DAA LABORATORY 4

Name:Dhruv Panchal

Roll NO;231070038

SY Btech Comp eng.

TASK 1:

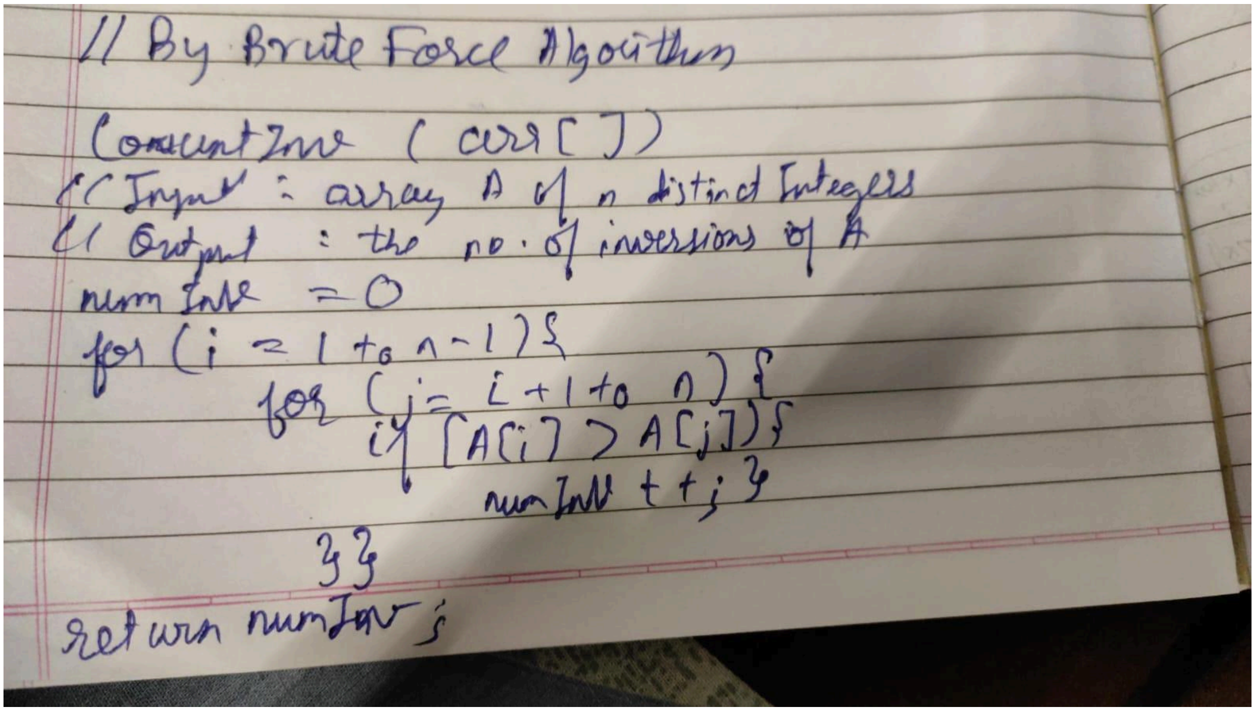
Aim:

Consider first/second year course-code choices of 100 students.

Find the inversion count of these choices.

Find students with zero, one, two, three inversion counts and comment on your result.

ALGO by Brute Force:



```
// By Brute Force Algorithm
CountInv ( arr[ ])
// Input : array A of n distinct Integers
// Output : the no. of inversions of A
numInv = 0
for (i = 1 to n-1){
        for (j = i+1 to n){
            if [A[i] > A[j]){
                numInv ++; }
        }}
return numInv;
```

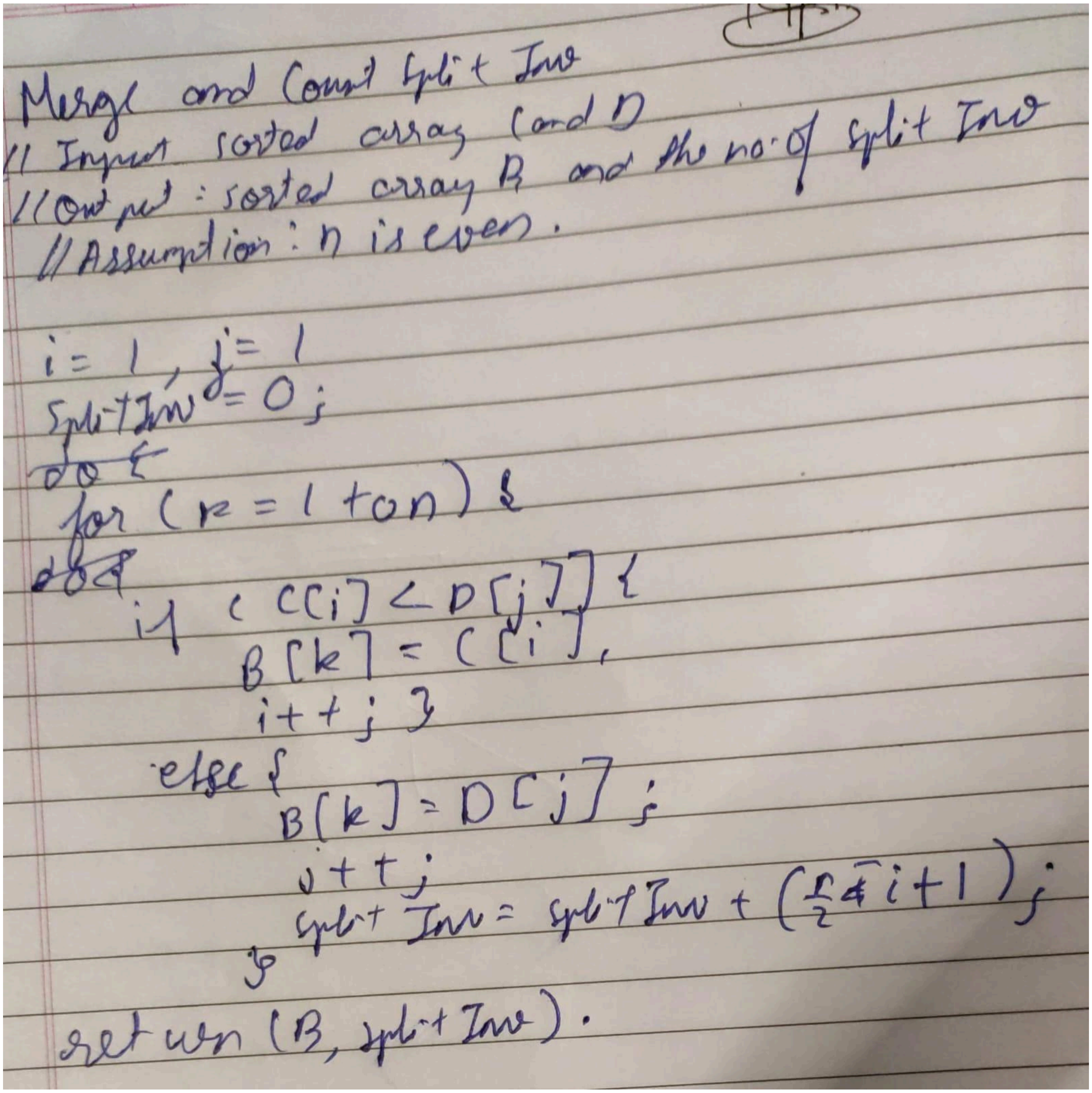
ALGO by Divide And Conquer:

// To count the no of inversions in a course code
divide & conquer technique

```
Count Inversion (vect arr[])

// Input : array A of n distinct integers
// Output : the ~~a~~ number of inversions of A

if (n = 0 or n = 1){
    return 0; }
else {
    left Inv = Cont Inversion (first half of A)
    right Inv = Count Inversion (second half of A)
    split Inv = Count ~~Split Inv~~ Inversion (A)
    return left Inv + right Inv + split Inv
```

TO break down all the in half

```
Sort and Count
if n = 0 ~~and~~ or n = 1{
    return (A, 0) }
else {
    (C, left Inv) = Sort and Count (first half of A)
    (D, right Inv) = Sort and Count(second half of A)
    (B, split Inv) = Sort and count (C, D)
    return (B, left Inv + RightInv + splitInv);
}
```



Merge and Count Split Inv

```
// Input sorted array C and D
// Output : sorted array B and the no. of Split Inv
// Assumption : n is even.

i = 1, j = 1
SplitInv = 0;
~~do {~~
for (k = 1 to n) {
~~do {~~
    if ( C[i] < D[j] ) {
        B[k] = C[i],
        i++; }
    else {
        B[k] = D[j];
        j++;
        SplitInv = SplitInv + ($\frac{n}{2}$ - i + 1);
    }
return (B, SplitInv).
```

TIME COMPLEXITY:

PRABHAT
PAGE NO. :
DATE : / /

Time Complexity {

//Brute Force Algo.

