1. Write an invert procedure, that outputs the inverse of a permutation; for example, [invert {3 2 5 1 4}] should return {4 2 1 5 3}.

```
proc invert {L} {
    set inv $L
    for {set i 0} {$i<[llength $L]} {incr i} {
        set pos [expr [lindex $L $i]-1]
        set inv [lreplace $inv $pos $pos [expr $i+1]]
    }
    puts $inv
    return $inv
}

invert {3 2 5 1 4}
```

2. Write a power procedure, that outputs the power of a permutation: for example, [power {2 3 4 5 1} 2] should return {3 4 5 1 2}.

```
proc compose {g h} {
    foreach a $h { lappend out [lindex $g $a-1] }
    set out
}

proc power {L pow} {
    set Lpow $L
    for {set i 1} {$i < $pow} {incr i} {
        set Lpow [compose $Lpow $L]
    }
    puts $Lpow
    return $Lpow
}

set x {2 3 4 5 1}
power $x 2
```

3. Write a cycles procedure, that writes a permutation as a product of disjoint cycles: for example, [cycles {3 4 1 5 2}] should return (1 3)(2 4 5), and [cycles {3 2 5 1 4}] should return (1 3 5 4)(2) -- though it's fine if you leave out the trivial cycle: [cycles {3 2 5 1 4}] could return (1 3 5 4).

```
proc cycles {L} {
    dict set visited temp temp #create an arrangement for mapping values to keys. The dictname is visited
    set arr {}
    for {set i 0} {$i < [llength $L]} {incr i} {
        if {[dict exist $visited [expr $i+1] ] == 1} { #dict exist tests whether a visisted exists and is defined
            continue
        } else {
            lappend arr ([expr $i+1]
            dict set visited [expr $i+1] true
            set j $i
            while { [dict exist $visited [lindex $L $j]] == 0 } {
                dict set visited [lindex $L $j] true
                lappend arr [lindex $L $j]
                set j [expr [lindex $L $j]-1]
            }
            lappend arr )
        }
    }
    puts $arr
    return $arr

}

cycles {3 4 1 5 2}
```