def selSort(L):
    """Assumes that L is a list of elements that can be compared
    using >. Sorts L in ascending order""
    prefixEnd = 0
    print 'List to be sorted =', L
    while prefixEnd != len(L):
        # look at each element in suffix
        for i in range(prefixEnd, len(L)):
            if L[i] < L[prefixEnd]:
                # swap position of elements
                L[prefixEnd], L[i] = L[i], L[prefixEnd]
        print 'Partially sorted list =', L
        prefixEnd += 1

import operator

def merge(left, right, compare = operator.lt):
    """Assumes left and right are sorted lists and
    compare defines an ordering on the elements.
    Returns a new sorted (by compare) list containing the
    same elements as (left + right) would contain.""

    result = []
    i, j = 0, 0
    while i < len(left) and j < len(right):
        if compare(left[i], right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    while (i < len(left)):
        result.append(left[i])
        i += 1
    while (j < len(right)):
        result.append(right[j])
        j += 1
    return result

def mergeSort(L, compare = operator.lt):
    """Assumes L is a list, compare defines an ordering
    on elements of L.
    Returns a new sorted list containing the same elements as L""

    if len(L) < 2:
        return L[:]
    else:
        middle = len(L)//2
        left = mergeSort(L[:middle], compare)
        right = mergeSort(L[middle:], compare)
        return merge(left, right, compare)
L = []
for i in range(5):
    L.append(random.randint(0,9))
print 'Sort in ascending order'
print mergeSort(L)
print 'Sort in descending order'
print mergeSort(L, operator.gt)

def sixFirst(i, j):
    if i == 6:
        return True
    elif j == 6:
        return False
    return i >= j

L.append(6)
print 'Sort in descending order by importance'
print mergeSort(L, sixFirst)