We are using a ~~f~~ nested for-loop in which first we iterate ~~of~~ all element of the array and in the next for loop we compare them with all other elements in the array to compare if there is any inversion.

and so as we know for 2 (nested for loop) the time complexity would be $\underline{O(n^2)}$.

// Divide & Conquer ~~AP~~ Technique.

Divide: The divide step just finds the middle of the subarray, which takes constant time.

$\therefore$ $O(1)$.

Conquer: We recursively solve two subproblems, each of size $n/2$; which contributes $2T(n/2)$.

Combine: ~~We have already noted~~ So Merge on a n-element subarray takes $O(n)$ time.

$\therefore$ Total Time

$$T(n) = \begin{cases} O(1) & , n = 1 \\ 2T(n/2) + O(n) & , \text{if } n > 1 \end{cases}$$

By master's Thm.

$$T(n) = 2T(n/2) + O(n)$$

$$f(n) = O(n) \quad a = 2,\ b = 2,\ d = 1$$

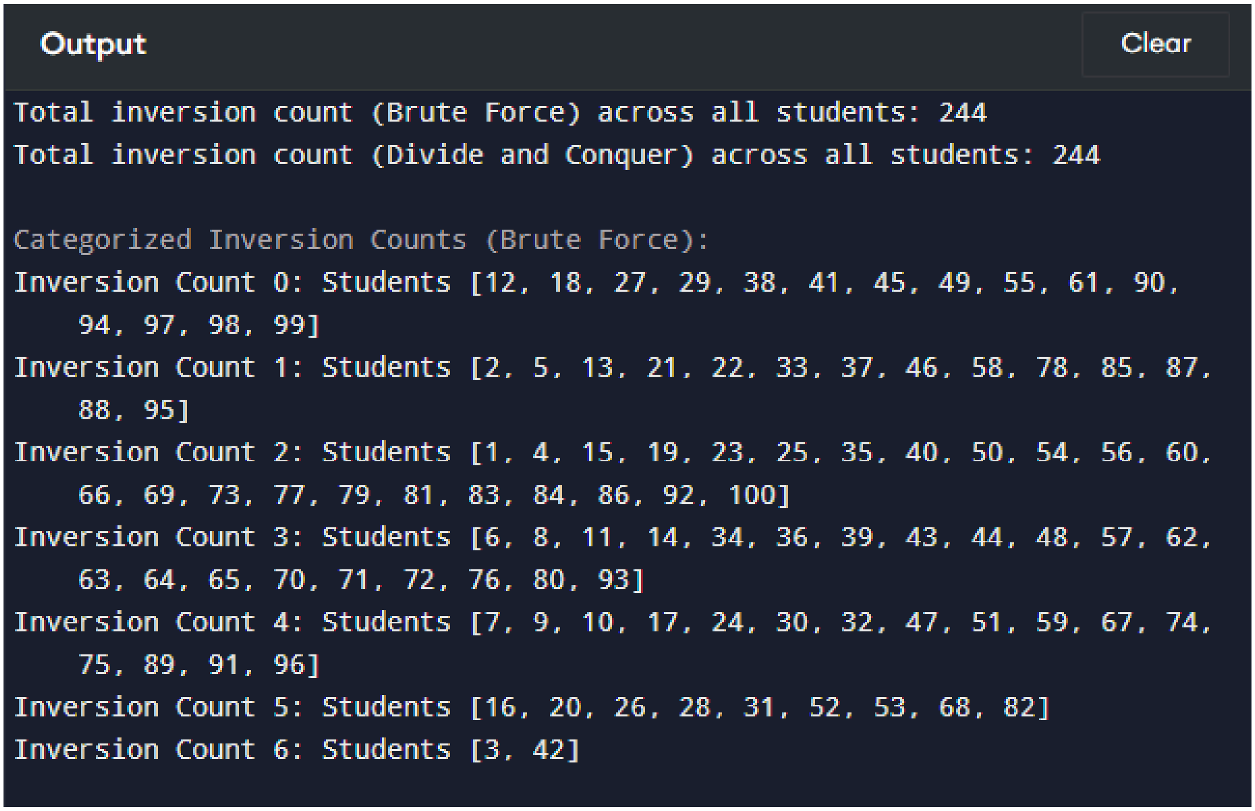
$$a = b$$

$$\therefore T(n) = O(n^d \log n)$$

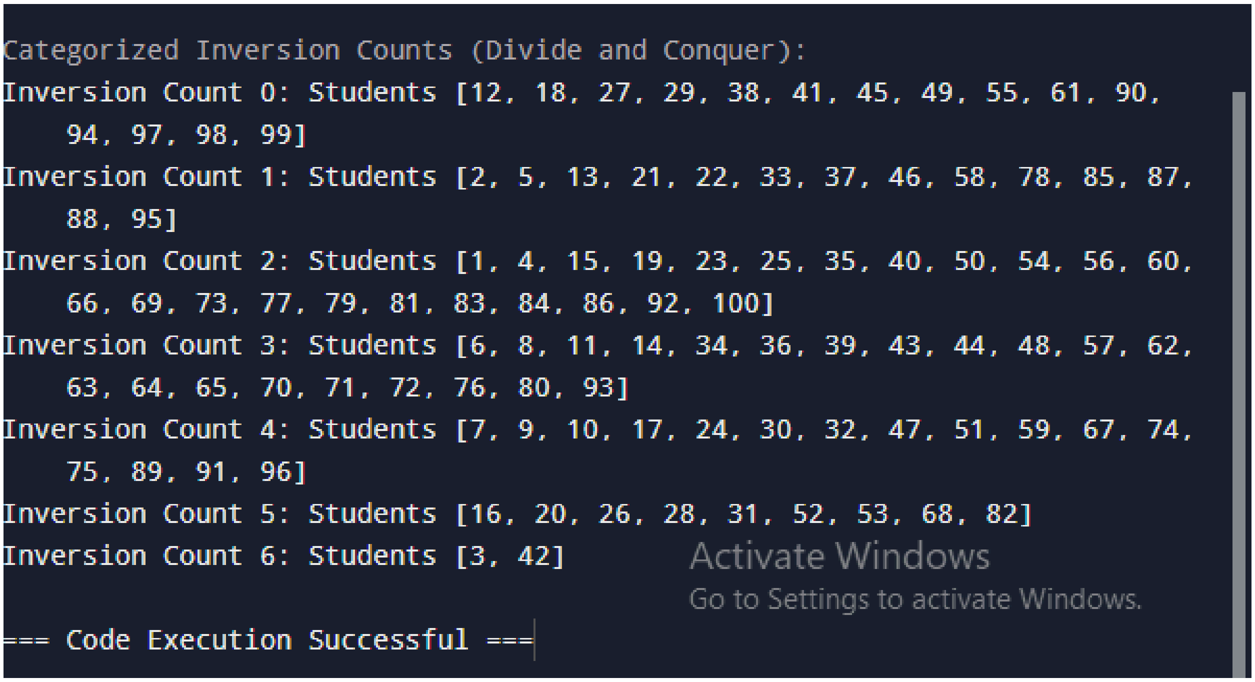
$$= O(n \log n)$$ //

POSITIVE TESTCASES:

1) **INput:**



```
Total inversion count (Brute Force) across all students: 244
Total inversion count (Divide and Conquer) across all students: 244

Categorized Inversion Counts (Brute Force):
Inversion Count 0: Students [12, 18, 27, 29, 38, 41, 45, 49, 55, 61, 90,
    94, 97, 98, 99]
Inversion Count 1: Students [2, 5, 13, 21, 22, 33, 37, 46, 58, 78, 85, 87,
    88, 95]
Inversion Count 2: Students [1, 4, 15, 19, 23, 25, 35, 40, 50, 54, 56, 60,
    66, 69, 73, 77, 79, 81, 83, 84, 86, 92, 100]
Inversion Count 3: Students [6, 8, 11, 14, 34, 36, 39, 43, 44, 48, 57, 62,
    63, 64, 65, 70, 71, 72, 76, 80, 93]
Inversion Count 4: Students [7, 9, 10, 17, 24, 30, 32, 47, 51, 59, 67, 74,
    75, 89, 91, 96]
Inversion Count 5: Students [16, 20, 26, 28, 31, 52, 53, 68, 82]
Inversion Count 6: Students [3, 42]
```



