UNIVERSITI TEKNIKAL MALAYSIA MELAKA	FACULTY OF ENGINEERING TECHNOLOGY, UNIVERSITI TEKNIKAL MALAYSIA MELAKA		
DISCRETE MATHEMATICS			
BEEC 3413	SEMESTER 1		SESI 2021/2022
LAB 4: RECURSIVE ALGORITHM			
NAME AND MATRIX NUMBER	1.**RAHMAN KAZI ASHIKUR**	**B081910450**	
COURSE	BEEC		
DATE	DEC,5, 2021		
NAME OF INSTRUCTOR	Azman		
EXAMINER'S COMMENT			VERIFICATION STAMP
			TOTAL MARKS

1 OBJECTIVES

1. To understand **recursive algorithms**.
2. To implement recursive in programming problems.

2 EQUIPMENTS

1. Personal Computer.
2. R Software.

3 SYNOPSIS & THEORY

3.1 RECURSIVE ALGORITHM

Deftnition 1. An algorithm is called *recursive* if it solves a problem by reducing it to an instance of the same problem with smaller input.

Recursion although it looks almost the same with looping, but it have different approach. recursive has a part which call itself with "smaller (or simpler)" input values, and which obtain the result for the current input.

In General, recursive computing require more memory and computational compare with iterative algorithm, but they are simpler and for many cases a natural way of thinking about the problem.

Example 1. Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

Solution:

```
procedure factorial(n: nonnegative integers)
if n := 0 then return 1
else return n x factorial(n-1)
{output is n!}
```

Example 2. Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

Solution:

```
procedure  power(a: nonzero real number, n: nonnegative integers)
if n := 0 then return 1
else return a x power(a,n-1)
{output is a^n}
```

3.2 Searching using Recursive Algorithm

3.2.1 Linear Search

Example 3. Express the linear search algorithm as a recursive procedure.

Solution:

```
**procedure** search(i,j,x: i,j,x integers, 1 <= i <= j <= n, A: sets of numbers)
if a[i] = x then return i
else if i = j then return 0
else return search(i+1,j,x,A)
{output is the loxation of x in a1, a2, ..., an if it appears; otherwise it is 0}
```

3.2.2 Binary Search

Example 4. Construct a recursive version of a binary search algorithm.

Solution:

```
procedure binary search(i, j, x: i,j,x integers, 1 <= i <= j <= n, A: sets of numbers)
m = [(i + j)/2]
if x = a[m] then return m
else if x < a[m] and i < m then return binary search(i, m - 1, x, A)
else if x > a[m] and j > m then return binary search(m + 1, j, x, A)
else return 0
{output is the loxation of x in a1, a2, ..., an if it appears; otherwise it is 0}
```

4 Procedure

4.1 Recursive Algorithm

1. Open R Gui, and create a new script.
2. Type the following program code, of **factorial** function.

```
factorial<-function(n){
if (n==0){return(1)}
else{return(n*factorial(n-1))}
}
```

3. Run the program and observe the output for $n = 1, n = 2, n = 3$, and $n = 100$.
4. Type the following program code, of a^n function.

```
power<-function(a,n){
if (n==0){return(1)}
else{return(a*power(a,n-1))}
}
```

5.Run the program and observe the output for $a = 2$ and $n = 1, n = 2, n = 3$, and $n = 100$.

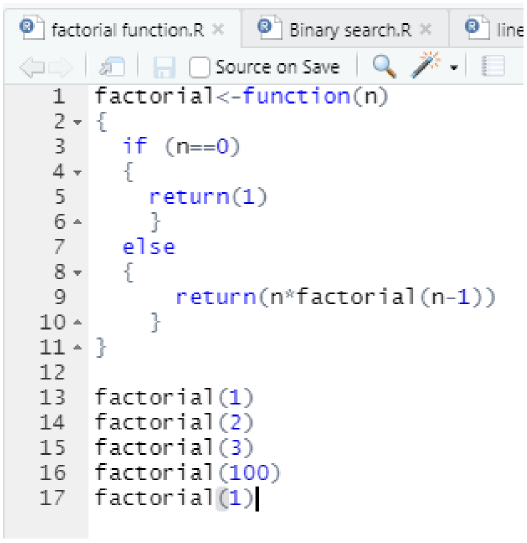
6.Create new functions for **Linear Search** algorithm using recursive.

