Merge sort (cont.)

```
def merge(left, right):
    merged = []
    l = 0
    r = 0
    while l < len(left) and r < len(right):
        if left[l] <= right[r]:
            merged.append(left[l])
            l += 1
        else:
            merged.append(right[r])
            r += 1
    merged.extend(left[l:])
    merged.extend(right[r:])
    return merged
```

- Merge the two sorted arrays using two pointers
- When first array is done, fill with second until it is also done
- Take smaller item until done
- Runs in O(n) time

Merge sort analysis

- T(n) = T(n/2) + T(n/2) + O(n) = 2T(n/2) + O(n)
- Guess: will be O(n log n), so will be less than dn log n for some constant d
- Try it out:

$$T(n) \leq 2T(n/2) + O(n)$$
$$T(n) \leq 2(dn\log(n/2)/2 + O(n/2)) + O(n)$$
$$T(n) \leq dn\log(n/2) + cn + cn$$
$$T(n) \leq dn\log(n) - dn\log(2) + 2cn$$

T(n) is less than dn log n if we choose d such that d > 2c/log(2), which we can do arbitrarily, so we have shown merge sort is O(n log n).