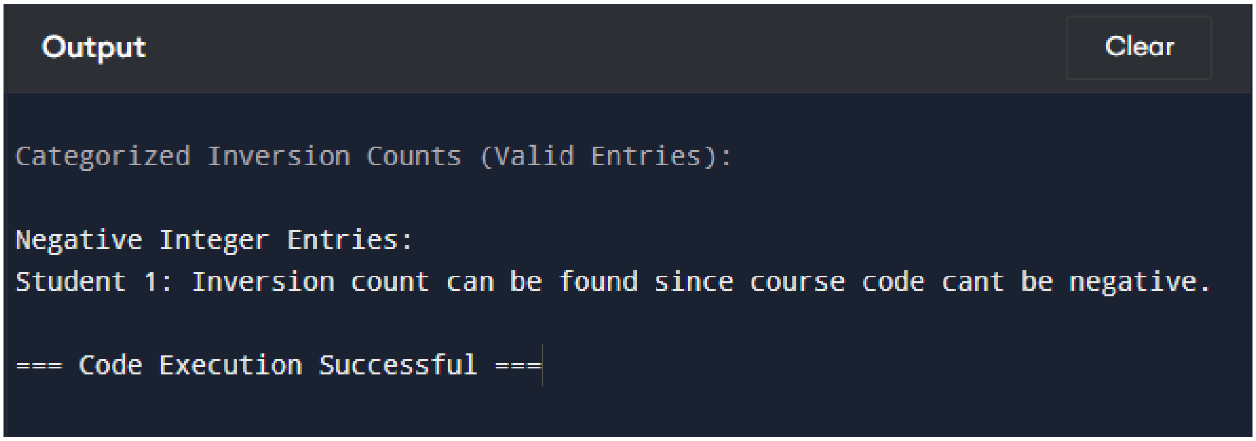
```
Categorized Inversion Counts (Divide and Conquer):
Inversion Count 0: Students [12, 18, 27, 29, 38, 41, 45, 49, 55, 61, 90,
    94, 97, 98, 99]
Inversion Count 1: Students [2, 5, 13, 21, 22, 33, 37, 46, 58, 78, 85, 87,
    88, 95]
Inversion Count 2: Students [1, 4, 15, 19, 23, 25, 35, 40, 50, 54, 56, 60,
    66, 69, 73, 77, 79, 81, 83, 84, 86, 92, 100]
Inversion Count 3: Students [6, 8, 11, 14, 34, 36, 39, 43, 44, 48, 57, 62,
    63, 64, 65, 70, 71, 72, 76, 80, 93]
Inversion Count 4: Students [7, 9, 10, 17, 24, 30, 32, 47, 51, 59, 67, 74,
    75, 89, 91, 96]
Inversion Count 5: Students [16, 20, 26, 28, 31, 52, 53, 68, 82]
Inversion Count 6: Students [3, 42]

=== Code Execution Successful ===
```

2)

NEGATIVE TESTCASES;

If no input is given(empty array)

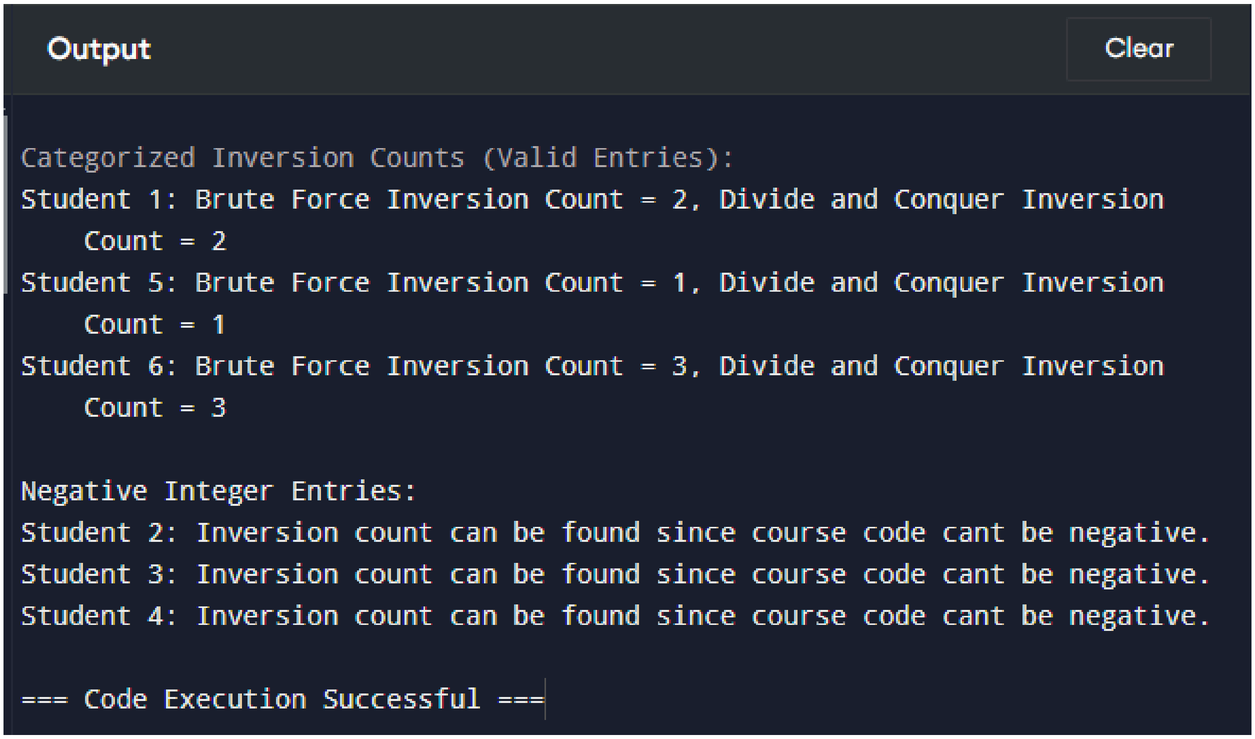
INPUT:[ ]



```
Categorized Inversion Counts (Valid Entries):

Negative Integer Entries:
Student 1: Inversion count can be found since course code cant be negative.

=== Code Execution Successful ===
```

IF negative no is inputted

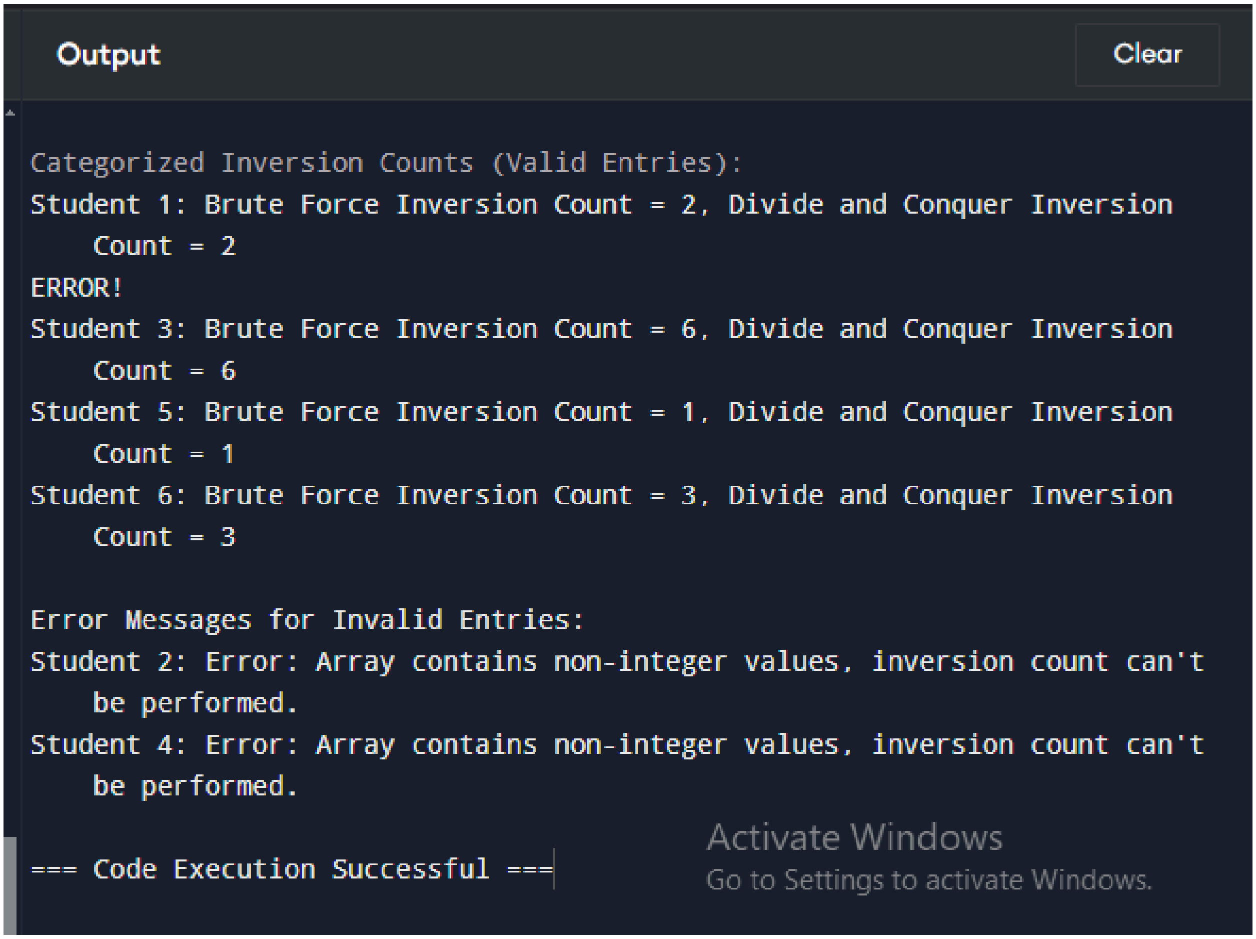
INPUT: [5, 2, 3, 6], [-3, -1, -5, -2], [-7, -6, -4, -1], [-6, -2, -5, -7], [2, 3, 8, 4], [5, 5, 5, 4]



```
Categorized Inversion Counts (Valid Entries):
Student 1: Brute Force Inversion Count = 2, Divide and Conquer Inversion
    Count = 2
Student 5: Brute Force Inversion Count = 1, Divide and Conquer Inversion
    Count = 1
Student 6: Brute Force Inversion Count = 3, Divide and Conquer Inversion
    Count = 3

Negative Integer Entries:
Student 2: Inversion count can be found since course code cant be negative.
Student 3: Inversion count can be found since course code cant be negative.
Student 4: Inversion count can be found since course code cant be negative.

=== Code Execution Successful ===
```

