1. Problem 1 : Fibonacci Series

The function calls itself recursively. When the function encounters 0/1 value, it is returned to the function that called it & so on.

Function that makes the call			value that returned
fib (5)	n= 5	=>	fib (4) + fib (3)
fib (4)	n=4	=>	fib (3) + fib(2)
fib (3)	n=3	=>	fib(2) + fib(1)
fib(2)	n= 2	=>	fib(1) + fib(0)
fib(1)	n=1	=>	if (n==1) => return 1
Value 1 returned to fib(2)			
fib(2)	n=2	=>	1 + fib(0)
fib(0)	n=0	=>	if (n==0) => return 0
fib(2)	n=2	=>	1 + 0 = 1
fib(3)	n = 3	=>	fib (2) + fib(1) = 1 + fib(1)
fib (1)	n=1	=>	return 1
fib (3)	n=3	=>	1 + 1 = 2
fib(4)	n=4	=>	2 + fib(2)
fib (2)	n=2	=>	fib (1) + fib (0) = 1
fib (4)	n=4	=>	2+ 1 = 3
fib (5)	n=5	=>	fib(4) + fib (3) = 3 + fib(3)
fib(3)	n=3	=>	2
fib(5)	n=5	=>	3 + 2 = 5

2. Problem 2 : Merge Arrays

a) Code uploded in Github

b) Time complexity of the function Sorting :

Since there is • 1 for loop that runs for the length of array 'n'
• 1 while loop that runs also runs 'n' times,

The time complexity is n2. $\boxed{\oplus(n^2)}$

But the program also contains the part of building the array using user input.
If the number of array 'k' & length of array 'n' is too large, the array building part has to be taken into consideration.

∴ For large k & n : $\boxed{\oplus(k \times n + n^2)}$

c) Ways to improve my implementation :

1. I could have used 'merge sort' to sort the unified array, it would have reduced the time complexity
2. If the input arrays are larger in size, concatenating them would take longer. So instead of concatenating them, a recursive sorting algorithm for each array would be better.

3. Problem 3 : Remove duplicates

a) Code uploaded in Github.

b) Time Complexity : The function has 2 for loops
1st for loop pick one element & the 2nd for loop is used for comparing the 1st element to all the other elements in the array.
∴ The time complexity is $\boxed{\Theta(n^2)}$

c) Ways to improve my implementation :

Instead of using 2 for loops to iterate through all the elements, I could have used 1 for loop to iterate through the elements & compared it with the input array to pick out the unique elements alone.
This would have kept the time complexity to $\Theta(n)$
& would have produced a much simpler code.