Hands on - 4

Problem - 0

Implement the fibonacci series and debug the code.

```
#include <stdio.h>
int fib(int n)
{
   if (n==0)
      return 0;
   else if (n==1)
      return 1;
   else
    return fib(n-1) + fib(n-2);
}
int main ()
{
 int result;
 result = fib(5);
 return 0;
}
```

If we analyse the above algorithm and pass the 'n' value as 5, the following are the recursive calls and function call stack.

fib(5) ; n = 5 ; fib(5) calls fib(4) and fib(3)

fib(4) ; n = 4 ; fib(4) calls fib(3) and fib(2)

fib(3) ; n = 3 ; fib(3) calls fib(2) and fib(1)

fib(2) ; n = 2 ; fib(2) calls fib(1) and fib(0)

fib(1) ; n = 1 ; fib(1) returns 1

fib(0) ; n = 0 ; fib(0) returns 0

fib(0) returns 0 fib(1) returns 1

fib(2) ; n = 2 ⇒ 1 + fib(0)

fib(0) ; n = 0 ⇒ returns 0

fib(2) returns 1 + 0 = 1 # the sum of fib(1) & fib(0)

fib(3) ; n = 3 ⇒ fib(2) + fib(1)

⇒ 1 + fib(1)

fib(1) returns 1

fib(3) returns 1+1 = 2 # the sum of fib(2) & fib(1)

fib(4) ; n = 4 ⇒ 2+ fib(2)

fib(2) ; n = 2 ⇒ fib(1) + fib(0) = 1

fib(4) returns 2 + 1 = 3 # the sum of fib(3) & fib(2)

fib(5) ; n = 5 ⇒ fib(4) + fib(3)

= 3 + fib(3)

fib(3) ; n = 3 ⇒ returns 2

fib(5) ; n = 5) ⇒ 3 + 2 = 5 # the sum of fib(4) & fib(3)

∴ Hence, fib(5) = 5 by using the above recursive calls.

Problem - 1 : Merge Arrays :

a) code uploaded in Github

b) Time Complexity : for the function mergekarrays Since there is 1 for loop, it iterates over each array in the arrays list and merges them using merge function. So, it runs 'n' times.

So, time complexity is $O(n)$

Since, we are iterating over 'K' arrays and merging them one by one. The overall time complexity will be given as :

$$T(n) = \Theta(K * n)$$

c) ways to improve implementation.

- The current algorithm involves repeatedly appending elements to the list. This could increase the need of memory reallocation. So, instead of this, I could have created the space for the merged array.

- If the input arrays are larger, appending or concatenating them would be critical. So, we can implement a recurisive Sorting algorithm or Consider optimising the input reading process.

Problem - 2 : Remove _ duplicates

a) code has been uploaded in Github.

b) Time complexity :

the for loop runs from 0 to n-1 where 'n' is length of input array. In each iteration, the loop checks the current element different from next.

So, here time complexity is $\Theta(n)$

The 'if' runs in constant time for each iteration.

ie : $\Theta(1)$

the dominant factors in time complexity is 'n' and the small factors are negligible.

So, time complexity = $\Theta(n)$

c) Ways to improve implementation :

I could have used more efficient and simpler algorithm. Though it has time complexity $\Theta(n)$. we can achieve linear time complexity by comparing adjacent elements and copying the different elements to the result array by avoiding duplicates-