DAA Lab 4

Aim → To find count inversions of Course codes of 100 Students using brute force method and divide and conquer.

Write integer multiplication program for large numbers ~~with digits~~ of size 10, 50, 100, 500, 1000 using brute force and Karatsuba algorithm.

Count Inversions

1. Brute force

```
func count_inversion (arr[])
    count = 0
    n = length → arr
    for i from 0 → n-1
        for j from i+1 → n-1
            if arr[j] < arr[i]
                count += 1
    return count
```

- Time Complexity

1] Outer loop → 0 to n-1 → n times

2] Inner loop → i+1 → n-1 → (n-i+1) times

3] Comparision and increment of count → Constant time → ~~O(n)~~ O(1)

4] Return count → Constant time → O(1)

∴ Total iterations are n-i-1

∴ Total time = $\sum_{i=0}^{n} (n-i-1) = \frac{n(n+1)}{2} = \frac{n^2}{2} - \frac{n}{2}$

∴ $T(n) \propto n^2$

∴ Worst Case Time Complexity → $O(n^2)$

Best case and Average Case Time complexity are also $O(n^2)$ as the function traverse entire array ~~before~~ whether its sorted or not.

$$\therefore T(n) = O(n^2) + \underset{\substack{\downarrow \\ \text{comparision} \\ \text{and increment} \\ \text{Count}}}{O(1)} + \underset{\substack{\downarrow \\ \text{return}}}{O(1)}$$

2] Divide and Conquer using merge sort

```
Count = 0
~~fun~~
func merge (arr, l, r, m)
  ~~i~~ temp → [ ]
  i ← left
  j ← mid+1

  while i <= mid and j <= r
      if arr[i] <= arr[j]
          temp ← append (arr[i]), i += 1
      else
          temp ← append (arr[j])
          Count += (mid - i + 1)
          j += 1

  while i <= mid
    temp ← append (arr[i])
    i += 1
    while j <= r
        temp ← append (arr[j])
        j += 1

    for i ~~is~~ from left → right
        arr[i] = temp[i - left]

func ms (arr, left, right)
    if left == right
        return
    mid ← (left + right) // 2
    ms (arr, left, mid)
    ms (arr, mid+1 right)
    merge (arr, left, right, mid)

~~def~~ func count_inv (arr)
    ms (arr, 0, arr.length-1)
    return count
```

- Time Complexity

$T(n) = 2T(n/2) + O(n)$ ~~+ O(1)~~

$2T(n/2)$ because at every step we divide the subproblem in two halves and work on both halves recursively.

$O(n)$ → merge the two sorted arrays back

By using Master Theorem

$a = 1 \quad b = 2 \quad d = 1$

Since $d = \log_b a$

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

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

It then takes constant time c for comparing elements and increment the count

∴ The ~~tot~~ best case time complexity is ~~Θ~~ Ω (n logn) + c

Average case → Θ(n logn) + c

Worst Case → O(n logn) + c

∴ We see that Divide and Conquer ~~approach~~ approach is more optimized and useful than Brute force

Test Cases

• ~~#~~ Positive Test Cases

1)

Student ID	Courses
6	1,3,4,6
7	5,2,3,7
8	4,6,1,2
9	7,5,3,1

Expected Output is

Inversion Count 0 : ID's : [6]

Inversion Count 2 : ID's : [7]

Inversion Count ~~3~~ 4 : ID's : [8]

Inversion Count 6 : ID's : [9]

2)

Student ID	Courses
10	2,4,5,6
11	1,3,5,7
12	6,2,4,3
13	5,7,6,1

Expected Output:
Inversion Count 0: ID's [10, 11]
Inversion Count 4: ID's [12, 13]

3)

Student ID	Courses
14	4,5,6,2
15	3,1,7,6
16	5,2,4,1
17	7,6,5,3

Expected Output:
Inversion Count 2: ID's : [15]
Inversion Count 3: ID's: [14]
Inversion Count 5: ID's : [16]
Inversion Count 6: ID's : [17]

- Negative Test Case

1)

Student ID	Courses
A	1,2,3,
B	4,5,6,
C	2,3,4,
D	5,6,1,

Error: Invalid Student ID and one course code missing

2)

Student ID	Courses
21	
23	3,1,5
23	2,4

Error: No courses for Student 21
Inversion Count 0: ID's [23]
Inversion Count: 1: ID's [22]

