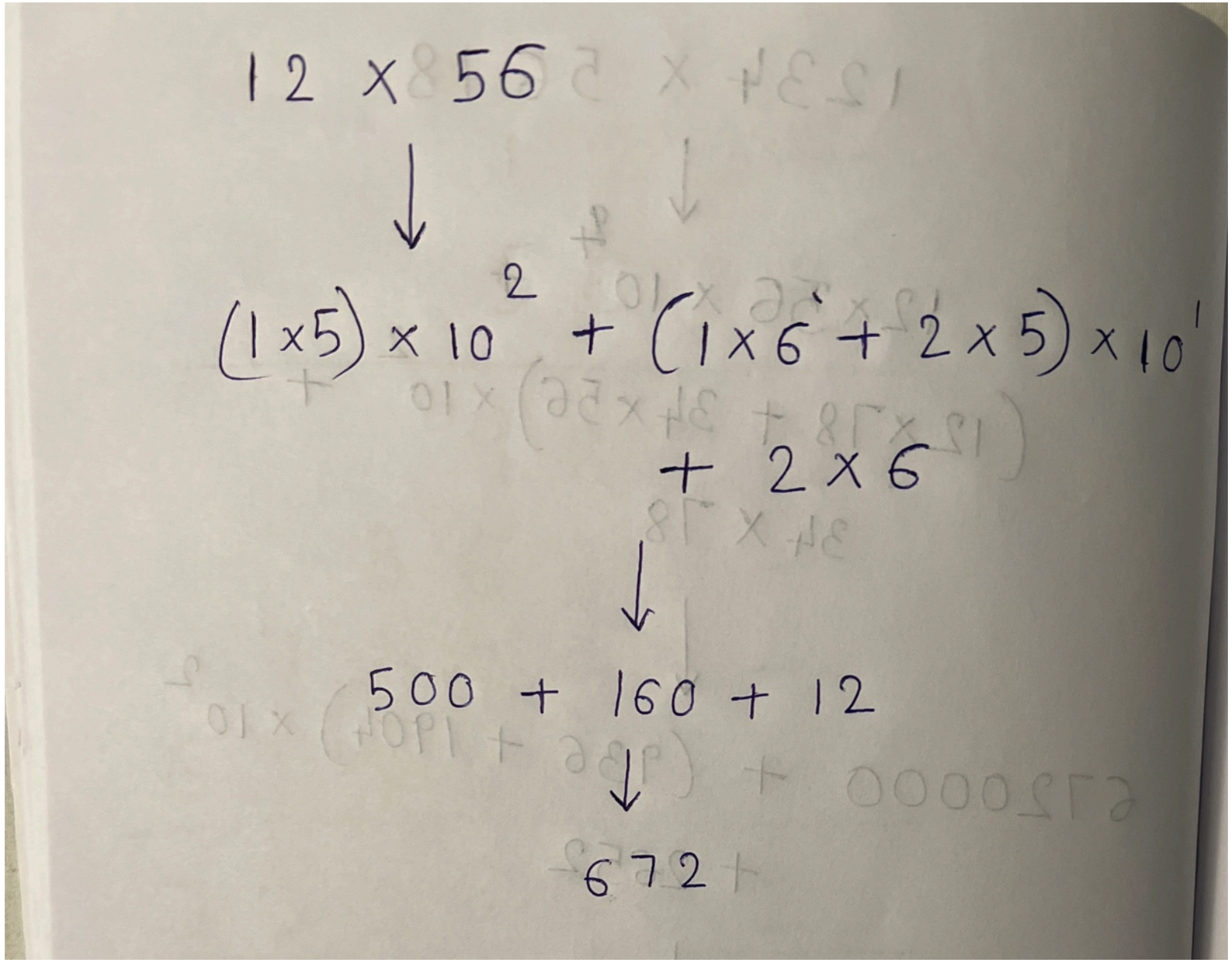
- Base Case?
  - We reach base case when both numbers have single digit.
  - At base case we return that product which takes O(1) time.
- Number of basic operations?
  - === Number of additions we perform in terms of n.
  - Let’s find the number of additions we perform in
    - $xH * yH * 10^{n} + (xH * yL + xL * yH) * 10^{(n/2)} + xL * yL$

