Recursion Exercise 1

Write a function to recursively compute $n!$

In [1]:
```
function factorial( n )

    #Base case
    if n <= 2
        return n
    end

    #Recursive case
    return n*factorial(n-1)

end
```

Out[1]:
```
factorial (generic function with 1 method)
```

In [2]:
```
factorial(5)
```

Out[2]:
```
120
```

Recursion Exercise 2

Write a recursive function that reverses an input string.

In [3]:
```
function reverseword( word )

    #Base case
    if length(word) == 0 || length(word) == 0
        return word
    end

    #Recursive case
    return string( word[end], reverseword(word[1:end-1]) )

end
```

Out[3]:
```
reverseword (generic function with 1 method)
```

In [4]:
```
reverseword( "hello" )
```

Out[4]:
```
"olleh"
```

Recursion Exercise 3

A palindrome is a word that reads the same forwards and backwards. Examples include abba, redivider, civic, ... Write a recursive function that takes in a word and returns if it is a palindrome or not.

In [5]:
```
function palindrome( word )

    #Base case
    if length(word) == 0 || length(word) == 0
        return true
    end

    #Check whether 1st and last are the same
    if word[1] != word[end]
        return false
    end

    #Check recursively
    return palindrome( word[2:end-1] )

end
```

Out[5]:
```
palindrome (generic function with 1 method)
```

In [6]:
```
palindrome("civic")
```

Out[6]:
```
true
```

In [7]:
```
palindrome("plane")
```

Out[7]:
```
false
```

Recursion Exercise 4

Write a recursive function that finds the minimum element in an array. Hint: Split into two smaller arrays and find min of those. How to find the min if you know these?

In [8]:
```
function minimumarr( arr )

    #Base case
    if length(arr) == 1
        return arr[1]
    end

    #Split the array into two
    midpoint = Int( floor(length(arr)*0.5) )
    arr1 = arr[1:midpoint]
    arr2 = arr[midpoint+1:end]

    #Find min of subarrays
    min1 = minimumarr(arr1)
    min2 = minimumarr(arr2)

    #Compare min of two, must be min of whole thing
    if min1 < min2
        return min1
    else
        return min2
    end

end
```

Out[8]:
```
minimumarr (generic function with 1 method)
```

In [9]:
```
minimumarr( [-2,1,123,5,12,-211] )
```

Out[9]:
```
-211
```

Recursion Exercise 5

Mergesort is an algorithm that uses divide-and-conquer algorithm to sort lists. The idea is that you split an array into two smaller subarrays, sort those subarrays, then combine the two of them such that they are sorted. Implement this using recursion.

In [10]:
```
function mergesort( arr )

    #Base case
    if length( arr ) == 0 || length( arr ) == 1
        return arr
    end

    #Split the array into two and sort those recursively
    midpoint = Int( floor(length(arr)*0.5) )
    arr1 = mergesort( arr[1:midpoint] )
    arr2 = mergesort( arr[midpoint+1:end] )

    #Combine the sorted recursive arrays
    sorted = []
    count1 = 1; count2 = 1
    while count1 <= length(arr1) && count2 <= length(arr2)

        if arr1[count1] < arr2[count2]
            push!(sorted, arr1[count1])
            count1 += 1
        else
            push!(sorted, arr2[count2])
            count2 += 1
        end

    end

    #Add the rest of the arrays that haven't been added yet
    while count1 <= length(arr1)
        push!(sorted, arr1[count1])
        count1 += 1
    end

    while count2 <= length(arr2)
        push!(sorted, arr2[count2])
        count2 += 1
    end

    return sorted

end
```

Out[10]:
```
mergesort (generic function with 1 method)
```

In [11]:
```
mergesort( [9,1,4,2,5,3,8] )
```

Out[11]:
```
7-element Vector{Any}:
 1
 2
 3
 4
 5
 8
 9
```

In [12]:
```
mergesort( [9,1,4,2,5,3,8,1,2,3] )
```

Out[12]:
```
10-element Vector{Any}:
 1
 1
 2
 2
 3
 3
 4
 5
 8
 9
```

Recursion Exercise 6

This problem is meant to be harder. Starting from zero, find the minimum number of steps to get to a number n, where on the ith step, you can either move backwards or forwards i steps. That is on step 1, you can either end at -1,1 and step 2, either at -3,1,-1,3, and so on.

In [13]:
```
function minsteps(source, step, dest)

    #Base case to make it stop - think about it yourself as to why we ne
    if abs(source) > dest
        return Inf
    end

    #Base case if done
    if source == dest
        return 0
    end

    #Recursive case
    return 1 + min( minsteps(source-step, step+1, dest), minsteps(source

end
```

Out[13]:
```
minsteps (generic function with 1 method)
```

In [14]:
```
minsteps( 0,1,3 )
```

Out[14]:
```
2.0
```

In [15]:
```
minsteps( 0,1,11 )
```

Out[15]:
```
5.0
```

In [ ]:
```

```