CSC148 winter 2014
sorting, recursion limits
week 11

Danny Heap / Dustin Wehr
heap@cs.toronto.edu / dustin.wehr@utoronto.ca
BA4270 / SF4306D
http://www.cdf.toronto.edu/~heap/148/F13/

March 28, 2014
$O(n \lg n)$ sorts compared

memoization
You had the chance in lab to tweak `merge_sort`, `quick_sort`, and `tim-sort` (Python’s built-in sort). Running `sort.py` gives an idea of how they scale.

- why does `tim-sort` do so well?
  - $\mathcal{O}(n)$ on “nearly-sorted” lists. In general, the closer to sorted the list is, the greater the speedup compared to quick sort and merge sort.
  - programmed in C (closer to the language understood by the processor)

- what is with `count_sort` anyway?
Some programming languages implement the simplest recursions as loops, but Python doesn’t. One consequence is that our first draft of `contains` can easily exceed the recursion depth. Rewrite it with `while`
redundant function calls

The most intuitive version of fibonacci ends up making many redundant function calls:

def fib(n):
    """Return the nth fibonacci number""
    if n < 2:
        return n
    else:
        return fib(n - 1) + fib(n - 2)

e.g. fib(20) calls fib(19) and fib(18), and fib(19) also calls fib(18), so executing fib(20) results in two separate, independent computations of fib(18).
memoize!

e.g. $\text{fib}(20)$ calls $\text{fib}(19)$ and $\text{fib}(18)$, and $\text{fib}(19)$ also calls $\text{fib}(18)$, so executing $\text{fib}(20)$ results in two separate, independent computations of $\text{fib}(18)$.

Looking deeper into the recursive calls reveals that the redundancy is compounded. $\text{fib}(n)$ will execute in time exponential in $n$, but possible to do it in time $O(n)$.

Never compute the same thing twice (if you can help it)!
def fib(n:int):
    """Return the nth fibonacci number""
    computed = {}  # already-computed values of fib
    def fibmem(k:int):
        if k in computed:  # this and next op are O(1)
            return computed[k]
        elif k < 2:
            computed[k] = k
        else:
            computed[k] = fibmem(k - 1) + fibmem(k - 2)
        return computed[k]

    return fibmem(n)