Merge sort

```
sequences (a:b:xs)
  | a > b     = descending b [a]  xs
  | otherwise = ascending  b (a:) xs
sequences xs  = [xs]

descending a as bs@(b:bs')
  | a > b          = descending b (a:as) bs'
descending a as bs = (a:as): sequences bs

ascending a as bs@(b:bs')
  | a <= b        = ascending b (\ys -> as (a:ys)) bs'
ascending a as bs = as [a]: sequences bs

mergeAll [x] = x
mergeAll xs  = mergeAll (mergePairs xs)

mergePairs (a:b:xs) = merge a b: mergePairs xs
mergePairs xs       = xs

merge as@(a:as') bs@(b:bs')
  | a > b     = b:merge as  bs'
  | otherwise = a:merge as' bs
merge [] bs   = bs
merge as []   = as

mergeSort :: Ord a => [a] -> [a]
mergeSort = mergeAll . sequences
```

Последовательный поиск в нескольких списках (аналогичный подход при поиске в БД).

```
empDep         = [("Mike", "It"), ("Jan", "Sales")]
depCountry     = [("It", "Japan"), ("Sales", "USA")]
countryCurrency = [("Japan", "JPY"), ("USA", "USD")]
currencyRate   = [("JPY", 112), ("USD", 1)]

f :: String -> Maybe Int
f emp=case lookup emp empDep of
        Nothing  -> Nothing
        Just dep -> case lookup dep depCountry of
                      Nothing      -> Nothing
                      Just country -> case lookup country countryCurrency of
                                        Nothing   -> Nothing
                                        Just curr -> lookup curr currencyRate

fB emp = lookup' empDep emp >>= lookup' depCountry >>= lookup' countryCurrency
         >>= lookup' currencyRate where
           lookup' ps k = lookup k ps

fD emp = do dep <- lookup emp empDep
            country <- lookup dep depCountry
            currency <- lookup country countryCurrency
            lookup currency currencyRate
```