STAT2005 Programming Languages for Statistics
Exercise for Chapter 5

1. (a) Risky has created a new non-negative integer sequence called Risky Sequence. Risky Sequence $\{R(n)\}$ is defined by

$$R(n) = \begin{cases} 1, & \text{for } n = 0 \text{ or } 1, \\ 4R(n-1) - 3R(n-2), & \text{for odd } n > 1, \\ 3R(n-1) - R(n-2), & \text{for even } n > 1. \end{cases}$$

Write function `R(n)` to return the n^{th} term in Risky Sequence by recursion.

(b) Padovan Sequence $\{P(n)\}$ is the an integer sequence defined by the recurrence relation

$$P(n) = P(n-2) + P(n-3),$$

with initial values $P(0) = P(1) = P(2) = 1$. Write function `P(n)` to return the n^{th} term in Padovan Sequence by recursion.

(c) Moser-de Bruijn sequence $\{S(n)\}$ is defined by the recurrence relation

$$S(2n) = 4S(n) \text{ and } S(2n+1) = 4S(n) + 1,$$

with initial values $S(0) = 0$ and $S(1) = 1$. Write function `S(n)` to return the n^{th} term in Moser-de Brujin Sequence by recursion.

2. (a) Company A has n tasks for employees every day. Each task has a corresponding number i, ranging from 1 to n. An employee taking part in task i can earn i dollars. Each employee is initially assigned with a task k, where $1 \le k \le n$.

To prevent employees from getting too tired, after taking part in task i, there is a probability of 0.2 and 0.3 to unlock the tasks $2i$ and $2i+1$ respectively for an employee if the tasks exist. Note that the two events are independent. As the employees are enthusiastic in work, if a task is unlocked, they will take part in it.

Let users input n and k, write function `expected(x)` to calculate the expected earning for an employee in a day by recursion and calculate the answer of `expected(k)`.

Here are some examples.

If $n = 5$ and $k = 1$, $expected(1) = 1 + 0.2 \times (2 + 0.2 \times 4 + 0.3 \times 5) + 0.3 \times 3 = 2.76$

If $n = 24$ and $k = 6$, $expected(6) = 6 + 0.2 \times (12 + 0.2 \times 24) + 0.3 \times 13 = 13.26$.

If $n = 100$ and $k = 3$, $expected(3) = 17.0811$

(b) Company B also has n tasks for employees every day. Each task has a corresponding number i, ranging from 1 to n. An employee taking part in task i can earn d_i dollars. Every task i has a stamina cost c_i, meaning that taking part in task i costs an employee c_i stamina. $d_1, d_2, \dots, d_n$ are stored in vector `d` and $c_1, c_2, \dots, c_n$ are stored in vector `c`.

An employee initially has $2n$ stamina. When the stamina becomes 0, the employee cannot work. If the remaining stamina is smaller than c_i, the employee cannot take part in task i. By exhausting every combination of tasks choices using recursion, write

`maxearn(d,c,remainingstamina)` to calculate the maximum possible amount of money an employee can earn.

For example, if $n = 10$, $d = \{1, 3, 7, 6, 8, 9, 2, 4, 4, 8\}$ and $c = \{5, 1, 2, 2, 4, 1, 3, 6, 3, 7\}$, an employee can choose task 2, 3, 4, 5, 6, 9, 10 to earn 45 dollars by using 20 stamina. You may use this example to test your solution.

3. Player 1 and Player 2 are playing a game. In this game, there are n toys on the table, where $n \geq 1$ is any positive integer. This game starts with player 1 and both players take turns to take action. For each action, a player can take 1 to $\min(5, \#\text{ of remaining toys})$ toys away from the table. When a player takes the last toy away from the table, the player wins.

(a) Write R codes to let user input n with the prompt "`Enter the number of toys:`". If the user's input is not a positive integer, print "`Please enter a positive integer!`" and ask him to input again until it is correct.

(b) Write R codes to simulate the whole game.

Program Flow:

1. Input n as described in (a) and begin with player 1.
2. Ask the corresponding player to enter the number of toys to be taken away from the table with prompt "`The current number of toys on the table is i. Player j, please enter the number of toys to be taken away (between 1 to k):`", where i should be the current number of toys on the table, j should be either 1 or 2 depending on which player's turn it is, k is $min(5, i)$. When the player enters an invalid number, the program shall display a warning message: "`Invalid input!`". This phase finishes this when the player inputs correctly.
3. Repeat Step 2 until there is no toys left on the table.
4. Display the messages "`Player 1 wins!`" or "`Player 2 wins!`" accordingly.

