```
sublistas :: [a] -> [[a]]
sublistas [] = [[]]
sublistas (x:xs) = [x:ys | ys <- sublistas xs]  ++ sublistas xs

-- (a)
poli :: Int -> Int -> Int -> Int -> Int
poli a b c = (\x -> a * x * x + b * x + c)

-- (b)
listaPoli :: [(Int, Int, Int)] -> [Int -> Int]
listaPoli l = [ poli a b c | (a,b,c) <- l ]

--listaPoli [] = []
--listaPoli ((a,b,c):xs) = poli a b c : listaPoli xs

appListaPoli :: [Int -> Int] -> [Int] -> [Int]
--appListaPoli lFuncoes lInt = [f x | f <- lFuncoes, x <- lInt   ]
appListaPoli [] _ = []
appListaPoli (f:fs) (a:as) = f a: appListaPoli fs as


------


matriz = [[1,2,2,3],[6,7,8,9]]
mQuadrada = [[1,2,2,3],[6,7,8,9],[7,6,5,9],[5,4,8,2]]

validaMatriz :: [[Integer]] -> Bool
validaMatriz m = listaValoresIguais(map length m)

listaValoresIguais :: [Int] -> Bool
listaValoresIguais (x:[]) = True
listaValoresIguais (x:y:[]) = x == y
listaValoresIguais (x:y:ys) = x == y && listaValoresIguais (ys)

-- b

diagPrincipal :: [[Integer]] -> [Integer]
diagPrincipal l = reverse (extraiPrincipal l (length(l)-1))

extraiPrincipal :: [[Integer]] -> Int -> [Integer]
extraiPrincipal l 0 = [((l!!0)!!0)]
extraiPrincipal l i = (l!!i)!!i : extraiPrincipal l (i-1)

-- c

permutaValores :: Int -> Int -> [t] -> [t]
permutaValores x y l =
 init (take x l) ++
  [(take y l)!!(y-1)] ++
  (take (y-x-1) (drop x l)) ++
  [(take x l)!!(x-1)] ++
  (drop y l)

---
```

```
filtrarEInserir [] n = ([], 0)
filtrarEInserir l n = (p1, p2)
  where
    p1 = listasSomaImparMaiorQuePar l
    --p2 = (n *  maximoLista(somaListas (listasSomaImparMaiorQuePar
l)))
    p2 = (n *  maximoLista(somaListas p1))

maximoLista [] = minBound::Int
maximoLista [x] = x
maximoLista (x:xs) = max x (maximoLista xs)

somaListas [] = []
somaListas (x:xs) = (foldr (+) 0) x : somaListas xs

listasSomaImparMaiorQuePar :: [[Int]] -> [[Int]]
listasSomaImparMaiorQuePar [] = []
listasSomaImparMaiorQuePar (x:xs) =
   if (somaListaCond x (\e -> mod e 2 /= 0))
        > (somaListaCond x (\e -> mod e 2 == 0))
   then x:listasSomaImparMaiorQuePar xs
   else listasSomaImparMaiorQuePar xs

somaListaCond :: [Int] -> (Int -> Bool) -> Int
somaListaCond l f = foldr (+) 0 (filter f l)



-----
type Codigo = Int
data Candidato = Presidente Codigo | Senador Codigo | Deputado Codigo
| Branco deriving (Show,Eq)

type Urna = [Candidato]
type Apuracao = [(Candidato, Int)]

urna = [(Presidente 5000), (Presidente 5001), (Senador 100), (Senador
101),
  (Deputado 50000), (Deputado 50001), (Deputado 50001), (Deputado
50001)]

totalVotos :: Urna -> Candidato -> Int
totalVotos u c = length (filter (\x -> x == c) u)

apurar :: Urna -> Apuracao
apurar [] =[]
apurar (c:cs) = (c, totalVotos (c:cs) c) : apurar (filter (\x -> x /=
c ) cs)
```