Theory:

This method multiplies two large integers using the divide and conquer approach. Instead of the traditional O(n²) multiplication, it splits each number into two halves and recursively computes three multiplications (ac, bd, and (a+b)(c+d)) to reduce the total number of multiplications.

- Faster than Classical Multiplication

Programming Language: Python

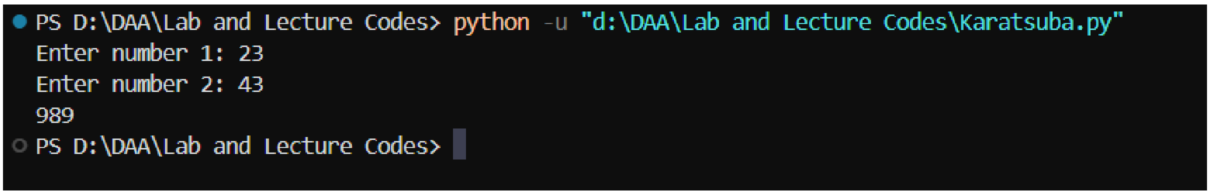
1) Karatsuba

Code:

```
def karatsuba(x,y):
    if x<10 or y<10:
        return x * y


    n = max(len(str(x)), len(str(y)))
    if n%2!=0:
        n -= 1


    a,b=divmod(x,10**(n//2))
    c,d=divmod(y,10**(n//2))


    ac = karatsuba(a, c)
    bd = karatsuba(b, d)
    abcd = karatsuba(a+b, c+d)-ac-bd
    return (ac*(10**n))+(abcd)*(10**(n//2))+bd
```

```
x = int(input("Enter number 1: "))

y = int(input("Enter number 2: "))

print(karatsuba(x,y))
```

Output:



```
● PS D:\DAA\Lab and Lecture Codes> python -u "d:\DAA\Lab and Lecture Codes\Karatsuba.py"
  Enter number 1: 23
  Enter number 2: 43
  989
○ PS D:\DAA\Lab and Lecture Codes> 
```

Space complexity: O(log n)

- **Justification:** Each recursive call requires additional stack frames and storage for intermediate results (like ac, bd, abcd). Since recursion depth is log n, the space required is O(log n).

Time Complexity

- **Best Case: O(1)**
  Justification: When both inputs are single-digit integers, the algorithm directly multiplies them without recursion.

- **Worst Case: O(n log n)**
  Justification: For large inputs, full recursive decomposition is required. At each level, three recursive multiplications and linear-time additions are performed, giving O(n log n).