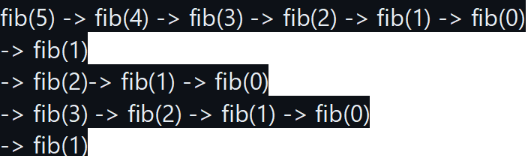
Implement the Fibonacci sequence:

GitHub link for Implementation of python code for Fibonacci sequence and for Fibonacci sequence for fib(5) and listed the stack overflow that I have observed. –

[DAA\_algorithms/Hands\_on\_4/Fibonacci Seq.py at main · renati81/DAA\_algorithms (github.com)](https://github.com/renati81/DAA_algorithms/blob/main/Hands_on_4/Fibonacci%20Seq.py)

The complete call stack, showing all recursive calls in the order they are made, is: –



**Problem – 1**:

1. GitHub link for implementation of sorted array with examples – [DAA\_algorithms/Hands\_on\_4/Merge\_sorted\_array.py at main · renati81/DAA\_algorithms (github.com)](https://github.com/renati81/DAA_algorithms/blob/main/Hands_on_4/Merge_sorted_array.py)
2. Time complexity for the above sorted array –

-Time complexity for number of levels recursion is O(log₂M) and for merging the sorted array is O(M).

-Total time complexity = (Number of levels) \* (time sorted per level)

T(M) = O(log₂M) \* O(M).

T(M) = O (M log₂M)

Here, we are calculating in terms of N and K (rows and columns)

So, T (K, N) = O ((K \* N) \* log₂ (K \* N))

3.Improved approaches –

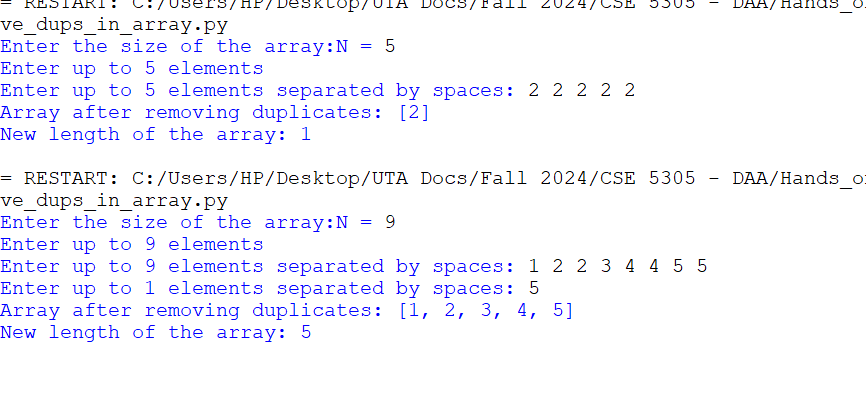
Implement the min-heap with optimized priority queues or libraries like Python’s heapq. Store only necessary data (e.g., tuples of values and indices) to minimize memory overhead. Merge arrays in batches or use multi-threading to speed up the process. For very large data, use external storage techniques or distributed computing frameworks. Apply adaptive algorithms and in-place merging to reduce unnecessary operations and memory usage.

**Problem – 2**:

1. GitHub link for implementation of sorted array with examples –

[DAA\_algorithms/Hands\_on\_4/Remove\_dups\_in\_array.py at main · renati81/DAA\_algorithms (github.com)](https://github.com/renati81/DAA_algorithms/blob/main/Hands_on_4/Remove_dups_in_array.py)

Output –



1. Time complexity for the above remove duplicates sorted array –

Here are two case as below

1. If array is sorted – The time complexity of input collection is O(N) and for removing duplicates is O(N). So, the combined time complexity of removing duplicates for the sorted array is O(N).
2. If array is unsorted – The time complexity of input collection id O(N), for sorting the array O(N logN) and for removing duplicates is O(N). So, the combined time complexity of removing duplicates for an unsorted array is O(N logN).
3. Improved approaches:

Couldn’t think of any other approach as this is both space and time efficient, as only one array is used, have modified the input array to form the output. So, I assume the approach is efficient.