2018 - 1

i) a)

```
fun union [], y = y
  | union (x::xs) y = if not (member x y) then union xs (x::y)
                      else union xs y;

fun intersection _ [] = []
  | intersection [] _ = []
  | intersection x::xs y = if member x y then x::(intersections xs y)
                           else intersection xs y;
```

Where the definition of member:

~~fun member e (x::xs)~~

```
fun member e [] = false
  | member e (x::xs) = if e=x then true
                       else member e xs;
```

~~b)~~ ii) union $O(n^2)$ as n recursive calls each time calling member which has $O(n)$ due to ~~due~~ n recursive calls

intersection $O(n^2)$ for the same reason as above

iii) ~~for unions ordered~~ either construct in order or sort after same fun as above (can also use halting linear search for member)

Sort after would give $O(n^2)$ as sorting can be done in $O(n\log n)$ if merge sort used so $O(n^2)$ still dominates

Not sure what best way of constructing the union set ~~in order~~ or intersection set in order.