Advanced Programming Topics

Algorithm analysis and design - divide-and-conquer
Lecture 3
James Marshall

Divide-and-conquer algorithms

Divide-and-conquer algorithms work by taking their input, dividing it into smaller sub-problems, conquering these independently, then combining the results into a single solution for output. Most functional programs are divide-and-conquer, since recursion works by repeatedly breaking problems down into simpler problems, until the simplest possible problem is found for which a solution is known, then combining the resulting solutions.

Analysis - insertion sort

```
--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
```

Analysis - mergesort

```
--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)
```

Analysis - quicksort

```
--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]
```