```
max' :: (Ord a) => [a] -> a
max' [] = error "empty list encountered"
max' (x:[]) = x
max' (x:xs) =
    | x > maxTail = x
    | otherwise = maxTail
    where maxTail = max' xs
```

recursive implementation for maximom function

common idiom in recursion implementations involving the use of where with guards in functions.

→ recursive implementation for replicate

↓

takes an int and some element and returns a list that has int no. of repitions of the element

```
replicate' :: (Num a, Ord a) => a -> b -> [b]
replicate' n x
    | n <= 0 = []
    | otherwise = x : replicate' (n-1) x
```

pattern matching

guards

case

where

let

```
take' :: (Num a, Ord a) => a -> [b] -> [b]
take' n [] = []
take' n (x:xs)
    | n <= 0 = []
    | otherwise = x : take' (n-1) xs
```

Recursion.

[3, 2 , 6 , 1 , 4, 5 ]

we defined the quicksort function using 'is' verb rather than passing proper instructions to perform the sorting.

replicate

elem.

take.

reverse

(i) place partition element at right position. ← quicksort

8 7 6 5 4 3 2 1

4 3 2 1 8 7 6 5

(ii) place the elements on left and right accordingly.

(iii) and call quicksort on that

```
quicksort :: (Ord a) => [a] -> [a]
quicksort [] = []
quicksort (x:xs) =
        let leftsorted = quicksort [a | a <- xs, a <= x]
            rightsorted = quicksort [b | b <- xs, b > x]
        in leftsorted ++ [x] ++ rightsorted.
```

function run

returns a list

binding the list returned from the function

finally returning

list comprehension to build two lists, one each for the left and right side of the partition element.

two empty list for left and right for single element