3]

Student ID	Courses
24	4,5,6
25	1,,3
26	2,7,8
27	6,,5

Error: Courses missing for all Students

2] Integer multiplication

1] Brute force - method

```
function multiplylargenumbers (num1, num2)
    isneg ← false
    if num1 starts with '-'
        isneg ← true if isneg is false or vice verse
        remove '-' from num1
    if num2 starts with '-'
    ~~is~~ isneg ← false if its true or vice verse
        remove '-' from num2
    n1 ← length of num1
    n2 ← length of num2

    if num1 or num2 is "0"
        return "0"

    result [] ← size n1+n2 intialized
                              with 0

    ~~reve~~ reverse num1
    reverse num2

    for i from 0 to n1-1
        for j from 0 to n2 - 1
            ~~temp~~ = num1 [i] * num2 [j] by
            temp converting char to nums
            result [i+j] += temp
            result [i+j+1] += result [i+j] /10
            result [i+j] % = 10
    resultstr = ""
    leading zero = false
```

```
for i from result.size()-1 to 0
    if (result[i] == 0 and leadingzero)
        continue
    leadingzero ← false
    resultstr += (result[i] + '0')

if resultstr is empty
    return "0"
if isnegative is true
    resultstr = "-" + resultstr

return resultstr
```

- Time Complexity

- Sign Handling : O(1)
- Edge Case for zero : O(1)
- Initialization : O(n1+n2)
- Reversal of String : O(n1+n2)
- Digit by digit multiplication :

$$\sum_{i=0}^{n1-1} \sum_{j=0}^{n2-1} 1 = \sum_{i=0}^{n1-1} n2 = n1 * n2$$

$\therefore$ $T(n) = O(n1 * n2)$

- Converting Result vector to String : O(n1+n2)

$\therefore$ Overall Time Complexity is :

$T(n) = 2O(1) + 2O(n1+n2) + O(n1 * n2)$

$\approx O(n1 * n2)$ → Worst Case and Average Case

For Best Case, if both numbers have 1 digit then it takes constant time O(1)

2] Karatsuba Algorithm

```
function addstr(num1, num2)
    result ← ""
    carry ← 0
    i ← len(num1) -1
    j ← len(num2) - 1
    while i>=0 or j>=0 or carry>0
        ~~&~~ sum ← carry
        if i>=0
            sum += num1[i]
            ~~i+=~~ i--
        if j>=0
            sum += num2[j]
            j--
        ~~carry /=~~
        carry ← sum / 10
        result.append (sum%10)

    reverse result
    return result

function subtractstr(num1, num2)
    result ← ""
    borrow ← 0
    i = len(num1) -1
    j = len(num2) -1

    while i>=0 or j>=0
        sub ← borrow
        if i>=0
            sub += num1[i]
            i--
```

```
if j >=0
   sub -= num2[j]
   j--
if sub < 0
  sub += 10
  borrow ← 1
else
  borrow ← 0
result.append((char) sub)

remove leading zero from result
reverse result
if result not empty
    return result
else
   return "0"

function karatsuba(num1, num2)
    if num1 or num2 is "0"
        return "0"
    isneg ← false
    n1 = num1
    n2 = num2
    remove leading '-' from num1 or num2
    if exist

    if num1 or num2 has length 1
        return (isneg ? "-" : "") + multiply
                                      (num1, num2)
    n ← max of length of num1 and num2
    half ← (n+1) / 2

    split n1 into a and b at half
```

Test Cases

1] a = 9819807824 b = 9930247524
Expected ans = 139474904949031647776

2] a = -99999. b = 99999
Expected ans = -9999800001;

3] a = 1029384756 b = 5647382910
Expected ans = 15853829539268940160

4] a = 12345678901 b = 98765432109
Expected ans = 2245742925739027234209

5] a = 9876543210~~88~~12 b = 123456789012
Expected ans = 14451686800521024120144

6] a = 1234567890123 b = 9876543210987
Expected ans = 14449664511964140440401601

Conclusion

Hence we have seen how we can perform Count inversions on list of integers, more efficiently using merge sort with time $O(n \log n)$ instead of brute force with time $O(n^2)$ and we can perform large integer multiplication more efficiently with Karatsuba algo with time $O(n^{1.585})$ instead of brute force with time $O(n^2)$: