Q1. (15%) The Fibonacci sequence is famous in mathematics. The sequence is defined as

$$F_1 = 1, F_2 = 1, F_n = F_{n-1} + F_{n-2} \text{ for } n > 2$$

(a) Write a *while*( ) loop to find the first Fibonacci number $k >$ greater than 100.

(b) For the number F_n=k in (a), what is the index n?

(c) Use *for*( ) to write a function *print.Fib* that takes an integer k as its input and prints **ALL** Fibonacci numbers <= k. Test k=100 and show me the results.

```
> #Q1. (10%) The Fibonacci sequence is famous in mathematics.
> # (a) Write a while( ) loop to find the first Fibonacci number k > greater than 100.
> fib=function(n){
+   if(n>2){
+     return(fib(n-1)+fib(n-2))
+   }else{
+     return(1)
+   }
+ }
>
> n=1
> while(fib(n)<=100){
+  n=n+1
+ }
> print(fib(n))
[1] 144
```

```
> #(b) For the number Fn=k in (a), what is the index n?
> print(n)
[1] 12
```

```
> # (c)Use for( ) to write a function print.Fib
> #that takes an integer k as its input and prints ALL Fibonacci numbers <= k.
> #Test k=100 and show me the results.
>
> print.Fib=function(k){
+   n=1
+   while(fib(n)<=k){
+     n=n+1
+   }
+
+   for(i in 1:n){
+     if(fib(i)<k){
+       print(fib(i))
+     }
+   }
+ }
> print.Fib(100)
[1] 1
[1] 1
[1] 2
[1] 3
[1] 5
[1] 8
[1] 13
[1] 21
[1] 34
[1] 55
[1] 89
```

Q2. (10%) Write a function *second.smallest*( ) that takes a vector x as its input. The function will return the number that is the second smallest (第二小) inside x.

Show me the results of *second.small*(x=c(2, 8, 8, 2, 5, 2, 5, 2)).

```
> #Q2. Write a function sencond.smallest() that takes a vector x as its input.
> #The function will return the number that is the second smallest inside x.
> #Show the results of second.smallest(x=c(2, 8, 8, 2, 5, 2, 5, 2)).
>
> x = c(2, 8, 8, 2, 5, 2, 5, 2)
>
> second.small = function (input) {
+   output <- unique(sort(input, decreasing = FALSE))
+   return(output[2])
+ }
>
> second.small(x)
[1] 5
```

Q3. (10%) Use *while*( ) and/or *if... else* to write a function *f.exist* that takes an *integer* z and a *vector* x as its inputs. The function *f.exist* will return TRUE only if z is inside x.

Test f.exist(z=10, x=c(1:10)) and f.exist(z=10, x=c(9, 3, 1)). Show the answers.

```
> #Q3. (10%) Use while( ) and/or if...else to write a function f.exist
> #that takes an integer z and a vector x as its inputs.
> #The function f.exist will return TRUE only if z is inside x.
> #Test f.exist(z=10, x=c(1:10)) and f.exist(z=10, x=c(9, 3, 1)). Show the answers.
>
> f.exist=function(z,x){
+   if(any(z==x)){
+     return(TRUE)
+   }
+   else{
+     return(FALSE)
+   }
+ }
> print(f.exist(z=10, x=c(1:10)))
[1] TRUE
> print(f.exist(z=10, x=c(9, 3, 1)))
[1] FALSE
```

Q4. (10%) Use *while*( ) and/or *if... else* to write a function *f.divide* that takes an *integer* z as its input. The function *f.divide* will return how many divisors (除數) z has (other than 1 & z itself). Test *f.divide*(100) and show me the results.

```
> #Q4. Use while() and/or if...else to write a function f.divide that takes an
> #integer z as its input. The function f.divide will return how many divisors
> #z has(other than 1 & z itself).
> #Test f.divide(100) and show the results.
>
> f.divide = function (n) {
+   index = 2
+   result = c()
+
+   while (index < n) {
+     if(n %% index == 0) {
+       result = append(result, index)
+     }
+     index = index + 1
+   }
+   #print(result)
+   return(length(result))
+ }
>
> f.divide(100)
[1] 7
```

Q5. (10%) Write a function *UNIQUE*( ) that takes a vector x as its input. The function will return a new vector with all unique (獨特的) numbers in x with duplicated (重複) elements removed. Do NOT use *unique*( ) in *R*. Show me the results of UNIQUE(x=c(2, 8, 8, 2, 5, 2, 5, 2)).

```
> #Q5. (10%) Write a function UNIQUE( ) that takes a vector x as its input.
> #The function will return a new vector with all unique numbers in x with duplicated elements removed.
> #Do NOT use unique( ) in R. Show me the results of UNIQUE(x=c(2, 8, 8, 2, 5, 2, 5, 2)).
>
> UNIQUE = function(x){
+
+   for(i in 1:length(x)){
+     j = i + 1
+     while(j <= length(x)){
+       if(x[i]==x[j]){
+         x=x[-j]
+       }
+       else{
+         j = j + 1
+       }
+     }
+   }
+   #x = x[ x != y ]
+   print(x)
+ }
> UNIQUE(x=c(2, 8, 8, 2, 5, 2, 5, 2))
[1] 2 8 5
```

Q6. (20%) The Babylonian method (巴比倫法) is famous for getting the square root (平方根) of any number. Suppose we have a positive number S, the Babylonian method suggests that

$$x_{n+1} = 0.5(x_n + S / x_n)$$

x_n is your current guess of $\sqrt{S}$ and x_{n+1} is your next guess of $\sqrt{S}$. You will STOP searching only if $|x_{n+1} - x_n| <$ tolerance.

Now, set S=125348, your initial guess x_0=600, and tolerance=1e-5.

Use *while*( ) to implement the algorithm and show me the square root ($\sqrt{S}$) you find. The answer should be 354.0452.

How about S=9527, initial guess x_0=87, and tolerance=1e-5? What's the answer?

How about S=5566, initial guess x_0=78, and tolerance=1e-5? What's the answer?

```
> #Q6. The Babylonian method is famous for getting the square root of any number.
> #Now, set S = 125348, your initial guess X_0 = 600, and tolerance = 1e-5/
> #Use while() to implement the algorithm and show the square root you find.
> #The answer should be 354.0452.
> #How about S = 9526, X_0 = 87, and tolerance = 1e-5? what's the answer?
> #How about S = 5566, X_0 = 78, and tolerance = 1e-5? what's the answer?
> babylonian=function(S,x0,tolerance){
+   x1=0.5*(x0 + S/x0)
+   while(abs(x1-x0)>tolerance){
+     x0=x1
+     x1=0.5*(x0 + S/x0)
+   }
+   return(x1)
+ }
> babylonian(125348,600,1e-5)
[1] 354.0452
> babylonian(9527,87,1e-5)
[1] 97.60635
> babylonian(5566,78,1e-5)
[1] 74.60563
```

Q7. (25%) Write a function *BesselI_Gen* that has five arguments (a, v, z, max, tolerance). The function will compute the **generalized** modified Bessel function of the first kind:

$$I_a^v(z)=\sum_{m=0}^{\infty}\frac{1}{\left[\Gamma(m+a+1)m!\right]^v}\left(\frac{z}{2}\right)^{2m+a} \quad \text{where } \Gamma(\bullet) \text{ is the } gamma(\ ) \text{ in } R.$$

As infinite sum is NOT possible in computers, please mimic the example in lecture 2 to compute the value of the function. Also, the *BesselI_Gen* function will continue to add up numbers only if

$$m<\max \ \& \ \left|I_a^v(z)_{m-1}-I_a^v(z)_m\right|>\text{tolerance}$$

Once you finish coding, run *BesselI_Gen*(5, 1, 10, 1000, 1e-5) in *R*. What is the value? Then run besselI(10, 5) (a built-in function for the modified Bessel function of the first kind) in *R*. Are the two values identical?

```
> #Q7. (25%) Write a function BesselI_Gen that has five arguments (a, v, z, max, tolerance).
>
> BesselI_Gen=function(a,v,z,max,tolerance){
+   m=0
+   val.last=-Inf
+   val=(gamma(m+a+1)*factorial(m))^(-v)*(z/2)^(2*m+a)
+   while(m<max && abs(val-val.last)>tolerance){
+     val.last=val
+     m=m+1
+     val=val.last+(gamma(m+a+1)*factorial(m))^(-v)*(z/2)^(2*m+a)
+   }
+   return(val)
+ }
>
> BesselI_Gen(5, 1, 10,1000, 1e-5)
[1] 777.1883
> besselI(10, 5)
[1] 777.1883
```