If letters are inputted in array

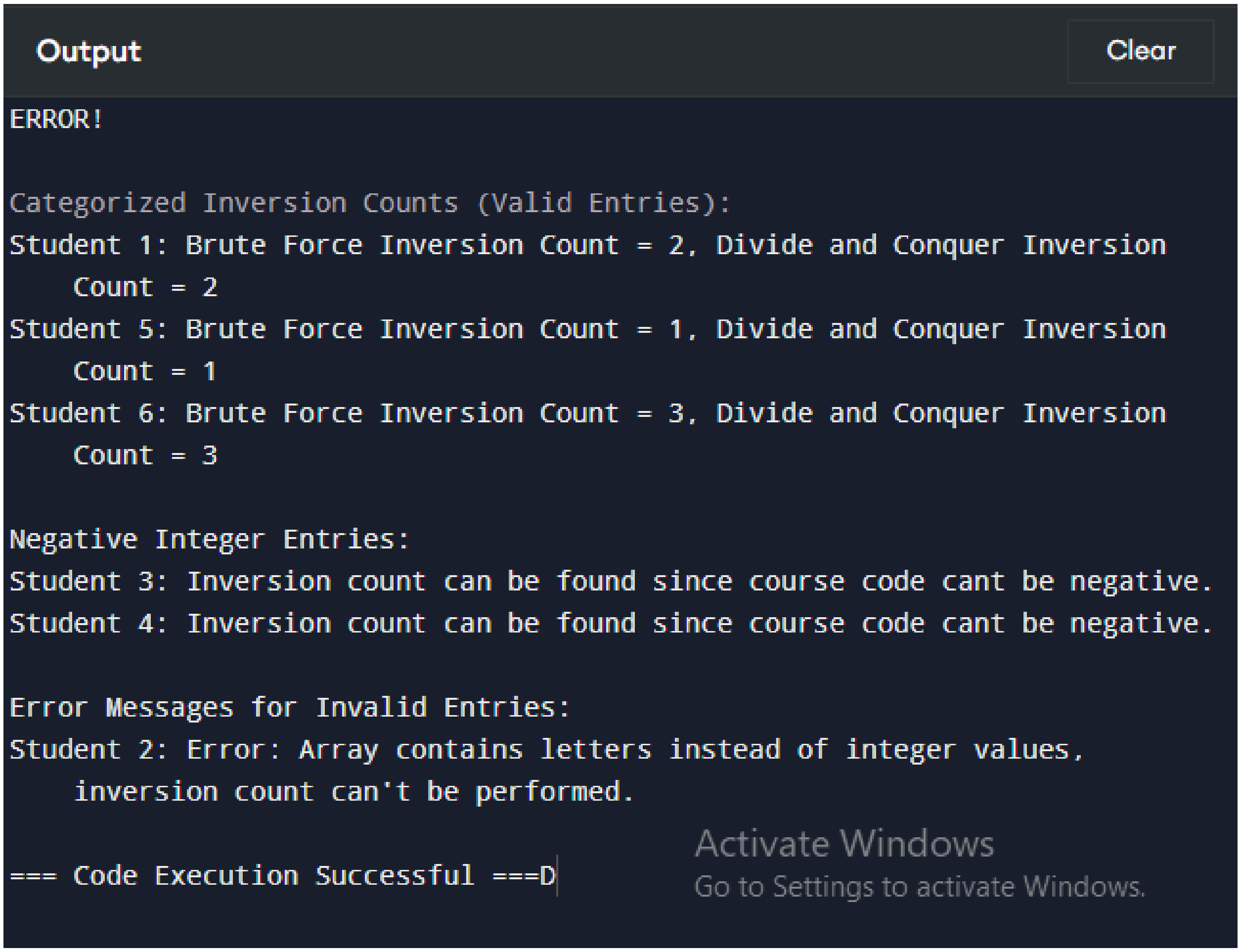
INPUT: [5, 2, 3, 6], ['a', 1, 5, 2], [7, 6, 4, 1], [6, 2, 'b', 7], [2, 3, 8, 4], [5, 5, 5, 4]



```
Categorized Inversion Counts (Valid Entries):
Student 1: Brute Force Inversion Count = 2, Divide and Conquer Inversion
    Count = 2
ERROR!
Student 3: Brute Force Inversion Count = 6, Divide and Conquer Inversion
    Count = 6
Student 5: Brute Force Inversion Count = 1, Divide and Conquer Inversion
    Count = 1
Student 6: Brute Force Inversion Count = 3, Divide and Conquer Inversion
    Count = 3

Error Messages for Invalid Entries:
Student 2: Error: Array contains non-integer values, inversion count can't
    be performed.
Student 4: Error: Array contains non-integer values, inversion count can't
    be performed.

=== Code Execution Successful ===
```

Both negative nd letters

input: [5, 2, 3, 6], ['a', 1, 5, 2], [-7, -6, -4, -1], [-6, -2, -5, -7], [2, 3, 8, 4], [5, 5, 5, 4]



```
ERROR!

Categorized Inversion Counts (Valid Entries):
Student 1: Brute Force Inversion Count = 2, Divide and Conquer Inversion
    Count = 2
Student 5: Brute Force Inversion Count = 1, Divide and Conquer Inversion
    Count = 1
Student 6: Brute Force Inversion Count = 3, Divide and Conquer Inversion
    Count = 3

Negative Integer Entries:
Student 3: Inversion count can be found since course code cant be negative.
Student 4: Inversion count can be found since course code cant be negative.

Error Messages for Invalid Entries:
Student 2: Error: Array contains letters instead of integer values,
    inversion count can't be performed.

=== Code Execution Successful ===D
```

CONCLUSION:

IN this task, we understand by divide and conquer we can do task with less time like in this case we could count the total no of inversions in just O(n logn) time while it took O(n^2) time while we did it in brute force algorithm.

TASK 2:

AIM:

Consider large integers of size 10, 50, 100, 500 and 1000 digits.

Write integer multiplication program

Write integer multiplication program using divide and conquer technique.

