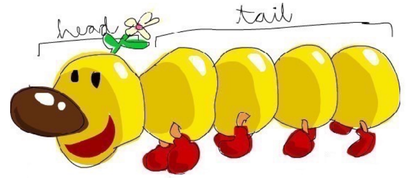
Merging Two Sorted Linked Lists

- "bad" mergesorted (recursive, very beautiful but slow)
- beautiful and fast: linked list or functional programming

```
merge :: (Ord a) => [a] -> [a] -> [a]
merge [] ys = ys
merge xs [] = xs
merge (x:xs) (y:ys)
    | x <= y    = x : merge xs (y:ys)
    | otherwise = y : merge (x:xs) ys
```

```
def mergesorted0(a, b): # v0: BAD; O(n^2) merging => O(n^2) overall
    if a == [] or b == []:
        return a+b
    elif a[0] <= b[0]:
        return [a[0]] + mergesorted0(a[1:], b) # O(n)
    else:
        return [b[0]] + mergesorted0(a, b[1:]) # O(n)
```



a[0] a[1:]

```
def mergesorted1(a, b): # v1: good; O(n) merging => O(nlogn) overall
    if a == [] or b == []:
        return a+b
    c = []
    i, j = 0, 0
    la, lb = len(a), len(b)
    while i < la or j < lb:
        if i == la or (j != lb and a[i] > b[j]):
            c.append(b[j])
            j += 1
        else:
            c.append(a[i])
            i += 1
    return c
```

```
class Node: # linked list
   def __init__(self, val, next=None):
      self.val = val
      self.next = next

first = Node(2, Node(9, Node(10, Node(15))))
```