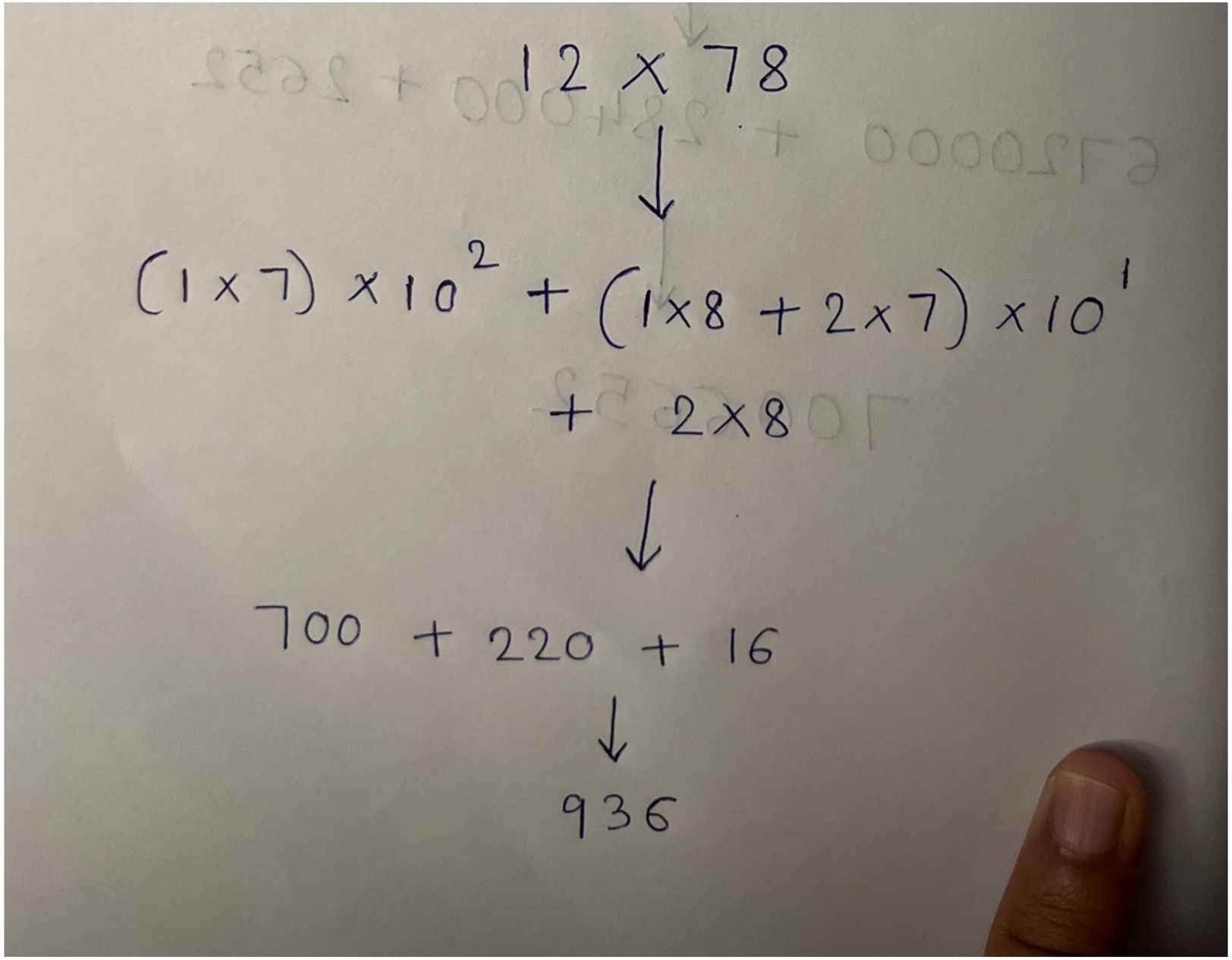
Divide & Conquer Technique

- Number of basic operations?
  - xH, yH, xL, yL have n / 2 digits
  - How many digits we get if we multiply any 2 of them?
  - We get a maximum of n digits.
  - So, In our equation
    - xH * yH, xH * yL, xL * yH, xL * yL all have a maximum of n digits
  - To add two n digit numbers, we perform n additions.
  - In, $xH * yH * 10^{n} + (xH * yL + xL * yH) * 10^{(n/2)} + xL * yL$ we have 3 such additions. So, we perform O(n) additions.
- Now, the recurrence relation is
  - $T(n) = 4 * T(n / 2) + O(n)$ and $T(1) = O(1)$

