Counting Inversions:-

```
Count-Inversions (A, p, r)
// Counts the number of inversions in an array.
// Input: Array A, starting index p,
          ending index r
// Output: Number of inversions in array.
if p < r:
    q = (p+r) / 2
    left_inversions = Count-Inversions (A, p, q)
    right_inversions = Count-Inversions (A, q+1, r)
    split_inversions = Count-Merge (A, p, q, r)
    return left_inversions + right_inversions +
                split_inversions
else return 0

Count-Merge (A, p, q, r)
// Counts the split inversions of an array,
    while merging its two halves.
// Input: Array A, with start, middle, end
          indices p, q, r.
// Output: Split inversions in array.
n1 = q - p + 1
n2 = r - q
let L[1..n1+1] & R[1..n2+1] be arrays.
for i = 1 to n1
    L[i] = A[p+i-1]
for j = 1 to n2
    R[j] = A[q+j]
L[n1+1] = ∞
R[n2+1] = ∞
```

```
i = 1
j = 1
inv_count = 0
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i++
    else
        A[k] = R[j]
        j++
        inv_count += (n1 - i + 1)
return inv_count
```

Time complexity :-

The recurrence for the count-Inversions function is

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

→ $2T(n/2)$ due to recursive calls on halves of array.

→ $O(n)$ due to merging halves of array.

By master method,

$a = 2, b = 2, d = 1 \rightarrow b^d = 1$

$\because a = b^d,$

$T(n) \in O(n \log n)$

∴ Time complexity is $O(n \log n)$

Test cases :- Inversions.

| | Choices | Number of inversions |
|---|---|---|
| 1) | [101, 102, 103, 104, 105, 106] | 0 |
| 2) | [101, 102, 103, 106, 105, 104] | 3 |
| 3) | [101, 102, 104, 105, 103, 106] | 2 |
| 4) | [101, 102, 104, 103, 105, 106] | 1 |
| 5) | [101, 102, 103, 104, 106, 105] | 1 |
| 6) | [101, 103, 104, 102, 105, 106] | 2 |
| 7) | [101, 102, 104, 106, 103, 105] | 3 |
| 8) | [102, 101, 103, 104, 105, 106] | 1 |
| 9) | [101, 104, 102, 103, 105, 106] | 2 |
| 10 | [101, 102, 104, 103, 105, 106] | 2 |

Integer Multiplication

1) Brute-Force Algorithm

```
integer_multiplication (x, y)
// Input: Two integers x & y
// Output: Product of x & y.
Reverse digits of x & y & convert to string
res = 0
for i1 = 0 to x.length - 1:
        for i2 = 0 to y.length - 1:
            res += int(x[i1]) * int(y[i2]) *
                   power(10, i1+i2)
return res
```

Time complexity:

In the worst case, lengths of x & y are same. As there are two loops over lengths of x & y, performing the basic operation 'res += int(x[i1]) * int(y[i2]) * power(10, i1+i2)'

Time complexity,

$$T(n) = \sum_{i1=0}^{x.length-1} \sum_{i2=0}^{y.length-1} (c)$$

$$= \sum_{i1=0}^{n-1} \sum_{i2=0}^{n-1} (c) \quad [x.length = y.length = n]$$

$$\therefore T(n) = n^2 (c)$$

$$\therefore \boxed{T(n) \in O(n^2)}$$

Hence time complexity is $O(n^2)$.

Divide & conquer method

karatsuba - multiplication (x, y)

// Input : Two integers x & y

// Output : Product of x & y

if x<10 or y<10 return x*y

m = minimum length between x & y.

m2 = m//2.

high1, low1 = x // $10^{m2}$, x mod $10^{m2}$

high2, low2 = y // $10^{m2}$, y mod $10^{m2}$

z0 = karatsuba - multiplication (low1, low2)

z2 = karatsuba - multiplication (high1, high2)

z1 = karatsuba - multiplication (low1 + low2, high1 + high2) - z0 - z2

return $z_2 * (10^{2m2}) + z_1 * 10^{m2} + z_0$

Time complexity:

The total time taken,

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n)$$

↑ 3 multiplication recursive calls

↑ O(n) for addition & subtraction.

By the master theorem,

$a = 3, b = 2, d = 1. \rightarrow b^d = 2^1 = 2$

$\therefore a > b^d$

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

$\therefore \boxed{T(n) \in O(n^{1.585})}$

Hence time complexity is $O(n^{1.585})$

Test cases :- Inversions.

| | Choices | Number of inversions |
|---|---|---|
| 1) | [101, 102, 103, 104, 105, 106] | 0 |
| 2) | [101, 102, 103, 106, 105, 104] | 3 |
| 3) | [101, 102, 104, 105, 103, 106] | 2 |
| 4) | [101, 102, 104, 103, 105, 106] | 1 |
| 5) | [101, 102, 103, 104, 106, 105] | 1 |
| 6) | [101, 103, 104, 102, 105, 106] | 2 |
| 7) | [101, 102, 104, 106, 103, 105] | 3 |
| 8) | [102, 101, 103, 104, 105, 106] | 1 |
| 9) | [101, 104, 102, 103, 105, 106] | 2 |
| 10 | [101, 102, 104, 103, 105, 106] | 2 |