7.Create new functions for **Binary Search** algorithm using recursive.

5 RESULT

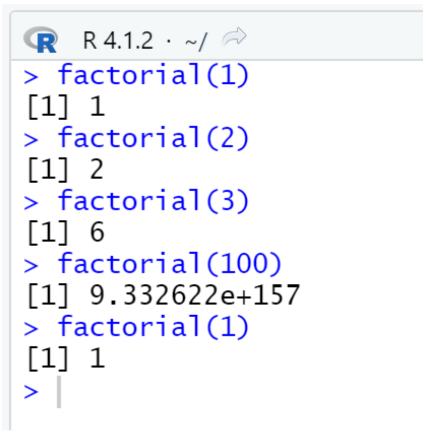
1. Follow all the procedures

factorial function



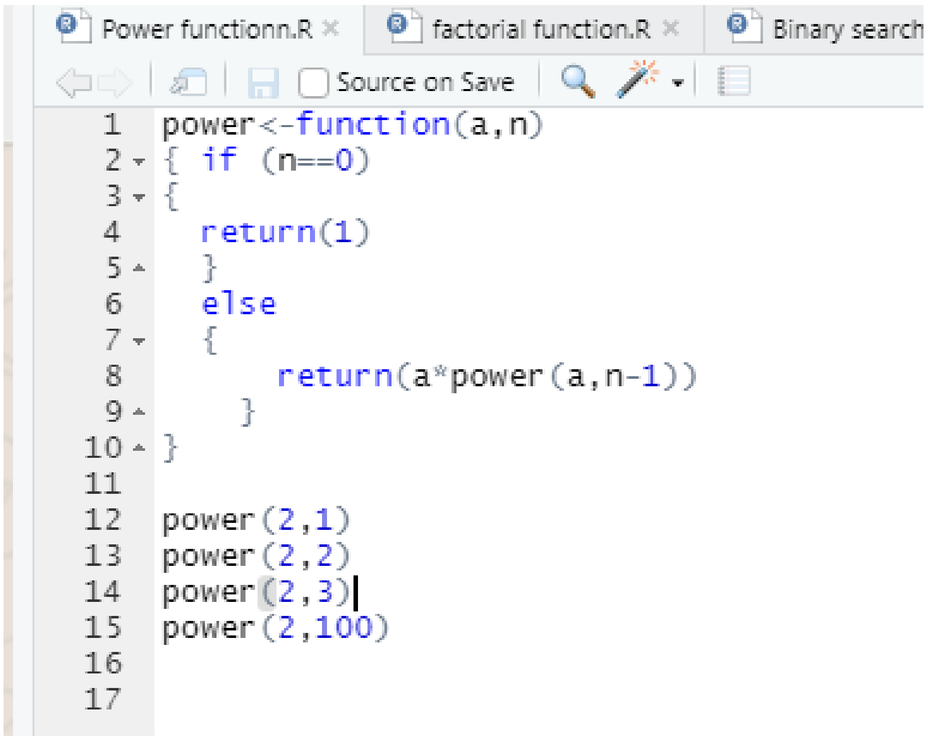
```
factorial<-function(n)
{
  if (n==0)
  {
    return(1)
  }
  else
  {
     return(n*factorial(n-1))
  }
}

factorial(1)
factorial(2)
factorial(3)
factorial(100)
factorial(1)
```

Output



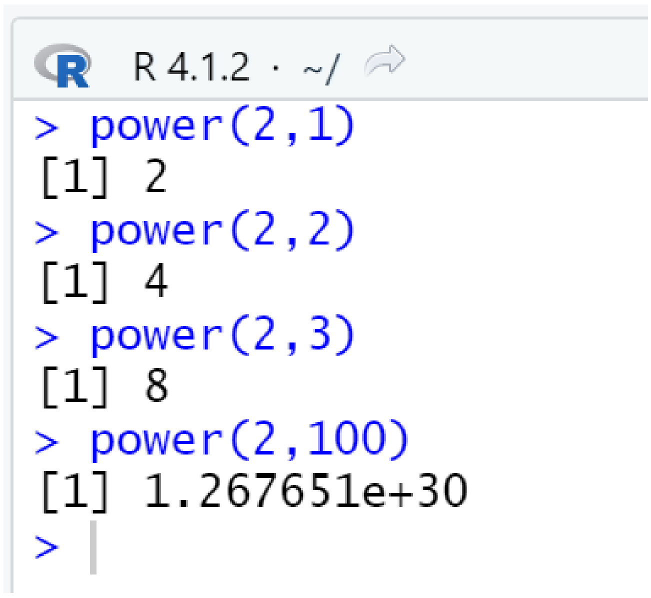
```
> factorial(1)
[1] 1
> factorial(2)
[1] 2
> factorial(3)
[1] 6
> factorial(100)
[1] 9.332622e+157
> factorial(1)
[1] 1
> 
```