Let’s look at an example:




12 x 56
(1x5) x 10^2 + (1x6 + 2x5) x 10^1
+ 2x6
500 + 160 + 12
672


12 x 78
(1x7) x10^2 + (1x8 + 2x7) x10^1
+ 2x8
700 + 220 + 16
936

$$34 \times 56$$

$$\downarrow$$

$$(3 \times 5) \times 10^{2} + (3 \times 6 + 4 \times 5) \times 10^{1} + 4 \times 6$$

$$\downarrow$$

$$1500 + 380 + 24$$

$$\downarrow$$

$$1904$$

$$34 \times 78$$

$$\downarrow$$

$$(3 \times 7) \times 10^{2} + (3 \times 8 + 4 \times 7) \times 10^{1} + 4 \times 8$$

$$\downarrow$$

$$2100 + 520 + 32$$

$$\downarrow$$

$$2652$$

$$1234 \times 5678$$

$$\downarrow$$

$$12 \times 56 \times 10^{4} +$$

$$(12 \times 78 + 34 \times 56) \times 10^{2} +$$

$$34 \times 78$$

$$\downarrow$$

$$6720000 + (936 + 1904) \times 10^{2}$$

$$+ 2652$$

$$\downarrow$$

$$6720000 + 284000 + 2652$$

$$\downarrow$$

$$7006652$$

What will be the n value when x and y have different number of digits?

n will be maximum of number of digits of x and y.

How will we divide x or y if they have odd number of digits?

We will take ceil(n/2) i.e; we round n/2 to the first number which is > n/2

Pseudo Code

```
from math import ceil, floor

#math.ceil(x) Return the ceiling of x as a float, the smallest integer value greater than or equal to x.

#math.floor(x) Return the floor of x as a float, the largest integer value less than or equal to x.

def naiveRecursiveLogic(x,y):
    #base case
    if x < 10 and y < 10: # in other words, if x and y are single digits
        return x*y
    n = max(len(str(x)), len(str(y)))
    m = ceil(n/2)   #Cast n into a float because n might lie outside the representable range of integers.
    x_H  = floor(x / 10**m)
    x_L = x % (10**m)
```

Pseudo Code

```
    y_H = floor(y / 10**m)

    y_L = y % (10**m)


    #recursive steps

    a = naiveRecursiveLogic(x_H,y_H)

    b = naiveRecursiveLogic(x_H,y_L)

    c = naiveRecursiveLogic(x_L,y_H)

    d = naiveRecursiveLogic(x_L,y_L)


    return int(a*(10**(m*2)) + (b + c) *(10**m) + d)


print(naiveRecursiveLogic(1234, 5678))
```

Time Complexity Analysis

- The recurrence relation is
  - $T(n) = 4 * T(n / 2) + O(n)$ and $T(1) = O(1)$
- Using Master's theorem, can anyone solve and say the asymptotic notation of the given recurrence relation?
- $a = 4$, $b = 2$ and $f(n) = O(n)$
- $\log_b a = \log_2 4 = 2$
- $f(n) = O(n) = O(n^1)$ ===> $d = 1$
- Since $d < \log_b a$ the given recurrence relation follows Case-I of Master's Theorem
- According to Case-I of Master's Theorem, if $d < \log_b a$, the given function grows slower than $n^{\log_b a}$. Therefore, $T(n) = \text{Theta}(n^{\log_b a})$
- So, $T(n) = \text{Theta}(n^2)$

We didn’t get any improvement from the previous approach we learned.
Then what is the advantage of this algorithm?

