TEJESH. KOLLAPANENI
1002208329

Hands On-4

Problem 0.

1. List out 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(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 & 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 calls results in two more functions $O(2^n)$

3. 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 $O(n)$

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

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

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

Substituting the above in original Eqn

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

$= K \cdot 2^n + c$

$\therefore$ the time complexity is $O(2^n)$

Improvement:

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