

Recursive Functions

1. Describe what the following function does by completing the comment lines.

```
[0] p = Q(N)
[0.1]NB. Input N is a list of real numbers
[0.2]NB. Output p is the square root of N if N is a single number
[0.2.1]NB. If N is a list and its last entry is r, then p is a list of the rth roots
[0.2.2]NB. of the preceding numbers.
[0.3]NB. For example Q(16) = 4; Q(32, 5) = 2 and Q(8, 27, 125, 3) = 2, 3, 5
[1] If length(N) = 1
    [1.1] p ← Q(N, 2)
    else
    [1.2] p ← curtail(N) ^ (1/last(N))
```

Here the *power* function, \wedge , is assumed to work term by term on every element of a list so that, for example, $(3, 4, 5) \wedge 3$ returns 27, 64, 125.

What does $Q(23.56, 38.37, 126.193, 2.25) \wedge 2.25$ return?

23.56, 38.37, 126.193

2. Write pseudo-code for a recursive function that, given a list, returns the list in reverse order.

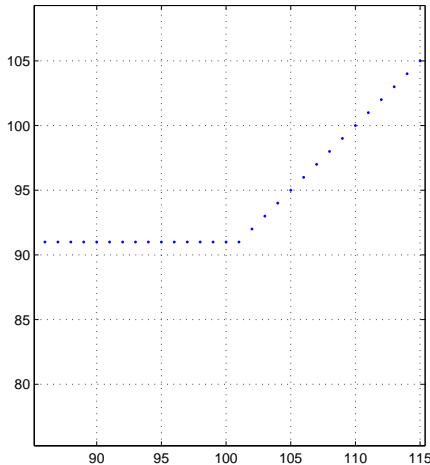
```
[0] S = reverse(L)
[0.1]NB. Input L is a list of numbers or characters
[0.2]NB. Output S is list L in reverse order
[1] if length(L) = 1
    [1.1] S ← L
    else
    [1.2] S ← last(L), reverse(curtail(L))
```

3. The following is an algorithm for McCarthy's function.

```
[0] p = M(n)
[1] If n > 100
    [1.1] p ← n - 10
    else
    [1.2] p ← M(M(n + 11))
```

Working in groups of four or five collect data on the output of this function.

Clearly the function is well-defined for $n > 100$, but do you think it is well defined for positive integers less than 100? What is its range of values for integers n in $1 \leq n \leq 100$?



The Matlab code for running McCarthy's function is shown below and a graph of the output is shown at left.

```
function p = M(n)
% n is any positive integer
% p is the value of the McCarthy function
if n>100
    p=n-10;
else
    p=M(M(n+11));
end
```

The linear relationship between input and output for $n \geq 101$ is clear from the algorithm, but the constant value (91) for $n \leq 100$ may be less expected. The point of the exercise is to trace how this comes about.

Let us trace the execution of $M(88)$, for example. Execution of this call is suspended with the function waiting on a result from

$$M(M(99)) \dots \dots \dots (1)$$

which itself is suspended waiting for a return value from

$$M(99) \dots \dots \dots (2)$$

Now $M(99)$ is suspended waiting on

$$M(M(110)) \dots \dots \dots (3)$$

Finally $M(110)$ returns 100.

Resuming the execution at (3):

$$M(M(110)) = M(100).$$

Now $M(100)$ is $M(M(111))$ and since $M(111)$ is 101, we have

$$M(100) = M(M(111)) = M(101) = 91.$$

Thus at (3) the number 91 is returned for $M(99)$ in (2). Resuming (1), we now need the value of $M(91)$.

But $M(91)$ is $M(M(102)) = M(92)$ and

$$M(92) \text{ is } M(M(103)) = M(93)$$

and in this manner we find that

$$M(91) = M(92) = \dots = M(101) = 91$$

Starting with any number below 90, say 50, we will get a sequence like $M(M(61)) = M(M(M(72))) = M(M(M(M(83)))) = M(M(M(M(M(94))))))$ which can begin executing with $M(94)$ or some other number in the nineties all of which have the value 91. Since $M(91) = 91$ the stack of M functions waiting to be evaluated will all return 91, which will be the final output.