ALGO by Brute Force:

// Algo for Integer Multiplication (Brute Force)

// Input : two n-digit positive integers x and y
// Output: The product x·y
Assumption: n is a power of 2.

```
If(n = 2) {
    return x.y;}
else{
    a = first half of x;
    b = ~~first~~ second half of x;
    c = first half of x;
    d = second half of y;
recursively { ~~a.b~~
        ac = a×b;
        bd = b×d;
        a.d = a×d;
        b.c = b×c;}

    return : $10^n$ ac + $10^{n/2}$ (ad+bc) + bd;
}
```

Time complexity :-
so, here we multiply ac, bd, ad, bc, recursively for n/2 elements which take $T(n/2)$ and $O(n)$ for other work;

$\therefore T(n) = \cancel{3}\, 4T(n/2) + cn$,

By master's theorem,

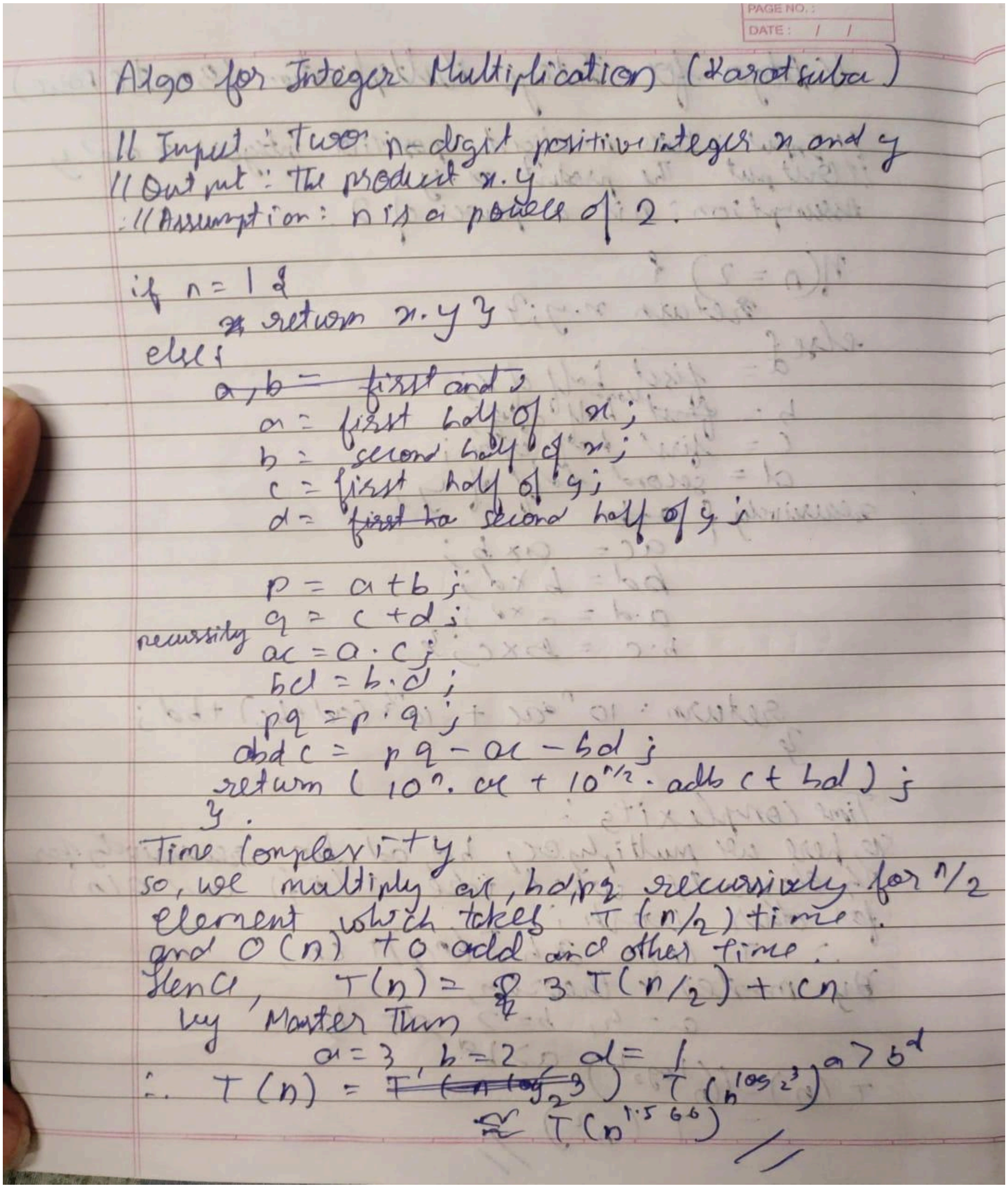
$a = 4,\ b = 2,\ d = 1;$

$a > b^a$

$T(n) = T(n^{\log_2 4})$

$= T(n^2)$ //

ALGO by Divide And Conquer:



PAGE NO. :
DATE : / /

Algo for Integer Multiplication (Karatsuba)

```
// Input : Two n-digit positive integer x and y
// Output : The product x.y
// Assumption : n is a power of 2.

if n = 1 {
    ~~x~~ return x.y }
else {
    ~~a, b = first and ;~~
        a = first half of x;
        b = second half of x;
        c = first half of y;
        d = ~~first ha~~ second half of y;

        p = a + b;
        q = c + d;
recursily  ac = a . c;
        bd = b . d;
        pq = p . q;
      adbc = pq - ac - bd;
    return ( 10^n . ac + 10^(n/2) . adbc + bd );
    }.
```

Time Complexity:

so, we multiply ac, bd, pq recursively for $n/2$ element which takes $T(n/2)$ time. and $O(n)$ to add and other time.

Hence, $T(n) = 3T(n/2) + cn$

by Master Thm

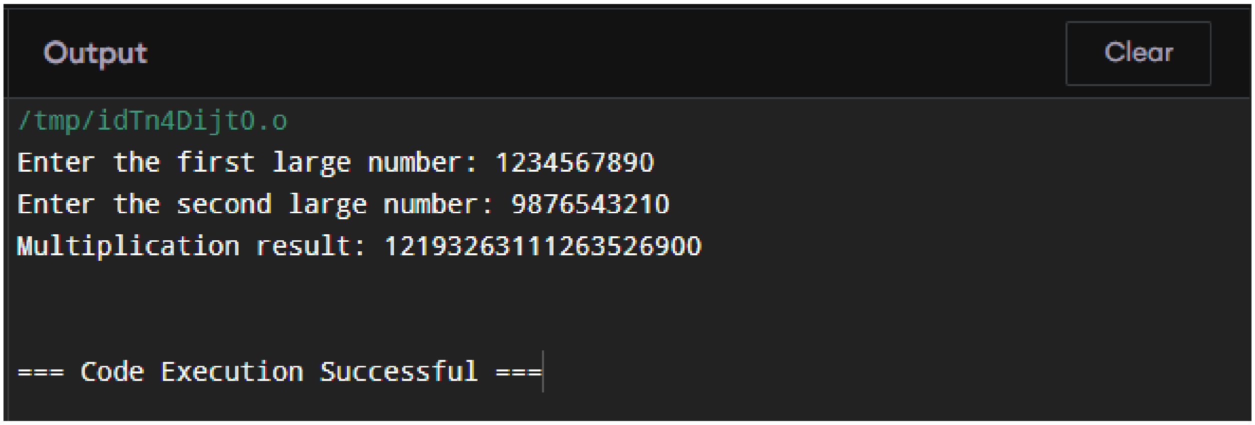
$a = 3$, $b = 2$, $d = 1$ $\quad a > b^d$

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

$\simeq T(n^{1.566})$ //

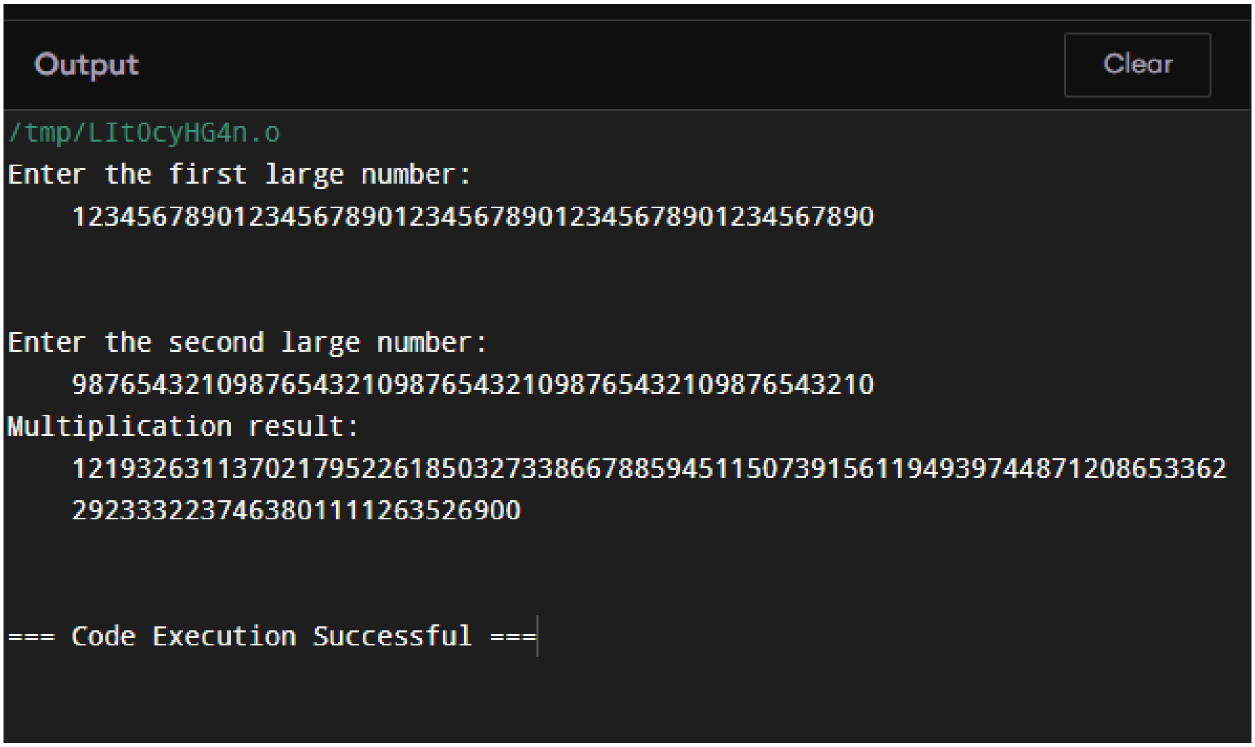
POSITIVE TESTCASES:

10 digits:



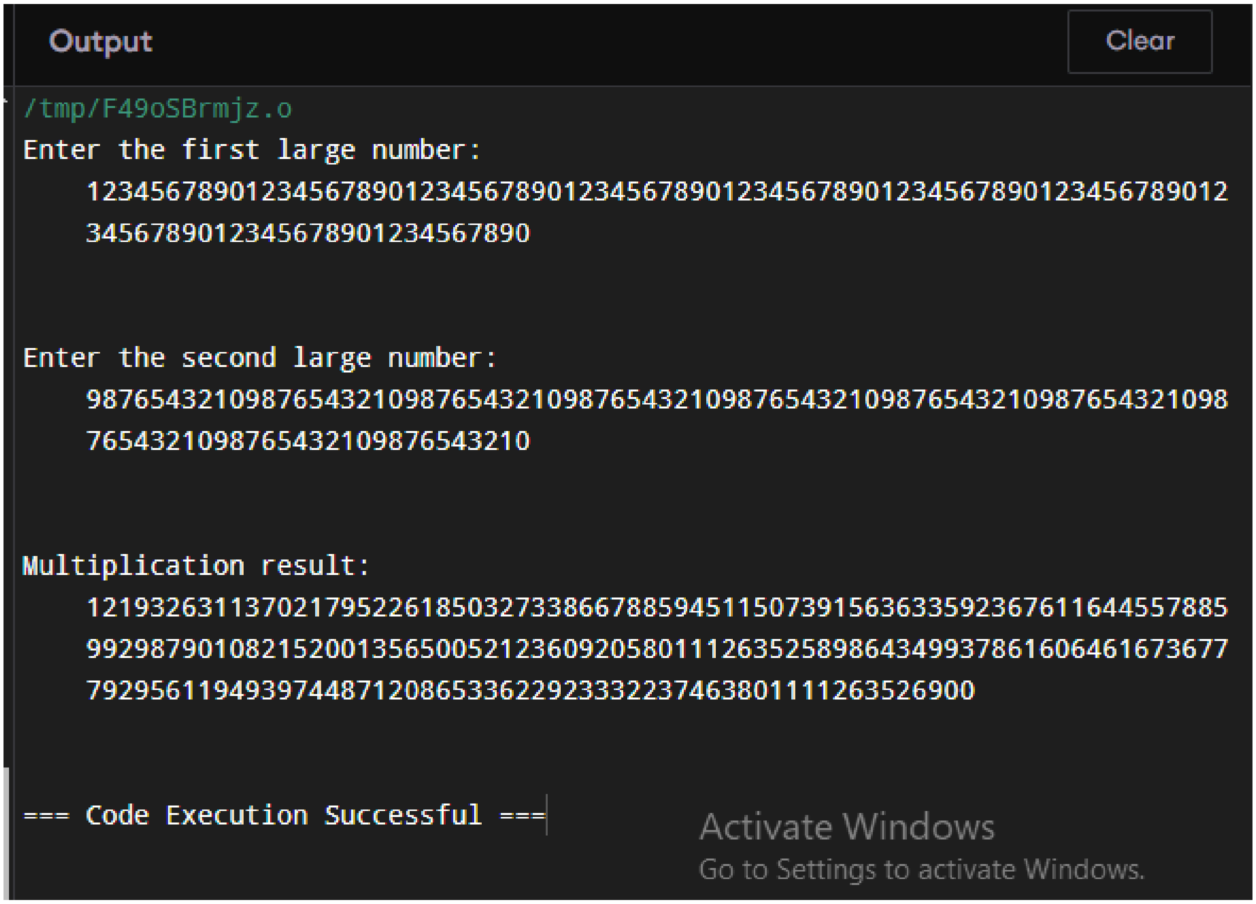
```
/tmp/idTn4Dijt0.o
Enter the first large number: 1234567890
Enter the second large number: 9876543210
Multiplication result: 12193263111263526900


=== Code Execution Successful ===
```

50 digits:



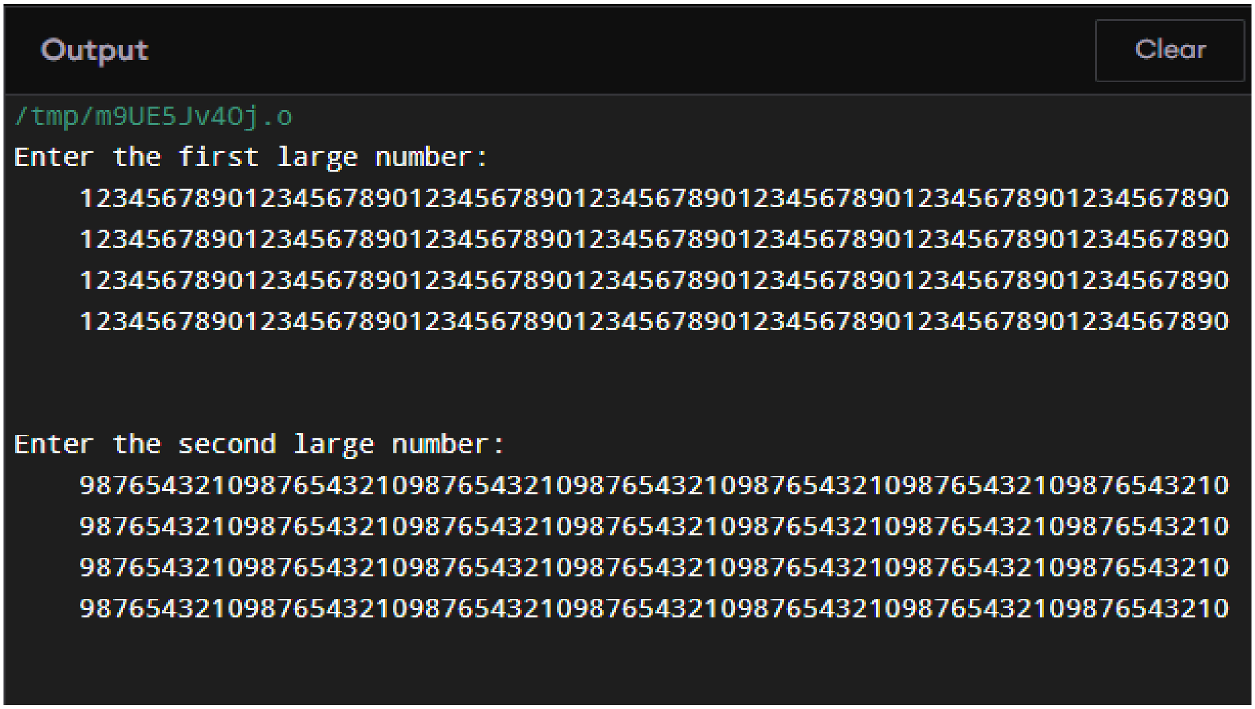
```
/tmp/LItOcyHG4n.o
Enter the first large number: 
    12345678901234567890123456789012345678901234567890


Enter the second large number: 
    98765432109876543210987654321098765432109876543210
Multiplication result: 
    121932631137021795226185032733866788594511507391561194939744871208653362
    292333223746380111263526900


=== Code Execution Successful ===
```

100 digits:

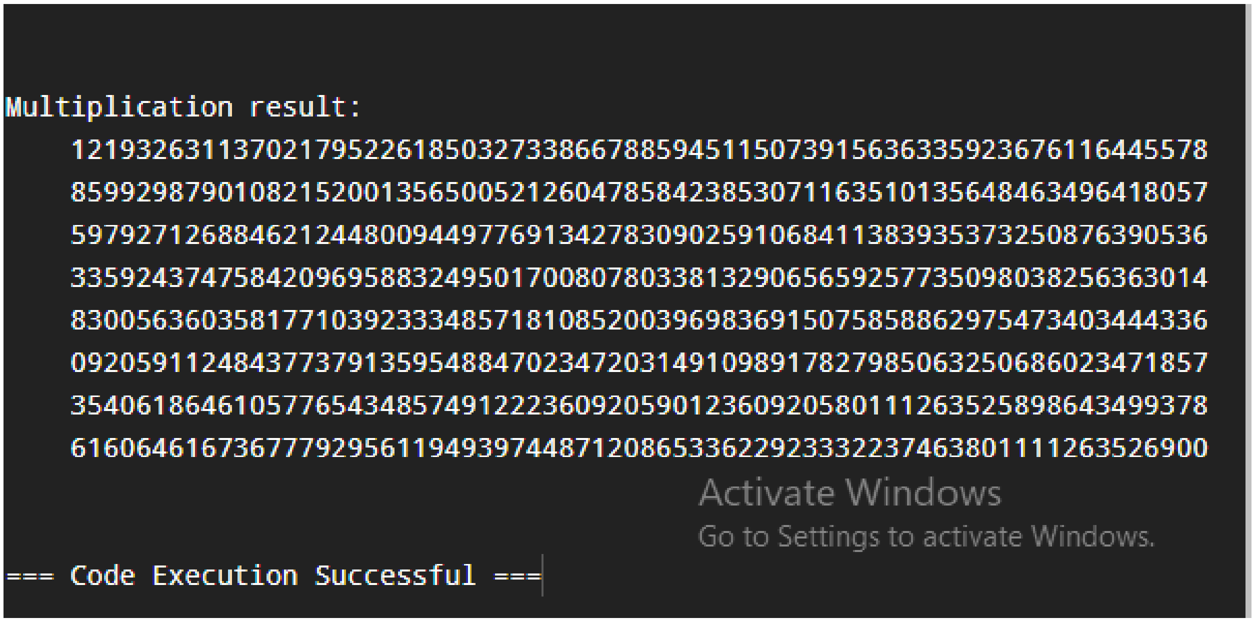


```
/tmp/F49oSBrmjz.o
Enter the first large number:
    123456789012345678901234567890123456789012345678901234567890123456789012
    3456789012345678901234567890


Enter the second large number:
    987654321098765432109876543210987654321098765432109876543210987654321098
    7654321098765432109876543210


Multiplication result:
    121932631137021795226185032733866788594511507391563633592367611644557885
    992987901082152001356500521236092058011126352589864349937861606461673677
    79295611949397448712086533622923332237463801111263526900


=== Code Execution Successful ===
```

500 digits:

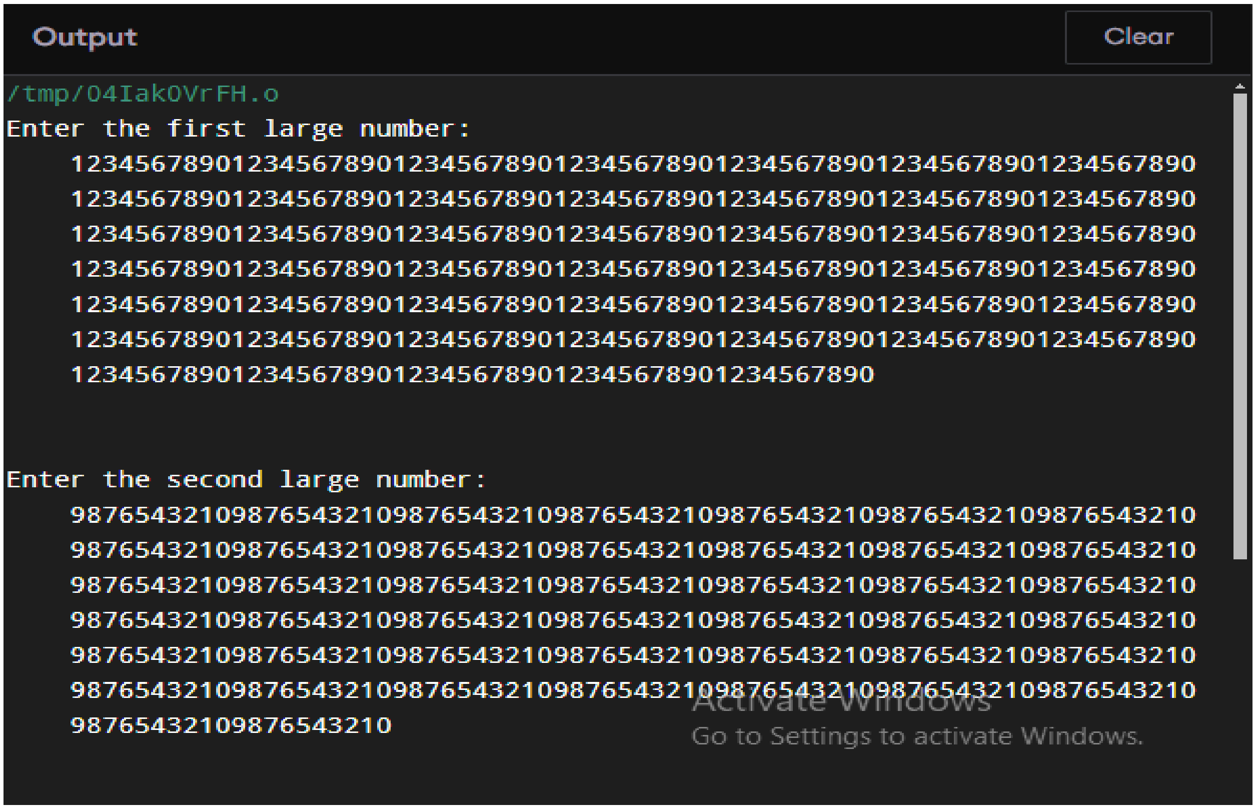


```
/tmp/m9UE5Jv40j.o
Enter the first large number:
    1234567890123456789012345678901234567890123456789012345678901234567890
    1234567890123456789012345678901234567890123456789012345678901234567890
    1234567890123456789012345678901234567890123456789012345678901234567890
    1234567890123456789012345678901234567890123456789012345678901234567890


Enter the second large number:
    9876543210987654321098765432109876543210987654321098765432109876543210
    9876543210987654321098765432109876543210987654321098765432109876543210
    9876543210987654321098765432109876543210987654321098765432109876543210
    9876543210987654321098765432109876543210987654321098765432109876543210
```

