Tuan

Question 6.8:

```
> data Tree a = Fork (Tree a) a (Tree a) | Empty
>  deriving (Eq, Ord, Show)

> insert :: (Ord a) => a -> Tree [a] -> Tree [a]
> insert x Empty = Fork Empty [x] Empty
> insert x (Fork at ns bt)
>  | x < head ns = Fork (insert x at) ns bt
>  | x == head ns = Fork at (x : ns) bt
>  | x > head ns = Fork at ns (insert x bt)

> flatten :: (Ord a) => Tree [a] -> [a]
> flatten Empty = []
> flatten (Fork at ns bt) = flatten at ++ ns ++ flatten bt

> bsort :: (Ord a) => [a] -> [a]
> bsort = flatten . foldr insert Empty
```

Question 7.1:

```
> cp :: [[a]] -> [[a]]
> cp = foldr f [[]]
>  where f xs yss = [x : ys | x <- xs, ys <- yss]
```

Question 7.2:

```
> cols' :: [[a]] -> [[a]]
> cols' xss = foldr f (singletonList xss) (init xss)
>  where f xs xss = zipWith (:) xs xss
>        singletonList xss = [[x] | x <- last xss]
```

Question 8.1:

```
> rjustify :: Int -> String -> String
> rjustify n str | length str <= n = replicate (n - length str) ' ' ++ str

> ljustify :: Int -> String -> String
> ljustify n str | length str <= n = str ++ replicate (n - length str) ' '
```

If the string is wider than the target length my code throws an error.

Question 8.2:

```
> type Matrix a = [[a]]

> scale :: Num a => a -> Matrix a -> Matrix a
```

```
> scale n [[]] = [[]]
> scale n xss = map (map (n*)) xss

> dot :: Num a => [a] -> [a] -> a
> dot xs ys | length xs == length ys = sum (zipWith (*) xs ys)

> add :: Num a => Matrix a -> Matrix a -> Matrix a
> add xs ys | (x1, y1) == (x2, y2) = map addTuples (zip xs ys)
>  where addTuples (xs, ys) = zipWith (+) xs ys
>        (x1, y1) = (length xs, length (head xs))
>        (x2, y2) = (length ys, length (head ys))

> mul :: Num a => Matrix a -> Matrix a -> Matrix a
> mul xss yss = [ [sum (zipWith (*) xs ys) | ys <- (cols' yss)]| xs <- xss ]

> table :: Show a => Matrix a -> String
> table xss = unlines (map unwords yss)
>  where yss = cols' (padRightLists (intToString (cols' xss)))
>        intToString = map (map show)

> padRightLists :: [[String]] -> [[String]]
> padRightLists xss = [map (rjustify (longestString xs)) xs | xs <- xss]
>  where longestString xs = fst (maximum ([(length x, x) | x <- xs]))
```