~~Activity 19S~~

```
~~FUNCTION fibonacci (number : INTEGER) RETURNS INTEGER~~
    ~~IF number = 0 OR number = 1~~
      ~~THEN~~
        ~~answer ← 1 // base case~~
      ~~ELSE~~
        ~~answer ← Fibonacci(number - 1) + fibonacci (number - 2)~~
        ~~// recursive call with general case~~
    ~~ENDIF~~
    ~~RETURN answer~~
~~ENDFUNCTION~~
```

~~Call number~~	~~Function call~~	~~number~~	~~answer~~	~~RETURN~~
~~1~~	~~fibonacci(4)~~	~~4~~	~~fibonacci(3) +~~ ~~fibonacci(2)~~	
~~2~~	~~fibonacci(3)~~	~~3~~	~~fibonacci(2) +~~ ~~fibonacci(1)~~	
~~3~~	~~fibonacci(2)~~	~~2~~	~~fibonacci(1) +~~ ~~fibonacci(0)~~	
~~4~~	~~fibonacci(1)~~	~~1~~	~~1~~	~~1~~
~~5~~	~~fibonacci(0)~~	~~0~~	~~1~~	~~1~~
~~3 continued~~	~~fibonacci(2)~~	~~2~~	~~1 + 1~~	~~2~~
~~2 continued~~	~~fibonacci(3)~~	~~3~~	~~1 + 2~~	~~3~~
~~1 continued~~	~~fibonacci(4)~~	~~4~~	~~3 + 2~~	~~5~~

End of chapter questions

1 a) i)

```
FOR ThisPointer ← 2 TO 10
    // use a temporary variable to store item which is
    to // be inserted into its correct location
    Temp ← NameList[ThisPointer]
    Pointer ← ThisPointer – 1
    WHILE (NameList[Pointer] > Temp) AND Pointer > 0
        // move list item to next location
        NameList[Pointer + 1] ← NameList[Pointer]
        Pointer ← Pointer - 1
    ENDWHILE
    // insert value of Temp in correct location
    NameList[Pointer] ← Temp
ENDFOR
```

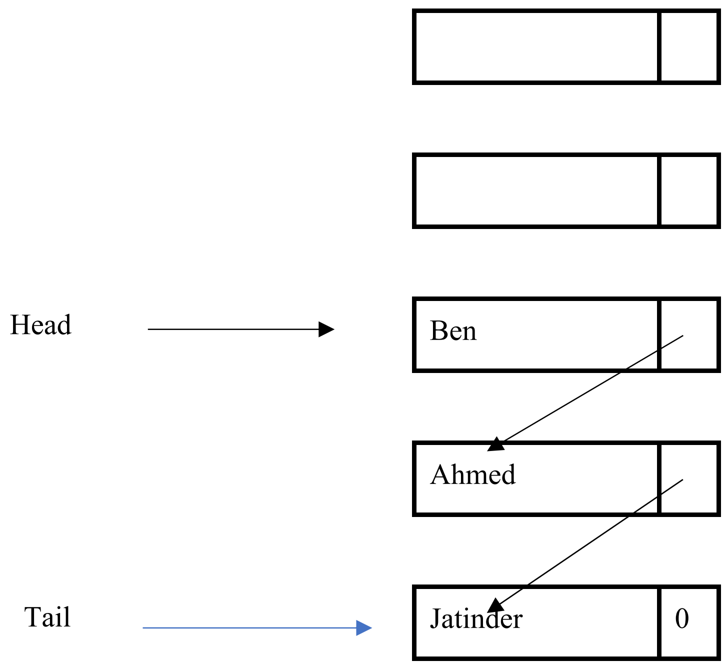
b) **ii)** The outer loop is always executed 9 times and the inner loop is never executed as `Temp` is always larger.

c) **i)** Both loops are always executed 9 times.

ii)

```
REPEAT
    NoMoreSwaps ← TRUE
    FOR Pointer ← 1 TO NumberOfItems – 1
        IF NameList[Pointer] > NameList[Pointer + 1]
        THEN
            NoMoreSwaps ← FALSE
            Temp ← NameList[Pointer]
            NameList[Pointer] ← NameList[Pointer + 1]
            NameList[Pointer + 1] ← Temp
        ENDIF
    ENDFOR
    NumberOfItems ← NumberOfItems - 1
UNTIL NoMoreSwaps
```

2 a)



b) i)

		Name	Pointer
HeadPointer	[1]		2
0	[2]		3
	[3]		4
TailPointer	[4]		5
0	[5]		6
	[6]		7
FreePointer	[7]		8
1	[8]		9
	[9]		10
	[10]		0

ii)

```
PROCEDURE RemoveName()
    //Report error if Queue is empty
    IF HeadPointer = 0
  THEN
    Error
  ELSE
        OUTPUT Queue[HeadPointer].Name
   //current node is head of queue
   CurrentPointer ← HeadPointer
        // update head pointer
        HeadPointer ← Queue[CurrentPointer].Pointer
       //if only one element in queue, then update tail pointer
  IF HeadPointer = 0
   THEN
     TailPointer ← 0
  ENDIF
       // link released node to free list
       Queue[CurrentPointer].Pointer ← FreePointer
       FreePointer ← CurrentPointer
   ENDIF
ENDPROCEDURE
```