Problem 0

1.list out the the function call stack

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

fib(1) -> fib(0)

fib(2) -> fib(1) -> fib(0)

-> fib(0)

fib(3) -> fib(2) -> fib(1) -> fib(0)

fib(1) -> fib(0)

-> fib(1)

2. Prove the time complexity of the algorithms.

For the equation the time complexity will be

T(n)=T(n-1)+T(n-2)+1

For each value of n, the function makes two recursive calls with parameters n-1 and n-2. This branching continues until it reaches n = 0 or n = 1, at which the recursion stops. The number of function calls grows exponentially with the value of n because each function call results in two more functions $O(2^n)$

3. Comment on way's you could improve your implementation

We can improve this by storing the values of fib(4), fib(3)……. Where these values can be reused instead of recalculation and calling all the below functions

Problem 1

T(1)=1

T(n)= T(n-1)+ c

T(n)= T(n-2)+c

T(n)= T(n-k)+kc

When n-k=1

K=n-1

=n+1

So time complexity will be O(n)

Comment on way's you could improve your implementation

To improve the time complexity of the factorial calculation, we can use an iterative approach instead of recursion. Recursion can lead to stack overflow errors for large input values of n, This directly calculates the factorial of n by multiplying all integers from 1 to n. and eliminates the need for recursive function calls

Problem 2

T(n)= T(n-1)+ T(n-2)+c

Assume T(n-1) =T(n-2) as both are nearly close and similar

T(n)=2 T(n-1)+c

Assume $f(n)=O(2^n)$

Substituting the above in original eqn

$2\ T(n-1)+c <= 2k.2^{n-1} +c$

$=k.2^n+c$

Therefore the time complexity is $O(2^n)$

Comment on way's you could improve your implementation

Instead of comparing each element with its previous element to check for duplicates, use binary search to find duplicates efficiently in the sorted array