We learned this recursive algorithm of solving integer multiplication so that we can understand the significance of Karatsuba’s Algorithm in a clear way.

Let’s understand Karatsuba’s Algorithm now…

Karatsuba’s Algorithm

Introduction

- The Karatsuba algorithm is a fast multiplication algorithm that uses a divide and conquer approach to multiply two numbers.
- Being able to multiply numbers quickly is very important since Computer scientists often consider multiplication to be a constant time O(1) operation.
- But for a large numbers the actual time complexity using a naive algorithm is $O(n^2)$.
- The Karatsuba algorithm decreases the number of subproblems to three as opposed to four which we saw earlier.
- Let’s see the logic…

Technique

- As we saw earlier in our divide and conquer approach,
  - $x * y = xH * yH * 10^{n} + (xH * yL + xL * yH) * 10^{(n/2)} + xL * yL$
- Karatsuba reduced number of subproblems from 4 to 3 by changing the middle part in the above equation i.e; (xH * yL + xL * yH)
- Let a = xH * yH

  d = xL * yL
- Then we can rewrite (xH * yL + xL * yH) as
  - (xH + xL) * (yH + yL) – a – d

Technique

- Let e = (xH + xL) * (yH + yL) – a – d
- Now,
    - $x * y = a * 10^{n} + e * 10^{(n/2)} + d$
- Can anyone guess how many subproblems are there in the given equation?

