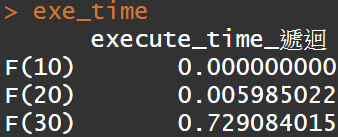
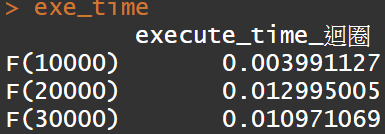
請用之前撰寫的遞迴費氏數列及迴圈費式數列程式碼，分別跑F(10), F(20), F(30)。請用程式紀錄跑的時間。





Code:

## 建立迴圈函數

Fibonacci\_loop <- function(input){

if(input <= 2){

return(1)

}else{

sum1 <- 1

sum2 <- 1

for(ii in 3:input){

out <- sum1 + sum2

if(ii %% 2 == 0){

sum1 <- out

}else{

sum2 <- out

}

}

return(out)

}

}

## 建立遞迴函數

Fibonacci <- function(input){

if(input <= 2){

return(1)

}else{

return(Fibonacci(input - 1) + Fibonacci(input - 2))

}

}

## 紀錄遞迴時間

exe\_time <- c()

for(ii in c(10,20,30)){

Start\_time <- Sys.time()

Fibonacci(ii)

exe\_time[ii/10] <- as.numeric(Sys.time() - Start\_time)

}

exe\_time <- matrix(exe\_time,3,1)

colnames(exe\_time) <- "execute\_time\_遞迴"

row.names(exe\_time) <- c("F(10)","F(20)","F(30)")

exe\_time

## 記錄迴圈時間

exe\_time <- c()

for(ii in c(10000,20000,30000)){

Start\_time <- Sys.time()

Fibonacci\_loop(ii)

(Fibonacci\_loop(ii))

exe\_time[ii/10000] <- as.numeric(Sys.time() - Start\_time)

}

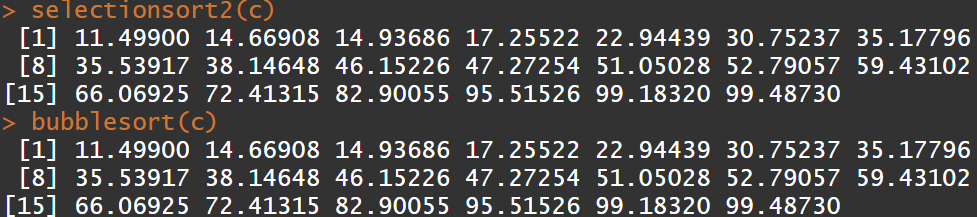
exe\_time<-matrix(exe\_time,3,1)

colnames(exe\_time)<-"execute\_time\_迴圈"

row.names(exe\_time)<-c("F(10000)","F(20000)","F(30000)")

exe\_time

請撰寫選擇排序法及氣泡排序法兩個函數，輸出數列為遞增數列。



Code:

#selection sort

s<-c()

selectionsort2 <- function(input)

{

for(i in 1:length(input))

{

aa<-which.min(input)

min<-input[aa]

input <- input[-aa]

s<-c(s,min)

}

return(s)

}

#bubble sort.

bubblesort <- function(input)

{

n<-length(input) #設定出序列的長度

for(i in 1:(n-1)) #設定跑幾次1~n-1的迴圈(最後一個不會再有右邊的可以交換)

{

for(k in 1:(n-i)) #設定每跑1~n-1次迴圈時內部要跑的次數

{

if(input[k]>input[k+1]) #設定交換條件(如果左大於右則交換)

{

bigger<-input[k]#將要交換的較大一方(左邊)的值設定到bigger這個變數

smaller<-input[k+1]#將要交換的較小一方(右邊)的值設定到smaller這個變數

input[k+1]<-bigger #將原先較大的值輸入到右邊

input[k]<-smaller #將原先較小的值輸入到左邊

#因此已完成左>右的話，的左右交換。

}

}

}

return(input)

}

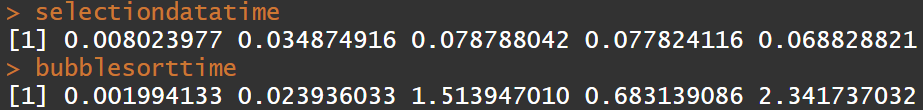
#創建一個隨機數列測試

c <- runif(20,10,100)

selectionsort2(c)

bubblesort(c)

請讀入partA\_timecp.csv檔案，並針對V1, V2, V3,V4,V5使用這兩個函數進行排序，並用程式紀錄跑的時間，比較一下兩個方法在input不同時候有甚麼差異。



兩個函數的消耗時間都呈現同一個現象：

(1) 樣本越多時，所花時間越多→[2]>[1]

(2) 樣本相同時，越漂亮的數列(呈現由小到大)所花時間最少，越不漂亮的數列(呈現由大到小)，所花時間最多。

→Best case < Average case < Worse case

→[4] < [3] < [5]

Code:

test<-read.csv(file.choose())

#selection sort

selectiondatatime<-c(rep(0,5))

for (i in 1:5)

{

Start\_time<-Sys.time()

selectionsort2(test[,i])

selectiondatatime[i]<-as.numeric(Sys.time()-Start\_time)

}

#bubble sort

bubblesorttime<-c(rep(0,5))

for (i in 3:5)

{

Start\_time<-Sys.time()

bubblesort(test[,i])

bubblesorttime[i]<-as.numeric(Sys.time()-Start\_time)

}

#因為bubble sort會詳細比較兩個位置的大小，不像sort是找每次的最小值，因此有NA時無法跑。

kk<-test[1:50,1]

kkk<-test[1:500,2]

Start\_time<-Sys.time()

bubblesort(kk)

a1<-as.numeric(Sys.time()-Start\_time)

bubblesorttime[1]<-a1

Start\_time<-Sys.time()

bubblesort(kkk)

a2<-as.numeric(Sys.time()-Start\_time)

bubblesorttime[2]<-a2

selectiondatatime

bubblesorttime