Fibonacci sequence:

```
def fib_seq(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fib_seq(n - 1) + fib_seq(n - 2)


number = int(input("Enter the number for generating fibonacci sequence: "))

result = fib_seq(number)
print(f"Fibonacci sequence of ({number}): {result}")
```

Function call stack:

Initial Call: fib_seq(5)
Calls fib_seq(4) + fib_seq(3)

Call to fib_seq(4):
Calls fib_seq(3) + fib_seq(2)

Call to fib_seq(3):
Calls fib_seq(2) + fib_seq(1)

Call to fib_seq(2):
Calls fib_seq(1) + fib_seq(0)

Call to fib_seq(1):
Returns 1

Call to fib_seq(0):
Returns 0

Back to fib_seq(2):
Returns fib_seq(1) (which is 1) + fib_seq(0) (which is 0)
Sum is 1 + 0 = 1

Back to fib_seq(3):
Returns fib_seq(2) (which is 1) + fib_seq(1) (which is 1)
Sum is 1 + 1 = 2

Back to fib_seq(4):

Returns fib_seq(3) (which is 2) + fib_seq(2) (which is 1)
Sum is 2 + 1 = 3

Back to fib_seq(5):
Returns fib_seq(4) (which is 3) + fib_seq(3) (which is 2)
Sum is 3 + 2 = 5

Call stack for fib_seq(5):

fib_seq(5) -> fib_seq(4) -> fib_seq(3) -> fib_seq(2) -> fib_seq(1) -> fib_seq(0)

The values returned by each call contribute to the final result of 5 for fib_seq(5)

Problem 1 : Merge K sorted arrays

Code uploaded in the below link:
https://github.com/Saranya-Muralidharan0802/DAA_development/tree/main/HandsOn_4

Time Complexity analysis for Problem 1:

- Let K be the number of arrays, and N be the size of each array.
- The worst-case time complexity is $O(K . N^2)$.
- In the worst case, for each element in the result, the algorithm might iterate through all K arrays to find the minimum element.
- An optimized approach using a priority queue (min-heap) can reduce the time complexity to O(K.N.log(K))

Possible Improvements:

1. Use of Priority Queue
   - Implementing a min-heap (priority queue) can significantly improve the time complexity.
   - By maintaining a priority queue of size K, the minimum element from each array can be efficiently retrieved.

2. Divide-and-Conquer Approach:
   - For large input arrays, consider a divide-and-conquer approach like merge sort.
   - Merge pairs of arrays first and then continue merging until the final result is achieved.

Problem 2: Remove Duplicates from Sorted Array

Code uploaded in the below link:
https://github.com/Saranya-Muralidharan0802/DAA_development/tree/main/HandsOn_4

Time Complexity analysis for Problem 2:

- Let N be the size of the sorted array.
- The time complexity is O(N) as the algorithm iterates through the array once.

Possible Improvements:

1. In-Place Modification:
   - If the input array can be modified in-place, consider optimizing the algorithm without using additional space.
   - Remove the need for creating a new array to store the result.