Power function



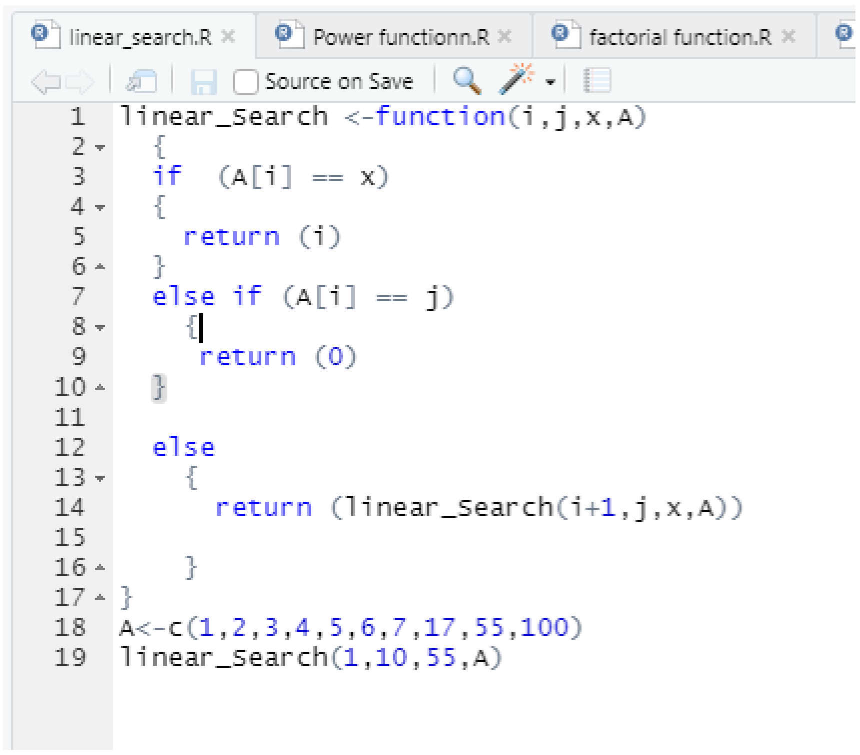
```
power<-function(a,n)
{ if (n==0)
{
  return(1)
  }
  else
  {
      return(a*power(a,n-1))
    }
}

power(2,1)
power(2,2)
power(2,3)
power(2,100)
```

Power function Result



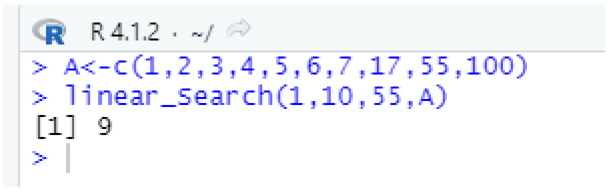
```
> power(2,1)
[1] 2
> power(2,2)
[1] 4
> power(2,3)
[1] 8
> power(2,100)
[1] 1.267651e+30
>
```

Linear Search algorithm

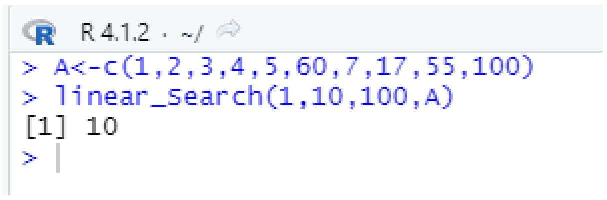


```r
linear_search <-function(i,j,x,A)
  {
  if   (A[i] == x)
  {
    return (i)
  }
  else if (A[i] == j)
    {
     return (0)
  }

  else
    {
      return (linear_search(i+1,j,x,A))

    }
}
A<-c(1,2,3,4,5,6,7,17,55,100)
linear_search(1,10,55,A)
```