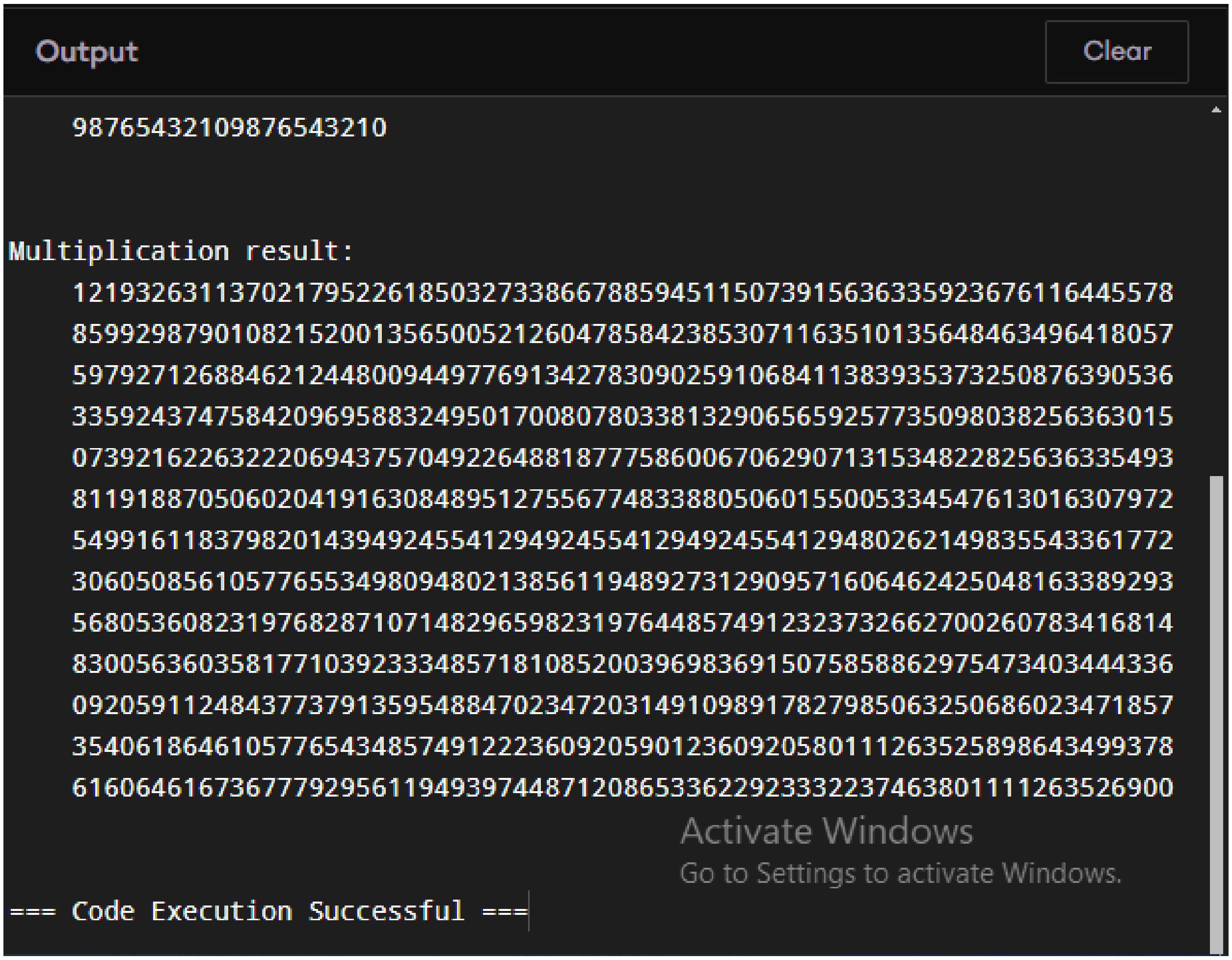


```
Multiplication result:
    1219326311370217952261850327338667885945115073915636335923676116445578
    8599298790108215200135650052126047858423853071163510135648463496418057
    5979271268846212448009449776913427830902591068411383935373250876390536
    3359243747584209695883249501700807803381329065659257735098038256363014
    8300563603581771039233348571810852003969836915075858862975473403444336
    0920591124843773791359548847023472031491098917827985063250686023471857
    3540618646105776543485749122236092059012360920580111263525898643499378
    6160646167367779295611949397448712086533622923332237463801111263526900


=== Code Execution Successful ===
```

1000 digits:

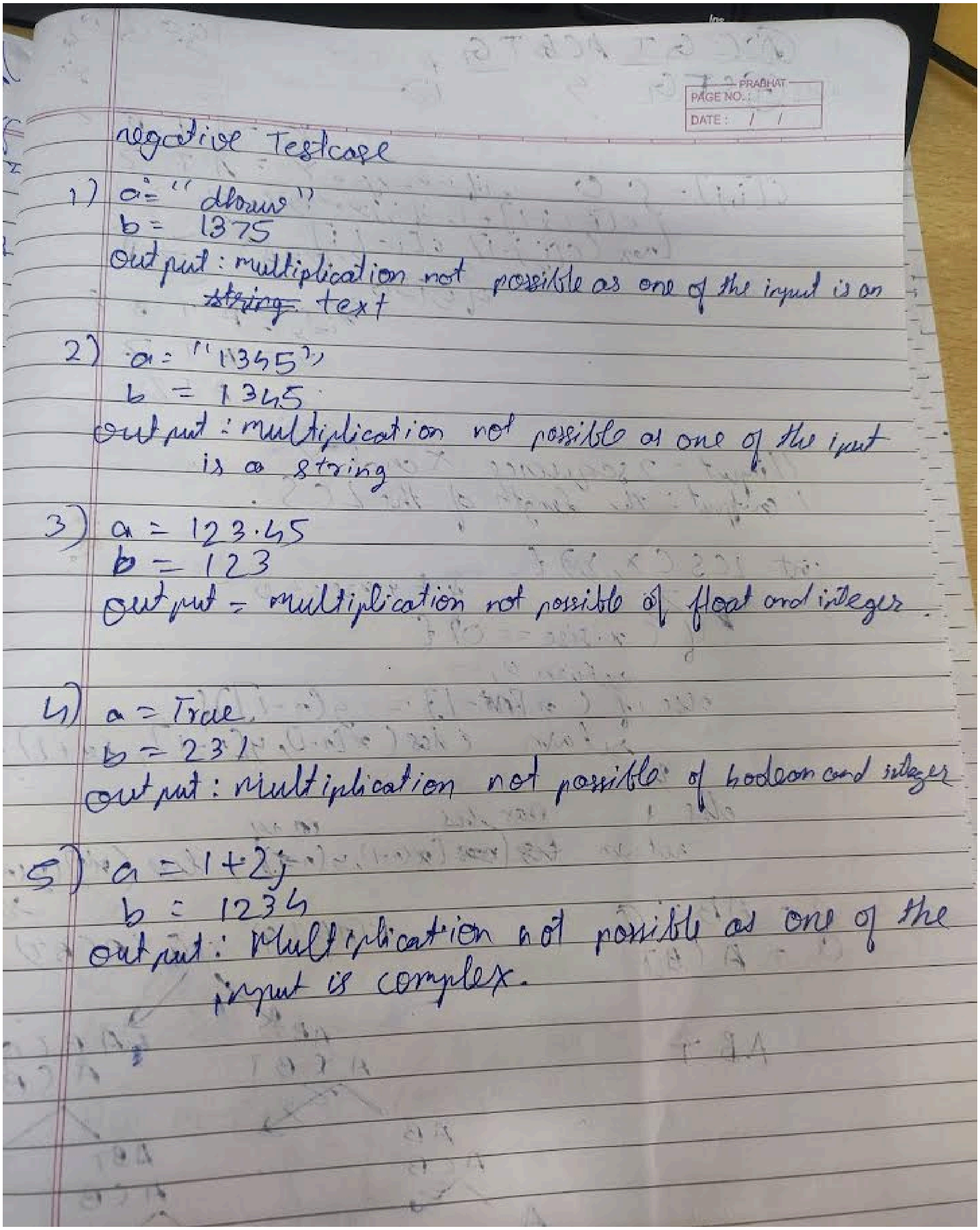


```
/tmp/04IakOVrFH.o
Enter the first large number:
    1234567890123456789012345678901234567890123456789012345678901234567890
    1234567890123456789012345678901234567890123456789012345678901234567890
    1234567890123456789012345678901234567890123456789012345678901234567890
    1234567890123456789012345678901234567890123456789012345678901234567890
    1234567890123456789012345678901234567890123456789012345678901234567890
    1234567890123456789012345678901234567890123456789012345678901234567890
    12345678901234567890123456789012345678901234567890


Enter the second large number:
    9876543210987654321098765432109876543210987654321098765432109876543210
    9876543210987654321098765432109876543210987654321098765432109876543210
    9876543210987654321098765432109876543210987654321098765432109876543210
    9876543210987654321098765432109876543210987654321098765432109876543210
    9876543210987654321098765432109876543210987654321098765432109876543210
    9876543210987654321098765432109876543210987654321098765432109876543210
    98765432109876543210
```



```
    98765432109876543210


Multiplication result:
    1219326311370217952261850327338667885945115073915636335923676116445578
    8599298790108215200135650052126047858423853071163510135648463496418057
    5979271268846212448009449776913427830902591068411383935373250876390536
    3359243747584209695883249501700807803381329065659257735098038256363015
    0739216226322206943757049226488187775860067062907131534822825636335493
    8119188705060204191630848951275567748338805060155005334547613016307972
    5499161183798201439492455412949245541294924554129480262149835543361772
    3060508561057765534980948021385611948927312909571606462425048163389293
    5680536082319768287107148296598231976448574912323732662700260783416814
    8300563603581771039233348571810852003969836915075858862975473403444336
    0920591124843773791359548847023472031491098917827985063250686023471857
    3540618646105776543485749122236092059012360920580111263525898643499378
    6160646167367779295611949397448712086533622923332237463801111263526900


=== Code Execution Successful ===
```

NEGATIVE TESTCASES;



PRABHAT
PAGE NO.:
DATE: / /

negative Testcase

1) a = "dlowe"
b = 1375
output : multiplication not possible as one of the input is an ~~string~~ text

2) a = "1345"
b = 1345
output : multiplication not possible as one of the input is a string

3) a = 123.45
b = 123
output = multiplication not possible of float and integer.

4) a = True
b = 237
output : multiplication not possible of boolean and integer

5) a = 1+2j
b = 1234
output : Multiplication not possible as one of the input is complex.

CONCLUSION:

By this task we learned that our normal multiplying method take n^2 time while by using karatsuba's algo we can do it in approx n^1.566 which make our work easy.