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).

Counting sort

```
def countingSort(A):
    # assume elements are in range 1..n
    counts = [0] * n
    for i in range(len(A)):
        counts[A[i]] += 1
    output = [ ]
    for i in range(len(counts)):
        while counts[i] > 0:
            counts[i] -= 1
            output.append(i)
    return output
```

- This takes O(n) time - independent O(n) loops
- Takes advantage of constraints on data and efficiency of modifying array elements to achieve O(1) time for each element of A
- Not comparison-based: can be shown that all comparison-based sorts take O(n log n) time