Linear Search algorithm output

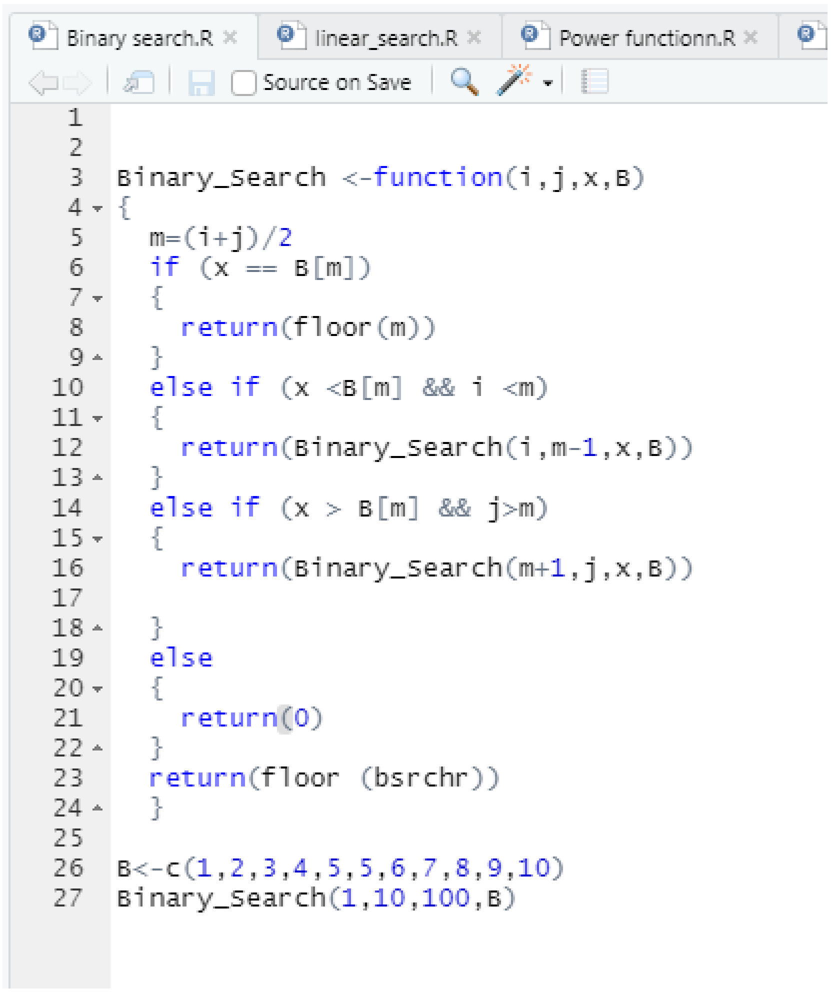


```
> A<-c(1,2,3,4,5,6,7,17,55,100)
> linear_search(1,10,55,A)
[1] 9
> 
```



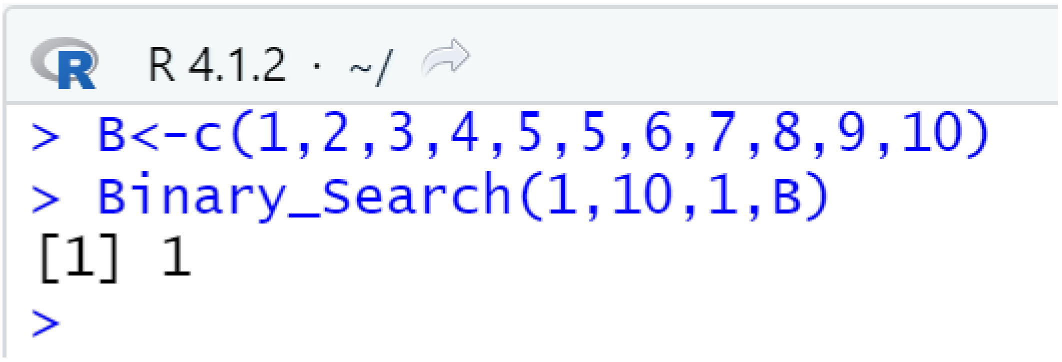
```
> A<-c(1,2,3,4,5,60,7,17,55,100)
> linear_search(1,10,100,A)
[1] 10
> 
```

