DAA

UNIT-1

Assignment-1

K.Nithin Kumar
192210164.

① The Fibonacci numbers, Commonly denoted F(n) form a sequence, Called the fibonacci Series, such that each number is the sum of the two preceding ones, Starting from 0 and 1, that is

$F(0) = 0, f(1) = 1$

$F(n) = F(n-1)+F(n-2)$. for $n>1$

Ⓐ The given problem is asking to Calculate the nth fibonacci number given the recursive formula:

$F(n)=F(n-1)+F(n-2)$

With initial Conditions $F(0)=0$ and $F(1)=1$,

Here's a simple python code to Calculate F(n) for the given n;

```
      def fibonacci(n):
          if n==0:
        return 0
        elif n==1;
            return 1
    else:
        return fibonacci(n-1)+fibonacci(n-2)
# Example usage:
      n=2
    result = fibonacci(n)
Print("output:", result).
```

For the given example where n=2, the output will be 1, as

$F(2)=F(1)+F(0)=1+0=1$

2) Given the head of a linked list, reverse the nodes the list k at a time, and return the modified list. k is a positive integer and is less than or Equal to the length of the linked list. If the number of nodes is not a Multiple of k then left-out nodes, in the end, should remain as it is. You May not alter the values in the list's nodes, only nodes themselves May be changed.

Ⓐ The implement a function to reverse k nodes at a time in a linked list:

```
        class Listnode;
            def __init__(self, val=0, next=none);
          self.val = val
          self.next = next

    def reverse_k_group(head, k):
        def reverse_list(node):
            prev, curr = none, node
          while curr:
              next_node = curr.next.
```

```
curr.next  =prev
  prev = curr
  curr = next_node
return prev

  def get-length (node):
        length = 0
    while node:
        length += 1
        node = node.next
  return length.
length = get-length (head)
dummy = List Node (0)
dummy. next =head
prev-group_end  =dummy

   for_ in range (length//k):
group_start =prev_group_end_next
group_end  =group_start
   for_ in range (k-1):
       group_end = group_end . next.
 next_group_start = group_end . next
   group_end. next =None.
prev_group_end. next = reverse_list (group_start)
```

```
group _start. next =next group_start
prev_group_end =ground. start

    return dummy. next.
```

NOTE :- Make sure to define the 'List Node' class before using this code, or adjust it based on your existing Implementation of linked lists.

5 Given a string expression of numbers and operators return all possible results from Computing all the different possible ways to group numbers and operators. You May return the answer in any order. the test Cases are generated such that output values fit in a 32-bit integer and the number of different results doesnot exceed 104.

A To Solve this problem, we can use a recursive approach to generate all possible ways to group numbers and operators. Here's a python Implementation.

```
      def diff_ways_to_Compute :
         def Calculate (op, left, right):
            if op == '+':
               return left + right.
```

```
        elif op =='-':
            return left - right
        elif op =='*';
            return left + right.
    def helper (start, end):
        result = [ ]
    for i in range (start, end):
        if expression[i] in{'+','-','*'}:
            left_results = helper (start, i)
        right_results = helper (i+1, end)
     for left in left_results :
       for right in right_results:
           result.append (calculate (expression[i], left,
                                     right))
    if not result :
        result.append (int (expression[start:end]))
    return result.
    return helper (0, len (expression1)
Expression = "2-1-1"
result = diff_ways_to_Compute.
Print ("output:", result).
```

In this Implementation, the 'diff_ways_to_compute' function takes an input expression and uses a helper function to recursively break down the expression into smaller sub problems, considering all possible ways to group number and operations.

The expression "2-1-1", the output will be '[0, 2]', representing the different possible results.

④ You are given a positive integer primefactors. You are asked to construct a positive integer n that satisfies the following conditions:

The number of prime factors of n (not neccessarily distinct) is at most prime factors.

The number of nice divisors of n is Maximized. note that a divisor of n is nice if it is divisible by every prime factors of n. for example, if n=12 then it prime number of factors [2,2,3], then 6 and 12 are nice divisors, while 3 and 4 are not.

Return the number of nice divisors of n. since that number can be too large, return it modulo 10^9+7.

def Can use dynamic programming to find the number of nice divisors for a given number n with a certain count of prime factors

Here's a python Implementation.

```
def Count-nice-divisors :
  MOD = 10**9+7

def power (x,y):
    result = 1
    while y > 0;
        if y % 2 ==1:
            result = (result * x) % MOD
        x = (x* x) % MOD
        y // = 2
    return result.

 def dfs(prime factors, Count, memo):
     if Count==1:
        return 1
    if (count , memo) in memo;
        return memo[(count, memo)]
  total = 0
  for i in range (1, count//2+1):
     total = (total + dfs (prime factors, i, memo)
                * dfs (Primefactors, Count-i ,memo))%
                                              MOD
   Memo [(count ,memo)] = total
     return total.

 result = 1
  for factor in prime factors:
```

```
result = (result* power(factor, dfs (prime factors,
              prime factors [factor], {} ·1))) % MOD

 return result.

Prime factors = 5
result = Count_nice-divisors (prime factors)
Print ("output:", result)
```

For the given example with 'prime factors=5" the output will be the number of nice divisors of n modulo 10^9+7.