1. (5 p)

Lösning:

```
oddAndA::[String]->[String]
oddAndA [] = []
oddAndA (x:xs)
    |elem 'a' x && odd (length x) = x:oddAndA xs
    |otherwise = oddAndA xs
```

Rättning:

0p är minsta poäng som kan ges på varje uppgift.
-2p logiska fel (Upprepningar av samma fel ger inte ytterligare minus.)
-1p för varje slarvfel (men högst 1p för samma typ av slarvfel)
Inga avdrag för slarvfel där innebörden tydligt framgår t.ex. Intger istället Integer
Uppgiftsspecifik bedömning:

- Har både (1) likhetstecken och guards i en funktion eller (2) inga guards och likhetstecken: -1p.
- Skriver ‘‘a’’ istället för 'a' för en Char: -1p.
- Skriver and istället för &&: -1p
- Skriver [x:xs] istället för (x:xs): -1p
- Dubbelt likhetstecken istället för enkelt: -2p
- Skriver a som ett variabelnamn helt utan citattecken: -2p.

2. (6 p)

Lösning:

(a) `take` har typsignaturen `take :: Int -> [a] -> [a]` Det första argumentet, en Int n till `take` kombineras till en ny funktion som returnerar de första n elementen ur en lista.

Alternativ förklaring:
Om en funktion har flera parametrar så kommer Haskell att skapa delfunktioner som har de tidigare argumenten låsta och är redo att ta emot nästa argument.

(b) `takeFive = take 5` Partiell funktionsapplikation är ett sätt att skapa funktioner genom att ange en eller flera parametrar till en befintlig funktion. Den nya funktionen fungerar som den gamla men med de första parametervärdena låsta. I exemplet kommer `takeFive` att ta de första 5 elementen ur en lista.

Rättning:

0p är minsta poäng som kan ges på denna uppgift.
-2p logiska fel
-1p för varje slarvfel (men högst 1p för samma typ av slarvfel)
Inga avdrag för slarvfel där innebörden tydligt framgår t.ex. Intger istället för Integer eller Int med liten förstabokstav som i Java. Specifika bedömningar:

(a)
- Glömt `::` i typsignaturen: -1 p
- Använt `=>` som pilar istället för `->`: -1 p
- Förutsätter att listan innehåller någon specifik typ som `String` eller `Int`: -1 p
- Begränsar i onödan elementen till att vara `Foldable`: -1 p
- Nämner inte att det bildas en ny funktion: -2 p
- Förklarar inte att den nya funktionen bara tar en parameter: -1 p

(b)
- Exemplet använder inte partiell funktionsapplikation: -2 p
- Småslarv i exemplets syntax som `:` istället för `=` : -1
- Förklaringen missar att det skapas en ny funktion: -2 p
- Förklarar currying istället för partiell funktionsapplikation: -3 p

3. (4 p)

Lösning:

```
ramanujan :: Int -> Bool
ramanujan n =
  let upper = ceiling(fromIntegral(n)**(1/3))::Int in
    (length [(a, b)|a<-[1..upper],b<-[a..upper],a^3+b^3==n])>=2
```

Rättning:

0p är minsta poäng som kan ges på denna uppgift.
-2p logiska fel
-1p för varje slarvfel (men högst 1p för samma typ av slarvfel)
Inga avdrag för slarvfel där innebörden tydligt framgår t.ex. Intger istället för Integer eller Int med liten förstabokstav som i Java.

- Kollar inte att $a \leq b$ så att 5 anses vara ett Ramanujantal: -2 p
- Listomfattning saknas eller är inte central: -4 p
- Inget lodstreck "|" i listomfattningen: -1 p
- Listomfattningen har parenteser istället för hakklamrar: -1 p
- Inget avdrag för att sakna optimering.
- Inget avdrag för att sakna typsignatur.

4. (5 p)

Lösning:

```
powerset :: [a] -> [[a]]
powerset [] = [[]]
powerset (x:xs) = map (x:) (powerset xs) ++ powerset xs
```

```
powerset::Ord a => [a]->[[a]]
powerset list =
   powersetHelper list (length list)

powersetHelper::Ord a => [a]->Int->[[a]]
powersetHelper _0 = [[]]
powersetHelper list n =
   nub ((map (sort . (take n)) (permutations list)) ++ (powersetHelper list (n-1)))
```

```
powerset :: [a] -> [[a]]
powerset list = powersetHelper list []
powersetHelper :: [a] -> [a] -> [[a]]
powersetHelper [] set = [set]
powersetHelper (x:xs) set = (powersetHelper xs set) ++
   (powersetHelper xs (set++[x]))
```

```
import Control.Monad
powerset :: [a] -> [[a]]
powerset = filterM (const [True, False])
```

```
import Control.Monad
powersetLambda :: [a] -> [[a]]
powersetLambda = filterM (λ_- > [True, False])
```

Rättning:

0p är minsta poäng som kan ges på denna uppgift.
-2p logiska fel
-1p för varje slarvfel (men högst 1p för samma typ av slarvfel)
Uppgiftsspecifik bedömning:

- Har med samma delmängd två gånger, även då alla element är olika: -2 p
- Saknar bara en delmängd på ett litet fall (<4 element): -2 p
- Saknar två delmängder på ett litet fall (<4 element): -4 p
- Saknar mer än tre delmängder på ett litet fall (<4 element): -5 p
- Enbart hårdkodad för vissa storlekar på delmängder: -5 p
- Typsignaturen saknas: -2 p
- Typsignaturen kräver att listan är Foldable: -1 p
- Monadvarianten: Glömmer importera Control.Monad: -1 p
- Monaden igen: Använder mapM eller mapM_ istället för filterM -5 p.