Binary search code

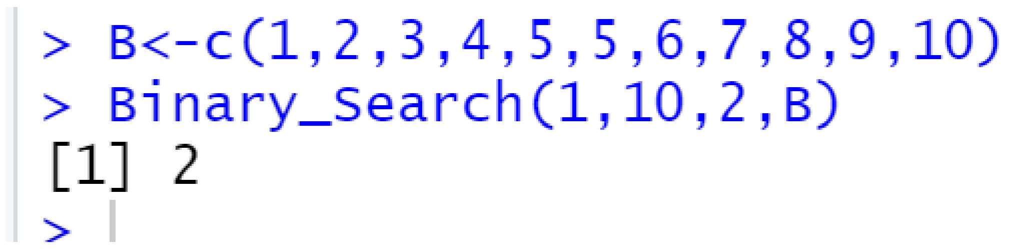


```


Binary_Search <-function(i,j,x,B)
{
  m=(i+j)/2
  if (x == B[m])
  {
    return(floor(m))
  }
  else if (x <B[m] && i <m)
  {
    return(Binary_Search(i,m-1,x,B))
  }
  else if (x > B[m] && j>m)
  {
    return(Binary_Search(m+1,j,x,B))

  }
  else
  {
    return(0)
  }
  return(floor (bsrchr))
  }

B<-c(1,2,3,4,5,5,6,7,8,9,10)
Binary_Search(1,10,100,B)
```

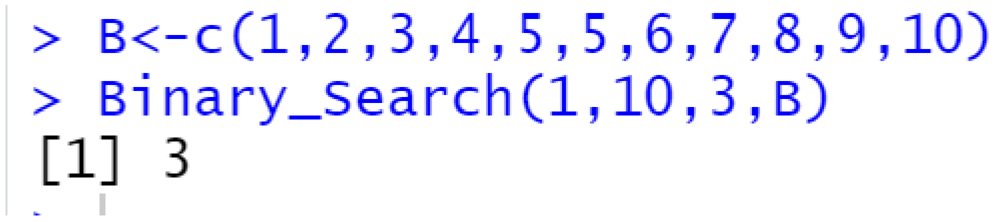
Binary search code output



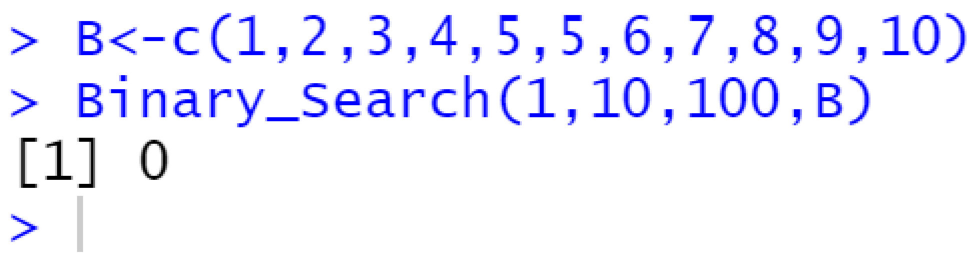
```
> B<-c(1,2,3,4,5,5,6,7,8,9,10)
> Binary_Search(1,10,1,B)
[1] 1
>
```



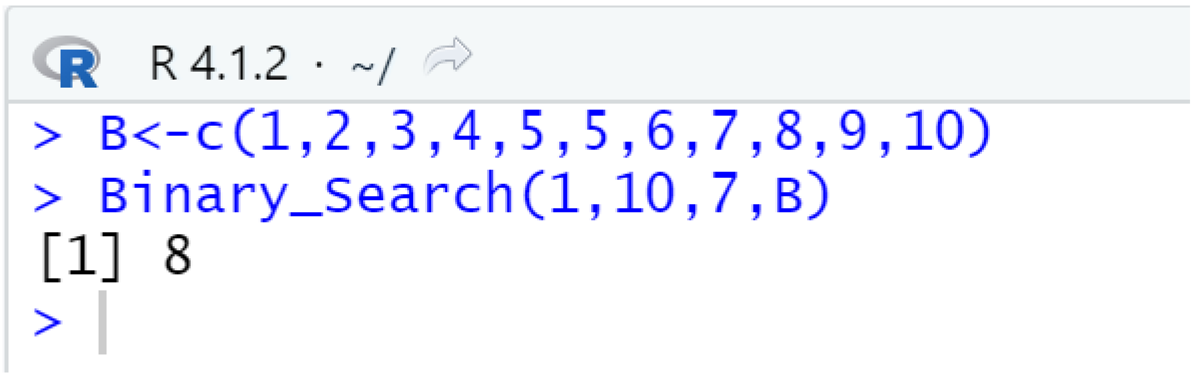
```
> B<-c(1,2,3,4,5,5,6,7,8,9,10)
> Binary_Search(1,10,2,B)
[1] 2
> 
```