4. Consider the following function for the n^{th} term of a sequence.

```
[0]  $an = T(n)$ 
[1] If  $n < 2$ 
    [1.1]  $an \leftarrow 1$ 
else
    [1.2]  $an \leftarrow T(n - 1) + T(n - 2)$ 
```

Use this algorithm to compute the first 6 terms of the sequence.

Both $T(0)$ and $T(1)$ will return 1, so $T(2)$ will return 2. $T(3)$ will return $T(2) + T(1)$ or 3. $T(4)$ will return 5 and $T(5)$ will return 8. The first six terms are 1,1,2,3,5,8.

Write two algorithms that return the first n terms of the sequence, one that is recursive and another that computes the values iteratively.

```

[0]  $L = FibonacciR(n)$ 
[0.1]NB. Input  $n$  is the number of terms in the Fibonacci sequence
[0.2]NB. Output  $L$  is the list of numbers in the Fibonacci sequence
[1] if  $n = 1$ 
[1.1] $L \leftarrow 1$ 
elseif  $n = 2$ 
[1.2] $L \leftarrow 1, 1$ 
else
[1.3] $L \leftarrow FibonacciR(n - 1)$ 
[1.3] $L \leftarrow L, last(L) + last(curtail(L))$ 

[0]  $L = FibonacciI(n)$ 
[0.1]NB. Input  $n$  is the number of terms in the Fibonacci sequence
[0.2]NB. Output  $L$  is the list of numbers in the Fibonacci sequence
[1] $L \leftarrow 1, 1$ 
[2] if  $n = 1$ 
[2.1] $L \leftarrow 1$ 
else
[2.2] for  $k = 1$  to  $n - 2$ 
[2.2.1] $L \leftarrow L, last(L) + last(curtail(L))$ 

```

Coding these functions into Matlab for example will confirm that they work.

```

function y=fibi(n)
    y=[1,1];
    if n==1
        y=1;
    else
        for k=1:n-2
            h=length(y);
            y=[y,y(h)+y(h-1)];
        end
    end
end

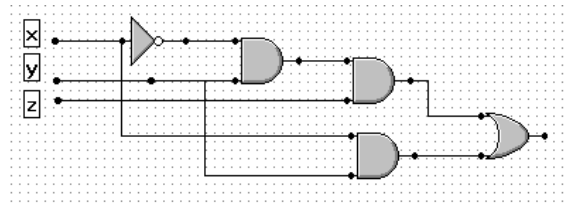
function y=fibr(n) if n==1
    y=1;
elseif n== 2
    y=[1,1];
else
    y=fibr(n-1);
    h=length(y);
    y=[y,y(h)+y(h-1)];
end
end

```

Karnaugh Maps

- Write the symbolic logic expression $(\neg x \wedge y \wedge z) \vee (x \wedge y)$ in Boolean notation and draw a digital circuit for it. Write the disjunctive normal form for the expression; find a suitable reduction using a Karnaugh map and draw the corresponding digital circuit for the result.

$$(\neg x \wedge y \wedge z) \vee (x \wedge y) \equiv x'yz + xy$$



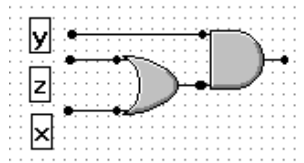
The disjunctive normal form for $x'yz + xy$ is

$$x'yz + xy(z + z') = x'yz + xyz + xyz'$$

and its Karnaugh map is shown below.

	yz	$y'z$	$y'z'$	yz'
x	1	0	0	1
x'	1	0	0	0

The reduced form of the expression is $yz + xy = y(x + z) \equiv y \wedge (x \vee z)$



- Find the minterms for the following function and draw a digital circuit with the least number of components for it

a	b	c	$F(a, b, c)$
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	1

The minterms are abc , $ab'c$, $ab'c'$ and $a'b'c'$.

The Karnaugh map for the expression is

F	bc	$b'c$	$b'c'$	bc'
a	1	1	1	0
a'	0	0	1	0

The reduced form is $ac + b'c' = ac + (b + c)'$.

