1. Given a positive integer \( N \), we want to return \( N^r \) which is the integer corresponding to the reverse of the base 10 representation of \( N \). For example, if \( N = 5381 \), then \( N^r = 1835 \). Complete the following Ocaml program that reverses the input integer.

```
exception Negative ;;
let rec appenddig (n, rev) = if n <= 9 then (rev*10 + n)
else appenddig (n/10 , ___rev*10 + n mod 10___) ;;
let revint = function n -> (* main function *)
    if n <= 0 then raise Negative
    else appenddig (__n , 0__) ;;
```

(i) Type of function \( \text{revint} : \text{int} \rightarrow \text{int} \)

(\( \text{appenddig} : \text{int} \times \text{int} \rightarrow \text{int} \) \( 3 \) marks if initialization was correct. No partial marks for the types.

(ii) To prove the correctness of the above program, we must characterise the behavior of the function \( \text{appenddig} \). Formally specify the the desired relation between the two parameters of \( \text{appenddig} \) that will imply the correctness of \( \text{appenddig} \). (Proof is not required).

If \( n = a_k a_{k-1} \ldots a_1 \) then after \( i \) calls to \( \text{appenddig} \),
\( n = a_k a_{k-1} \ldots a_i \) and \( \text{rev} = a_1 a_2 \ldots a_{i-1} \).

An equivalent answer is for input \( N \), \( n \) concatenated with \( \text{revint} (\text{rev}) = N^r \).

A description of the program, or a recurrence relation won’t fetch any marks.

2. Write a recursive program \( \text{zip}(l_1, l_2) \) that given two lists \( l_1 \) and \( l_2 \), produces a single list whose elements are pairs of corresponding elements from \( l_1 \) and \( l_2 \). For example \( \text{zip}([1;2;3], [4;5;6]) \) returns \( [(1,4); (2,5); (3,6)] \). The program should raise an exception \( \text{UnequalLengths} \) if the lists are of unequal length.

```
exception UnequalLengths;;
let rec zip (l1, l2) = match l1 with
```

(\( 2 \) marks if equality check was correct. No credit if that was not asked for, so no credit for that.)
3. Given a positive integer $N$, define an ordered partition as an ordered sequence $(a_1, a_2, \ldots, a_k)$ of positive numbers $\leq N$ ($0 < a_i \leq N$) such that $a_1 + a_2 + \cdots + a_k = N$. For example $(3,1,2,1,2)$ is an ordered partition of 9. Note that the ordered partitions $(3,1,2,1,2)$ and $(3,2,2,1,1)$ are different because the order of the numbers are different. Given a number $N$ we would like to count how many ordered partitions it has. For example, $N = 4$ has 8 ordered partitions:

- Write all the ordered partitions of 6.
  There are 32 of them:
  
  - (6)
  - (1,1,1,1,1,1)
  - (1,2,1,1,1) : 5 permutations
  - (1,3,1,1) : 4 permutations
  - (1,4,1) : 3 permutations
  - (1,5), (5,1)
  - (2, 2, 1, 1) : 6 permutations
  - (2,3,1) : 6 permutations
  - (3,3)
  - (4,2), (2,4)
  - (2,2,2)

  \[ P_{1} = 1 \]

  \[ P_{N} = P_{N-1} + P_{N-2} + \cdots + P_{1} + 1 \]

  This implies that if $N > 1$, then $P_{N} = 2P_{N-1}$.

- Use this inductive definition to write a program in Ocaml which outputs the number of partitions of $N$. 

\[ (x,y)\bowtie (\text{zip}(xs,ys))\]
exception Nonpositive ;;

(* Type of function count_partitions ___int__________ -> _____int___________ *)

let rec count_partitions = function n ->
  if (n <= 0) then raise Nonpositive
  else if (n = 1) then 1
  else 2*count_partitions(n-1);

An alternate solution is to write a function directly from the recurrence (without simplifying).

let rec partition n =
  let rec sumpart (n , res) = if n = 0 then res else
    sumpart (n-1 , res + partition n)
  in  if n =1 then 1 else (1 + sumpart (n-1, 0) ) ;;