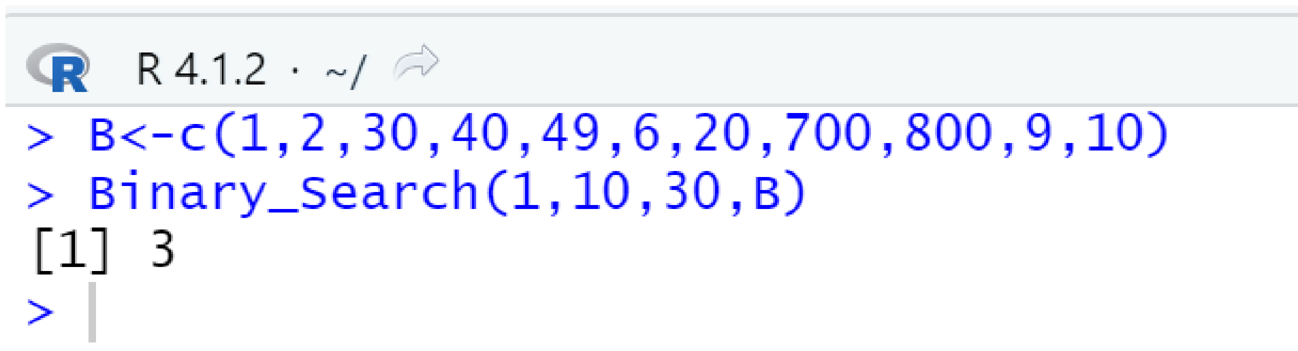
```
> B<-c(1,2,3,4,5,5,6,7,8,9,10)
> Binary_Search(1,10,3,B)
[1] 3
```



```
> B<-c(1,2,3,4,5,5,6,7,8,9,10)
> Binary_Search(1,10,100,B)
[1] 0
> 
```



```
> B<-c(1,2,3,4,5,5,6,7,8,9,10)
> Binary_Search(1,10,7,B)
[1] 8
> 
```



```
> B<-c(1,2,30,40,49,6,20,700,800,9,10)
> Binary_Search(1,10,30,B)
[1] 3
> 
```

6 DISCUSSION

What is recursive actually?

A function that calls itself is called a recursive function and this technique is known as recursion.

This special programming technique can be used to solve problems by breaking them into smaller and simpler sub-problems.

An example can help clarify this concept.

Let us take the example of finding the factorial of a number. Factorial of a positive integer number is defined as the product of all the integers from 1 to that number. For example, the factorial of 5 (denoted as 5!) will be

5! = 1*2*3*4*5 = 120

This problem of finding factorial of 5 can be broken down into a sub-problem of multiplying the factorial of 4 with 5.

5! = 5*4!

Or more generally,

n! = n*(n-1)!

Now we can continue this until we reach 0! which is 1.

The implementation of this is provided below.

what I understand from this laboratory?

From this lab I understand about recursive function. Recursive function is a function that use itself in the function creating a loop until the loop is broken or the condition is meet. This can be taken advantages on to create a function that be efficient and doesn’t need more function to be functional. Recursive makes value every loop into its own function smaller and smaller creating less need of computational power if handling a lot of number or information. This will be benefits the computer by making it need less computational power to handle the same number of information that greater computer that doesn’t use the recursive function

7 CONCLUSION

From this lab I understand recursive algorithms to implement recursive in programming problems. I learn that by implementing recursive, our codes going to be more efficient. I understand about recursive function. Recursive function is a function that use itself in the function creating a loop until the loop is broken or the condition is meet

8 REFERENCE

•The R Manuals, http://www.r-project.org/