1(a)

```
R <- function(n) {
    if (n==0 || n==1) {
        return(1)
    }
    if (n %% 2 == 1) {  #odd
        return(4*R(n-1) - 3*R(n-2))
    }
    else {  #even
        return(3*R(n-1) - R(n-2))
    }
}
```

best do n<=1

```
# Q1b
P <- function(n){
  if (n<=2) # n is an integer>=0
    return (1)
  return (P(n-2)+P(n-3))
}

# Q1c
S <- function(n){
  if (n<=0) # n is an integer>=0
    return(0)
  if (n==1)
    return(1)
  if (n%%2==0) # even n
    return(4*S(n/2))
  return(4*S(n%/%2)+1) # odd n
}
```

2(a)

```
expected <- function (k) {
    if (k > n) {
        return (0)
    }
    return (k + 0.2 * expected (2 * k) + 0.3 * expected (2 * k + 1))
}
```

(b)

```
maxearn <- function(d,c, remainingstamina) {
    num_jobs <- length(d)
    temp_earn <- d()
    for (i in 1:numjobs) {
        if ( remainingstima >= c[i] ) {
            temp_earn <- c(tmp_earn, d+ maxearn ( d[i+1:num_jobs],
                                                  c[i+1:num_jobs],
                                                  remainingstamina - c[i] )
        }
    }
    return ( max ( temp-earn) )
}
```

3(a)

```
repeat {
    n<- readline(prompt= "Enter the number of toys : \n")
    if ( class(n) != "integer" || n <=0) {
        cat ("Please enter a positive integer !" )
    }
    else {break}
}
```

yields to as.numeric

(b)

```
player ← 1

repeat {

    # read input and check
    sprintf( " The .... is %d, Player %d, ... 1 to %d \n", n, player, min(5, n))
    inp ← as.numeric( readline())

    repeat {
        if (inp < 1 or inp > min(5, n) ) {
            cat c(" Invalid input !")
        } else { break }
    # chek end
    n ← n - inp
    if ( n == 0 ) {
        sprintfc(" Player %d wins !", player )
        break
    }

# update
    if (player == 1) player ← 2 else player ← 1

}
```

```r
# Q2a
n <- as.integer(readline(prompt="Input n: "))
k <- as.integer(readline(prompt="Input k: "))
expected <- function(x){
  if (x*2>n) return(x)
  if (x*2+1>n) return(x+0.2*expected(x*2))
  return(x+0.2*expected(x*2)+0.3*expected(x*2+1))
}
expected(k)

# Q2b
maxearn <- function(d,c,remainingstamina){
  n <- length(d)
  if (n==1) # base case
    return(ifelse(remainingstamina>=c,d,0))
  if (remainingstamina>=c[n])
  # if there is enough stamina,
  # compare the money earned by the two options
  # (choosing this task or not)
    return(max(
      d[n]+maxearn(d[1:(n-1)],c[1:(n-1)],remainingstamina-c[n]),
      # pick up task n
      maxearn(d[1:(n-1)],c[1:(n-1)],remainingstamina)
      # do not choose this task
    ))
  return(maxearn(d[1:(n-1)],c[1:(n-1)],remainingstamina)) # not enough stamina
}
```

```r
# Q3a
repeat{
  n <- as.numeric(readline(prompt="Enter the number of toys: "))
  if (is.na(n)) cat("Please enter a positive integer!")
  # check characters
  else if (n<=0) cat("Please enter a positive integer!")
  # check non-positive number
  else if (n-floor(n)>0) cat("Please enter a positive integer!")
  # check floats
  else break
}

#Q3b
i <- n # i stores the current number of toys on the table
j <- 2 # j represents which player's turn
while (i>0){
  j=(2-j)%%2+1 # switch turn
  repeat{
    input <- as.numeric(readline(prompt=
    cat("The current number of toys on the table is",i,". Player",j,", please en

    # check whether the input is valid
    if (is.na(input)) cat("Invalid input! ")
    else if (input<=0|input>min(5,i)) cat("Invalid input! ")
    else if (input-floor(input)>0) cat("Invalid input! ")
    else break # break if it is valid
  }
  i=i-input # update the number of toys on the table
}
if (j==1){
  print("Player 1 wins!")
} else print("Player 2 wins!")
```