Reg.No : 24BCE0554
Name : Partha Pratim Gogoi
Topic : Karatsuba's Algorithm

Algorithm:

```
1   num1 = …                                                    [T.C: O(1)]
2   num2 = …                                                    [T.C: O(1)]
3   karatsuba(num1,num2)                          [T.C: T(n) = O(n^1.5)]
    3.1  n = max(log(num1)+1, log(num2)+1)                      [T.C: O(1)]
    3.2  half = n/2                                             [T.C: O(n)]
    3.3  p = 10^(half)                                          [T.C: O(n)]
    3.4  int a = num1/p                                         [T.C: O(n)]
    3.5  int b = num1%p                                         [T.C: O(n)]
    3.6  int c = num2/p                                         [T.C: O(n)]
    3.7  int d = num2%p                                         [T.C: O(n)]
    3.8  int z0 = karatsuba(a,c)                  [T.C: T(n) = T(n/2)]
    3.9  int z1 = karatsuba(b,d)                  [T.C: T(n) = T(n/2)]
    3.10 int z2 = karatsuba(a+b,c+d)              [T.C: T(n) = T(n/2)]
    3.11 return z2*p^(2) + (z1-z2-z0)*p + z0                    [T.C:O(1)]
```

Time Complexity:

```
karatsuba(a,c)          → T(n/2)   [halfed digits, implies 'n/2 time']
karatsuba(b,d)          → T(n/2)   [halfed digits, implies 'n/2 time']
karatsuba(a+b,c+d)      → T(n/2)   [halfed digits, implies 'n/2 time']
mult, div operations    → O(n)     [defined]
max( … )                → O(1)     [constant time, single comparision]
```

Hence,

$T.C. = [3T(n/2) + O(n)] + 5O(n) + 2O(1)$

By Master's Theorem

$T(n) = O(n^{\log_2 3})$

$\sim O(n^{1.5})$

Total Time Complexity $= O(n^{1.5}) + 5O(n) + 2O(1)$

$= O(n^{1.5})$

Source Code:

```
#include <iostream>
#include <math.h>
using namespace std;

//////////////////
/// Main Logic ///
//////////////////
long long karatsuba(long long num1, long long num2) {
   // ---------------------------------- //
   // Base Condition for smaller numbers //
   // ---------------------------------- //
   if (num1<10 || num2<10) return num1*num2;

   // ----------------------------- //
   // Initialize starting variables //
   // ----------------------------- //
   int n = max(log10(num1)+1, log10(num2)+1);
   int half = n/2;
   long long p = pow(10, half);

   // ----------------- //
   // Split the numbers //
   // ----------------- //
   // First number
   int a = num1 / p;
   int b = num1 % p;
   // Second number
   int c = num2 / p;
   int d = num2 % p;

   // -------------------------- //
   // 3 recursive multiplications //
   // -------------------------- //
   int X = karatsuba(a,c);
   int Y = karatsuba(b,d);
   int Z = karatsuba(a+b, c+d)-X-Y;

   // ------------ //
   // Final result //
   // ------------ //
   return X*p*p + Z*p + Y;
}

//////////////////
/// Driver Code ///
//////////////////
int main() {
   long long num1, num2;
   cout << "Enter first number: ";
   cin >> num1;
   cout << "Enter second number: ";
   cin >> num2;

   long long res = karatsuba(num1, num2);
   cout << "The result is: " << res << endl;
   return 0;
}
```

Sample Output:

```
rug-arch@Oxide [Karatsuba's Algorithm]>> g++ karatsuba.cpp
rug-arch@Oxide [Karatsuba's Algorithm]>> ./a.out
Enter first number: 14241
Enter second number: 1235
The result is: 469610834
```

Reg.No : 24BCE0554
Name : Partha Pratim Gogoi
Topic : Maximum Sub-array Sum

Algorithm

```
1  getMaxCrossSum(arr, low, mid, high)                          [T.C: O(n)]
   1.1  leftSide = INT_MIN
   1.2  sum = 0
   1.3  for i mid→low                                           [T.C: O(n)]
      1.3.1   sum+=arr[i]
      1.3.2   leftSide = max(leftSide, sum)
   1.4  rightSide = INT_MIN
   1.5  sum = 0
   1.6  for i low→mid                                           [T.C: O(n)]
      1.6.1   sum+=arr[i]
      1.6.2   rightSide = max(rightSide, sum)
   1.7  return leftSide + rightSide
2  getMaxSubarraySum(arr, low, high)                          [T.C: O(nlogn)]
   2.1  if low==high then return arr[low]
   2.2  mid = low + (high-low)/2
   2.3  return max(getMaxSubarraySum(arr,low, mid),     [T(n)=T(n/2)]
                   getMaxSubarraySum(arr,mid+1,high),   [T(n)=T(n/2)]
                   getMaxCrossSum(arr,low,mid,high))      [T(n)=O(n)]
3  getMaxSubarray()
```

Time Complexity

getMaxSubarray()

T.C. = T(n/2) + T(n/2) + O(n)
= 2T(n/2) + O(n)

By Master's Theorem

T.C. = O(nlogn)

Total Time Complexity = O(nlogn)

Source Code

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

//////////////////////
/// Cross sum logic ///
//////////////////////
long long getMaxCrossSum(const vector<int>& arr, int low, int mid, int high) {
      // --- Initialize sum --------- //
      long long sum = 0;