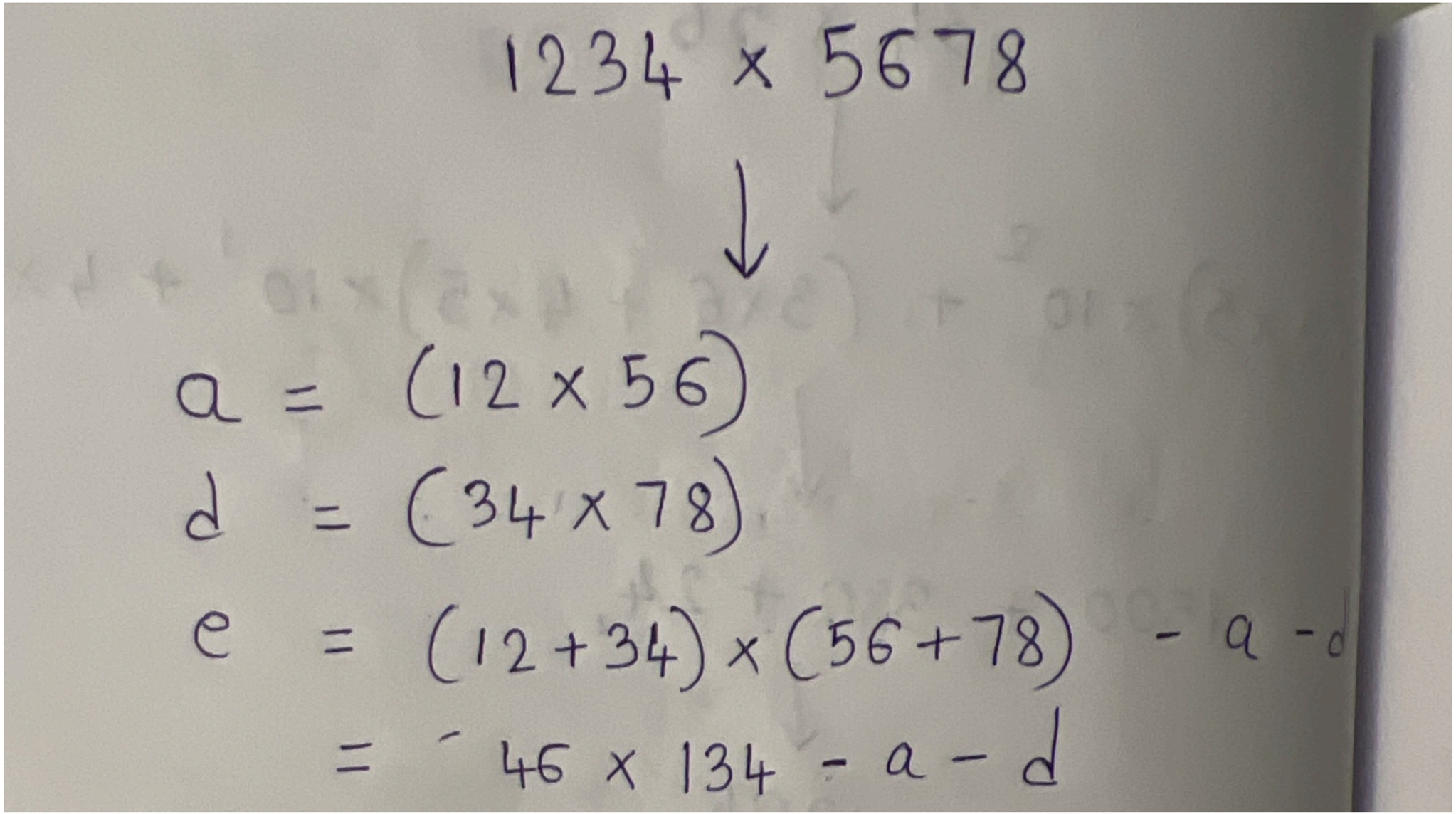
It’s **3**

- So, we have 3 recursive calls in our logic and at each call we reduce n size by half.

Technique

- Base Case is when both x and y has a single digit.
- Number of additions change from 3 to 6 with each addition having O(n) Time Complexity same as before.
- So, the recurrence relation will be
  - $T(n) = 3 * T(n / 2) + O(n)$ and $T(1) = O(1)$

Let’s look at an example:



$$12 \times 56$$

$$\downarrow$$

$$a = 1 \times 5$$

$$d = 2 \times 6$$

$$e = (1+2) \times (5+6) - a - d$$

$$= (3 \times 11) - a - d$$

3×11

↓

$a = (0 \times 1)$

$d = 3 \times 1$

$e = (a+3) \times (1 \pm 1) - a - d$

$= (3 \times 2) - a - d$

↓

$a = 0$

$d = 3$

$e = 3 \times 2 - 0 - 3 = 3$

↓

$0 \times 10^2 + 3 \times 10^1 + 3$

↓

33

12×56

↓

$a = 1 \times 5$

$d = 2 \times 6$

$e = (1+2) \times (5+6) - a - d$

$= (3 \times 11) - a - d$

↓

$a = 5$

$d = 12$

$e = 33 - 5 - 12$

$= 16$

↓

$5 \times 10^{2} + 16 \times 10^{1} + 12$

↓

672

$$34 \times 78$$

$$\downarrow$$

$$a = 3 \times 7$$

$$d = 4 \times 8$$

$$e = (3+4) \times (7+8) - a - d$$

$$= (7 \times 15) - a - d$$

7×15

↓

$a = 0 \times 1$

$d = 7 \times 5$

$e = (0+7) \times (1+5) - a - d$

$= 7 \times 6 - a - d$

↓

$a = 0$

$d = 35$

$e = 42 - 0 - 35$

$= 7$

↓

$0 \times 10^2 + 7 \times 10^1 + 35$

↓

105

34×78

↓

$a = 3 \times 7$

$d = 4 \times 8$

$e = (3+4) \times (7+8) - a - d$

$= (7 \times 15) - a - d$

↓

$a = 21$

$d = 32$

$e = 105 - 21 - 32$

$= 52$

↓

$21 \times 10^{2} + 52 \times 10^{1} + 32$

↓

2652

$$46 \times 134$$

$$\downarrow$$

$$a = 0 \times 1$$

$$d = 46 \times 34$$

$$e = (0+46) \times (1+34) - a - d$$

$$= 46 \times 35 - a - d$$

46×34

↓

$a = 4 \times 3$

$d = 6 \times 4$

$e = (4+6) \times (3+4) - a - d$

$= (10 \times 7) - a - d$

↓

$a = 12$

$d = 24$

$e = 70 - 12 - 24$

$= 34$

↓

$12 \times 10^2 + 34 \times 10^1 + 24$

↓

1564

10×7

↓

$a = 1 \times 0$

$d = 0 \times 7$

$e = (1+0) \times (0+7) - a - d$

$= 1 \times 7 - a - d$

↓

$a = 0$

$d = 0$

$e = 7 - 0 - 0 = 7$

↓

$0 \times 10^2 + 7 \times 10^1 + 0$

↓

70

46×35

↓

$a = 4 \times 3$

$d = 6 \times 5$

$e = (4+6) \times (3+5) - a - d$

$= 10 \times 8 - a - d$

↓

$a = 12$

$d = 30$

$e = 80 - 12 - 30$

$= 38$

↓

$12 \times 10^2 + 38 \times 10^1 + 30$

↓

1610

10×8

↓

$a = 1 \times 0$

$d = 0 \times 8$

$e = (1+0) \times (0+8) - a - d$

$= 1 \times 8 - a - d$

↓

$a = 0$

$d = 0$

$e = 8 - 0 - 0 = 8$

↓

$0 \times 10^2 + 8 \times 10^1 + 0$

↓

80

46×134

↓

$a = 0 \times 1$

$d = 46 \times 34$

$e = (0+46) \times (1+34) - a - d$

$= 46 \times 35 - a - d$

↓

$a = 0$

$d = 1564$

$e = 1610 - 0 - 1564 = 46$

↓

$0 \times 10^4 + 46 \times 10^2 + 1564$

↓

6164

1234×5678

↓

$a = (12 \times 56)$

$d = (34 \times 78)$

$e = (12+34) \times (56+78) - a - d$

$= 46 \times 134 - a - d$

↓

$a = 672$

$d = 2652$

$e = 6164 - 672 - 2652$

$= 2840$

↓

$672 \times 10^4 + 2840 \times 10^2 + 2652$

↓

7006652

Pseudo Code

```
from math import ceil, floor

#math.ceil(x) Return the ceiling of x as a float, the smallest integer value greater than or equal to x.

#math.floor(x) Return the floor of x as a float, the largest integer value less than or equal to x.

def karatsuba(x,y):
    #base case
    if x < 10 and y < 10: # in other words, if x and y are single digits
        return x*y
    n = max(len(str(x)), len(str(y)))
    m = ceil(n/2)   #Cast n into a float because n might lie outside the representable range of integers.
    x_H  = floor(x / 10**m)
    x_L = x % (10**m)
```

Pseudo Code

```
    y_H = floor(y / 10**m)

    y_L = y % (10**m)


    #recursive steps

    a = karatsuba(x_H,y_H)

    d = karatsuba(x_L,y_L)

    e = karatsuba(x_H + x_L, y_H + y_L) - a - d


    return int(a*(10**(m*2)) + e*(10**m) + d)


print(karatsuba(12345, 45678))
```

Time Complexity Analysis

- The recurrence relation is
  - $T(n) = 3 * T(n / 2) + O(n)$ and $T(1) = O(1)$
- $a = 3$, $b = 2$ and $f(n) = O(n)$
- $\log_b a = \log_2 3 = 1.585$
- $f(n) = O(n) = O(n^1) ===> d = 1$
- Since $d < \log_b a$ the given recurrence relation follows Case-I of Master's Theorem
- According to Case-I of Master's Theorem, if $d < \log_b a$, the given function grows slower than $n^{\log_b a}$. Therefore, $T(n) = Theta(n^{\log_b a})$
- So, $T(n) = Theta(n^{1.585})$

Summary

- Naive Multiplication Algorithm can be implemented in both iterative and recursive approach.
- Naive Multiplication algorithm has a time complexity of $O(n^2)$.
- Karatsuba Algorithm is a fast multiplication recursive algorithm that uses divide and conquer approach to multiply 2 integers.
- Karatsuba Algorithm has a time complexity of $O(n^{1.585})$.
- Karatsuba Algorithm can be used to multiply numbers in all base systems (base 10, base 2 etc..).
- There are some other faster algorithms as well like Toom-Cook algorithm which runs in $O(n^{1.465})$ and Schönhage–Strassen algorithm which runs in $O(n \log n \log \log n)$.

Any Questions?

Thank You