Advanced Programming Topics

Asymptotic complexity analysis
Lecture 2
James Marshall

Asymptotic notation (from COM2003)

$O(g(n))$ — 'big-oh' — upper bound

$\Omega(g(n))$ — 'big-omega' — lower bound

$\Theta(g(n))$ — 'big-theta' — upper and lower bound

Upper bounds

$f(n) \in O(g(n)) \Rightarrow$

Lower bounds

$f(n) \in \Omega(g(n)) \Rightarrow$

Upper and lower bounds

$f(n) \in \Theta(g(n)) \Rightarrow$

Worst-case vs average-case complexity

Usually it is far easier to work out the maximum number of steps an algorithm will take, rather than the average number, since we then need to know something about how probable different inputs to the algorithm are. Asymptotic complexities normally refer to worst-case complexity.

Best and worst cases for some sorting algorithms

```
--insertion sort (from Thompson, The Craft of Functional
Programming)
iSort :: Ord a => [a] -> [a]
iSort []     = []
iSort (x:xs) = ins x (iSort xs)

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

--mergesort (from Thompson, The Craft of Functional Programming,
with merge function added by J. A. R. Marshall)
mergeSort :: Ord a => [a] -> [a]
mergeSort xs
  | length xs < 2 = xs
  | otherwise
     = merge (mergeSort first) (mergeSort second)
       where
       first = take half xs
       second = drop half xs
       half = (length xs) `div` 2

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

--quicksort (naive) (from Thompson, The Craft of Functional
Programming)
qSort :: Ord a => [a] -> [a]
qSort [] = []
qSort (x:xs) = qSort [y | y<-xs, y<=x] ++ [x] ++ qSort [y | y<-xs,
     y>x]
```

Average-case complexity of qSort

Despite `qSort` having a worst-case complexity of order , it is one of the most popular sorting algorithms used for large lists. Why?

Intuition: The most efficient way for `qSort` to proceed is to split the list it is sorting into two equally sized sub-lists for sorting, with the first sublist containing only elements smaller than the 'pivot', and the second sublist containing only elements that are larger.

Consider what happens in the 'average case' of a list of uniformly-distributed random numbers. At every divide step the pivot chosen by the naive version of `qSort` above is the head of the list:

Because of this in the average case `qSort` behaves like `mergeSort`, and so has average-case asymptotic complexity in (assuming that uniformly-randomly distributed lists represent the average case)

Other reasons for quicksort's popularity