      // --- Left side -------------- //
      long long leftSide = INT_MIN;
      sum = 0;
      for (int i=mid; i>=low; i--) {
            sum += arr[i];
            leftSide = max(leftSide, sum);
      }
      // --- Right side ------------- //
      long long rightSide = INT_MIN;
      sum = 0;
      for (int i=mid+1; i<=high; i++) {
            sum += arr[i];
            rightSide = max(rightSide,sum);
      }
      // --- Debugging --- //
      // cout << leftSide << endl;
      // cout << rightSide << endl;
      // cout << leftSide+rightSide << endl << endl;

      return leftSide + rightSide;
}
//////////////////
/// Propagation ///
//////////////////
int getMaxSubarraySum(const vector<int>& arr, int low, int high) {
      if (low == high) return arr[low];

      // --- Initialize mid -------- //
      int mid = low + (high-low)/2;

      // --- Get the maximum possible subarray ------- //
      // --- CASE 1: Left subarray sum is largest ---- //
      // --- CASE 2: Right subarray sum is largest --- //
      // --- CASE 3: Cross subarray sum is largest --- //
      int leftSum = getMaxSubarraySum(arr, low, mid);
      int rightSum = getMaxSubarraySum(arr, mid+1, high);
      int crossSum = (int) getMaxCrossSum(arr, low, mid, high);

      // --- Find Max sub-array sum value --- //
      int maxSum = max(leftSum, rightSum);
      maxSum = max(maxSum, crossSum);

      return maxSum;
}
```

```
/////////////////
/// Intiation ///
/////////////////
int maxSubarraySum(vector<int>& arr) {
      return getMaxSubarraySum(arr, 0, arr.size()-1);
}

///////////////////
/// Driver Code ///
///////////////////
int main() {
      vector<int> arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
      int maxSum = maxSubarraySum(arr);

      cout << "Maximum sub-array sum possible in given array is: ";
      cout << maxSum << endl;

      return 0;
}
```

Sample Output

```
24BCE0554

◄ 0s ◎ ../'[53] - Maximum Subarray Sum'/a.out
Maximum sub-array sum possible in given array is: 6
```

Reg.No : 24BCE0554
Name : Partha Pratim Gogoi
Topic : Longest Common Subsequence

Algorithm:

```
1   longestCommonSubsequence(a,b):                                  [T.C: O(n*m)]
1.1   n = length(a)                                                   [T.C: O(1)]
1.2   m = length(b)                                                   [T.C: O(1)]
1.3   dp[n+1][m+1] = {0};                                  [T.C: O((n+1)*(m+1))]
1.4   for i 1→n:                                                      [T.C: O(n)]
1.4.1   for j 1→m:                                                    [T.C: O(m)]
1.4.1.1     if (a[i-1]==b[j-1]):                                    [T.C: O(1)]
1.4.1.2     dp[i][j] = 1+dp[i-1][j-1]                               [T.C: O(1)]
1.4.1.3     else:                                                   [T.C: O(1)]
1.4.1.4     dp[i][j] = max(dp[i][j-1],dp[i-1][j])                   [T.C: O(1)]
1.5   return dp[n][m]
```

Time Complexity:

Total Time Complexity = O((n+1)*(m+1)) + O(n)*O(m) + 2O(1)
= O(n*m)

Source Code:

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;


int longestCommonSubsequence(string a, string b) {
      int n = a.length();
      int m = b.length();

      // --- Create 0-initialized 2D array to build LCS --- //
      // --- vector<int> inner(m+1, 0); ------------------- //
      // --- vector<vector<int>> dp(n+1, inner); ---------- //
      vector<vector<int>> dp(n+1, vector<int>(m+1, 0));

      // --- LCS Algorithm --- //
      for (int i=1; i<n+1; i++) {
      for (int j=1; j<m+1; j++) {
            // -------------------------- //
            // --- Check element match --- //
            // -------------------------- //
            if (a[i-1] == b[j-1]) {
                  // ---------------- DEBUG Statements -------------- //
                  // cout << endl << "Element match:" << endl; ------ //
                  // cout << dp[i][j] << endl; ---------------------- //
                  // cout << dp[i-1][j-1] << endl; ------------------ //
                  // ------------------------------------------------ //
                  dp[i][j] = 1 + dp[i-1][j-1];
            } else {
                  // ---------------- DEBUG Statements -------------- //
                  // cout << endl << "Element mismatch:" << endl; --- //
                  // cout << dp[i][j] << endl; ---------------------- //
                  // cout << dp[i][j-1] << endl; -------------------- //
                  // cout << dp[i-1][j] << endl; -------------------- //
                  // ------------------------------------------------ //
                  dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
            }
      }}

      return dp[n][m];
}

int main() {
      string a = "wrtvertubryvcwrybutruibvctyret";
      string b = "wqercfw6vtetubervycwetyebvc";

      cout << "Length of Longest Common Subqeuence: " <<
longestCommonSubsequence(a,b);
      cout << endl;

      return 0;
}
```

Sample Output:

```
24BCE0554

◄ 0s ◎ ../'[1143] - Longest Common Subsequence'/a.out
Length of Longest Common Subqevence: 15
```