**Aim:** Implementing Application-based Algorithm using the D&C Approach

**IDE:** Visual Studio Code

## Karatsuba Algorithm

### Theory: -

The Karatsuba algorithm is based on **divide and conquer**. It decomposes the multiplication of two large numbers into smaller, simpler multiplications by splitting each number into parts. Here's a step-by-step breakdown of how it works:

**Step 1: Splitting the Numbers**

Given two n-digit numbers X and Y:

- Split X and Y into two halves:
  - X can be represented as $X_1 \times 10^m + X0$
  - Y can be represented as $Y_1 \times 10^m + Y0$

Here:

- $X_1$ and $X_1$ are the higher parts of X and Y.
- $X_0$ and $X_0$ are the lower parts of X and Y.
- m is chosen to be about half of n.

For example, if X=1234 and Y=5678, we could split them as:

- $X = 12 \times 10^2 + 34$
- $Y = 56 \times 10^2 + 78$

**Step 2: Recursive Multiplication**

The product P= X × Y can be expanded as :

$P = (X_1 \times 10^m + X_0) \times (Y_1 \times 10^m + Y_0)$

Expanding this, we get:

$P = X_1 \times Y_1 \times 10^{2m} + (X_1 \times Y_0 + X_0 \times Y_1) \text{ x } 10^m + X_0 \times Y_0$

The Karatsuba algorithm optimizes this by rewriting the middle term:

1. Compute $Z_0 = X_0 \times Y_0$
2. Compute $Z_2 = X_1 \times Y_1$
3. Compute $Z_1 = (X_1 + X_0) \times (Y_1 + Y_0) - Z_2 - Z_0$

Thus, we have:

$P = Z_2 \times 10^{2m} + Z_1 \times 10^m + Z_0$

**Step 3: Recursive Calls and Time Complexity**

Each multiplication now requires only three multiplications of half-size numbers instead of four, as in the standard multiplication. This division and recursive approach lead to a recurrence relation that results in a time complexity of $O(n \wedge \log_2 3) \approx O(n \wedge 1.585)$.

**Algorithm: -**

**Programming Language: -** C++

**Code :-**

```
#include <bits/stdc++.h>
using namespace std;


int Get_Size(long long num) {
    return num == 0 ? 1 : static_cast<int>(log10(num)) + 1;
}

long long int Karatsuba(long long num1, long long num2) {

    if (num1 < 10 || num2 < 10) {
        return num1 * num2;
    }
```

```
    int length = max(Get_Size(num1), Get_Size(num2));
    int half = length / 2 + length % 2;
    long long powerOf10 = static_cast<long long>(pow(10, half));
    long long powerOf102x = powerOf10 * powerOf10;
    long long a = num1 / powerOf10;
    long long b = num1 % powerOf10;
    long long c = num2 / powerOf10;
    long long d = num2 % powerOf10;
    long long ac = Karatsuba(a, c);
    long long bd = Karatsuba(b, d);
    long long ab_cd = Karatsuba(a + b, c + d);
    long long int ans = ac * powerOf102x + (ab_cd - ac - bd) * powerOf10 + bd;

    return ans;
}

int main() {

    long long x;
    cout << "Enter the First Number :- ";
    cin >> x;

    long long y;
    cout << "Enter the Second Number :- ";
    cin >> y;

    long long int ans = Karatsuba(x, y);
    cout << "The Product of " << x << " and " << y << " is: " << ans;

    return 0;

}
```

**Output :-**

```
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 5> cd "d:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 5\" ; if ($?) { g++ Karatsuba_Multiplication.cpp -o Karatsuba_Multiplication } ; if ($?) { .\Karatsuba_Multiplication }
Enter the First Number :- 123456
Enter the Second Number :- 456789
The Product of 123456 and 456789 is: 56393342784
PS D:\Aryan Data\Usefull Data\Semester - 5\Design-and-Analysis-of-Algorithms\Lab - Manual\Experiment - 5>
```

**Space Complexity:- ________**
**Justification: -**

______________________________________________
______________________________________________
______________________________________________
______________________________________________
______________________________________________

**Time Complexity:**

**Best Case Time Complexity: ________**
**Justification: -**

______________________________________________
______________________________________________
______________________________________________
______________________________________________
______________________________________________

**Worst Case Time Complexity:- __________**
**Justification: -**

______________________________________________
______________________________________________
______________________________________________
______________________________________________
______________________________________________

**Conclusion:-**

______________________________________________
______________________________________________
______________________________________________
______________________________________________
______________________________________________