**Problem 1 : Easy**

**Problem: 1 Tower of Hanoi with Numerical Representation**

You are given three rods labeled A, B, and C, and a number of disks of different sizes which can slide onto any rod. Initially, all the disks are stacked in ascending order of size on rod A, with the smallest disk being the topmost (e.g., disk 1 is the smallest, disk 2 is larger than disk 1, and so on).
The objective of the puzzle is to move the entire stack of disks from rod "***A***" to rod "***C***", obeying the following rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the top disk from one rod and placing it onto another rod.
3. No disk may be placed on top of a smaller disk.

Your task is to write a program to solve the Tower of Hanoi puzzle for a given number of disks and to print the sequence of moves required to solve the puzzle optimally, using the numerical representation of the disks (e.g., moving disk 1, disk 2, etc.).

***Solution :***

```
#R Programme To SOlve Tower of Hanoi
hanoi = function(n, from_rod, to_rod, aux_rod) {
  if (n == 1) {
       cat("Move disk 1 from rod", from_rod, "to rod", to_rod, "\n")
       return()
  }
  hanoi(n-1, from_rod, aux_rod, to_rod)
  cat("Move disk", n, "from rod", from_rod, "to rod", to_rod, "\n")
```

```
  hanoi(n-1, aux_rod, to_rod, from_rod)
}

num_disks = 3
cat("Sequence of moves:\n")
hanoi(num_disks, "A", "C", "B")
```

**2**. In preparation for a month-long expedition to the Himalayas, a team of climbers is selecting equipment to carry. Each item of gear is characterized by two key attributes: its weight and its value to the expedition (measured in terms of utility and necessity). Due to the arduous nature of the journey, there's a strict limit on the total weight that can be carried.

Implement an algorithm to maximize the total value of the selected gear without exceeding the weight limit of the backpack.

*Solution:*

```
# Define the function to solve the knapsack problem

knapsack <- function(values, weights, maxWeight) {

  numberOfItems <- length(values)

  # Create matrix to store the maximum value that can be obtained for every given number of items and weights

  dpMatrix <- matrix(0, nrow = numberOfItems + 1, ncol = maxWeight + 1)



  # Populate the dpMatrix

  for (i in 1:(numberOfItems + 1)) {

    for (w in 1:(maxWeight + 1)) {

      if (i == 1 || w == 1) {

        dpMatrix[i, w] <- 0

      } else if (weights[i-1] <= w) {

        dpMatrix[i, w] <- max(dpMatrix[i-1, w], dpMatrix[i-1, w-weights[i-1]] + values[i-1])

      } else {

        dpMatrix[i, w] <- dpMatrix[i-1, w]

      }

    }

  }

  # The bottom-right corner of the matrix will have the answer
```

```
  return(dpMatrix[numberOfItems + 1, maxWeight + 1])
}

# Example usage
values <- c(6, 10, 12)
weights <- c(2, 2, 3)
maxWeight <- 5

# Solve the knapsack problem
maxValue <- knapsack(values, weights, maxWeight)
print(paste("Maximum value that can be achieved: ", maxValue))
```

## Bsb