Aleks Itskovich (ai1221)

Julia/Haskell

Julia

Pairs With Target Sum

```
225  function sumpairs(values::Array, target::Int)
226    return sumpairsHelper(values, target, Set(), [])
227  end
228
229  function sumpairsHelper(values::Array, target::Int, passed::Set, pairs::Array)
230    if isempty(values)
231      return pairs
232    end
233
234    value = values[1]
235    diff = target - value
236
237    if in(diff, passed)
238      push!(pairs, (value, diff))
239    end
240
241    push!(passed, value)
242
243    return sumpairsHelper(values[2:end], target, passed, pairs)
244  end
```

Output

```
Any[(11, 1), (9, 3), (4, 8), (10, 2), (5, 7)]
```

All Pairs

```
254  function findAllPairs(values::Array)
255    return allPairsHelper(values, Set(), Dict())
256  end
```

```
258  function allPairsHelper(values::Array, passed::Set, pairsBySum::Dict)
259    if isempty(values)
260      return pairsBySum
261    end
262
263    currentValue = values[1]
264    for passedValue in passed
265      pair = (passedValue, currentValue)
266      sum = passedValue + currentValue
267
268      if haskey(pairsBySum, sum)
269        currentPairs = pairsBySum[sum]
270        push!(currentPairs, pair)
271      else
272        pairsBySum[sum] = [pair]
273      end
274    end
275
276    push!(passed, currentValue)
277
278    return allPairsHelper(values[2:end], passed, pairsBySum)
279  end
```

```
Dict{Any,Any}(16 => [(7, 9), (12, 4), (11, 5)],11 => [(3, 8), (7, 4), (9, 2), (1, 10)],21 => [(12, 9), (11, 10)],7 =>
 [(3, 4), (2, 5)],9 => [(1, 8), (7, 2), (4, 5)],10 => [(3, 7), (1, 9), (8, 2)],19 => [(12, 7), (8, 11), (9, 10)],17 =
> [(8, 9), (7, 10), (12, 5)],8 => [(1, 7), (3, 5)],22 => [(12, 10)],6 => [(4, 2), (1, 5)],4 => [(1, 3)],3 => [(1, 2)]
,5 => [(1, 4), (3, 2)],20 => [(8, 12), (11, 9)],23 => [(12, 11)],13 => [(1, 12), (9, 4), (11, 2), (3, 10), (8, 5)],14
 => [(3, 11), (12, 2), (4, 10), (9, 5)],15 => [(3, 12), (8, 7), (11, 4), (10, 5)],12 => [(1, 11), (3, 9), (8, 4),
 10), (7, 5)],18 => [(7, 11), (8, 10)])
```

Haskell

Pairs With Target Sum

```haskell
sumpairs :: (Num a, Ord a) => [a] -> a -> [(a, a)]
sumpairs [] _ = []
sumpairs xs x = sumpairsHelper (sort' xs) (reverse' (sort' xs)) x []

sumpairsHelper :: (Num a, Ord a) => [a] -> [a] -> a -> [(a, a)] -> [(a, a)]
sumpairsHelper [] [] z ps = ps
sumpairsHelper (x:xs) (y:ys) z ps
  | y < x = ps
  | otherwise =
      case compare (x + y) z of
        LT -> sumpairsHelper xs (y:ys) z ps
        EQ -> sumpairsHelper xs ys z (ps ++ [(x,y)])
        GT -> sumpairsHelper (x:xs) ys z ps

sort' :: Ord a => [a] -> [a]
sort' [] = []
sort' (x:xs) = (sort' smaller) ++ [x] ++ (sort' larger)
  where smaller = [a | a <- xs, a <= x]
        larger = [a | a <- xs, a > x]

reverse' :: [a] -> [a]
reverse' [] = []
reverse' (x:xs) = reverse' xs ++ [x]
```

Output:

```
[1 of 1] Compiling Main             ( sums.hs, interpreted )
Ok, one module loaded.
*Main> sumpairs [1,3,8,12,7,11,9,4,2,10,5] 12
[(1,11),(2,10),(3,9),(4,8),(5,7)]
```

All Pairs

```haskell
import Data.Map (Map)
import qualified Data.Map as Map

findAllPairs :: (Num a, Ord a) => [a] -> Map a [(a,a)]
findAllPairs [] = Map.empty
findAllPairs xs = findAllPairsHelper xs [] Map.empty

findAllPairsHelper :: (Num a, Ord a) => [a] -> [a] -> Map a [(a,a)] -> Map a [(a,a)]
findAllPairsHelper [] _ m = m
findAllPairsHelper (x:xs) ys m = findAllPairsHelper xs (x:ys) (sumAllPairs x ys m)

sumAllPairs :: (Num a, Ord a) => a -> [a] -> Map a [(a,a)] -> Map a [(a,a)]
sumAllPairs x [] m = m
sumAllPairs x (y:ys) m = do
  let add = x + y
  let pairs = Map.findWithDefault [] add m
  let newPairs = (x,y):pairs
  sumAllPairs x ys (Map.insert add newPairs m)

```