① m w w_1 w_2
m_1 w_1 w w_2
m_2 w w_2 w_1

② w m m_2 m_1
w_1 m_1 m_2 m
w_2 m_1 m m

after swap.

m w
m_1 w_1
m_2 ~~w_2~~ w_2

m ~~w_2~~ ~~m_1~~ w_2
m_1 w_1
m_2 w

Conclusion: You can see in the table ① and ②. using G-S matching. I get (m, w) (m_1, w_1) (m_2, w). after 1 swap the position m_1 and m in w row. then I get (m, w_2) (m_1, w_1) (m_2, w_2).

2.) Solution:
Each of them are Big O in increasing order below.

Justification: For log, the base case is does not matter when the compare each other, and log always smaller than n; also, when ~~α~~ $\alpha < \beta$, $\log_\alpha$ is big O $\log_\beta$ n polynomial and power function, polynomial function always is O (less) power)

$\log n!$ $\log 3^n$ $\sqrt{2}^{\log n}$ $\log^2 n$ $n^{\log\log n}$

$\log n$ $\sqrt{n}$ n $n\log n$ $\binom{n}{2} + n\log n$ n^2 n^3 n^{100}

$2^{\log^{1001} n}$ $4^{\log n}$ $2^{100\log n}$ $n \cdot 2^n$ 2^{2^n} $2^{2^{n+1}}$

Q3. prove the leaves exactly one more than nodes which have two childsren.

Base: let node = 1, the depth of a node is the number of edges from the root to the node, so there is one leave, the leave is exactly one more than the node.

Induction: Assume that the number of nodes = k, for all trees of size n nodes. There are 2 cases.

case1: the root has exactly 1 child, there is only one leave in the end, that is exactly one more than the nodes with two children

case2: root has 2 children, T_1, T_2, we know that $T_1 = T_2 =$ Leave -1, For the tree, the two children T is $T_1 + T_2$ + the root, that is L_1 (noted by leave 1) -1 + L_2 (noted by leave 2) -1. therfore the total nodes that with two children have number of leave, L (noted by leaves) -1.

```
def sumOfZero(n):


  for i in range(len(n)):
    for j in range(i+1, len(n)):  # the complexity of inner for loop
is BigO^2
        sumOfTwo = -(n[i] + n[j])

        if sumOfTwo in n: # the complexity of checking is log(n)
            print(sumOfTwo, 'sumoftwo')
            return n[i], n[j], sumOfTwo
        else:
            return "no"


numbers = [int(x) for x in input("Enter numbers: ").split()]
numbers.sort()    # the complexityof this sort is O(n* log n)
# sorting the numbers input
result = sumOfZero(numbers) # calling the function
if(result):
    print("Yes, it has 3 numbers whose sum is zero", result)
else:
    print("No")
```

```
def zerosumfaster(A):
   ht = dict()
   for i in range(len(A)):
        h = A[i]
        ht[h] = h
   print(ht)
   print(ht[A[0]])
   for i in range(len(ht)):
     for j in range(i+1, len(ht)):  # the complexity of inner for loop
is BigO^2

        keySum = - (ht[A[i]] + ht[A[j]])

        if keySum in ht:  # this is will take a content time.
           return keySum, ht[A[i]], ht[A[j]]
        else:
           return 'no'



numbers = [int(x) for x in input("Enter numbers: ").split()]
numbers.sort()

# sorting the numbers input
result = zerosumfaster(numbers) # calling the function
if(result):
    print("Yes, it has 3 numbers whose sum is zero", result)
else:
    print("No")
```