```
module lib where

infix 4 _≤_ _<_ _≡_
infixl 6 _+_ _∸_ _+_ _-_
infixl 7 _*_

data ℕ : Set where
  zero : ℕ
  suc : ℕ → ℕ
{-# BUILTIN NATURAL ℕ #-}

data ℤ : Set where
  pos : ℕ → ℤ
  negsuc : ℕ → ℤ
{-# BUILTIN INTEGER          ℤ #-}
{-# BUILTIN INTEGERPOS       pos #-}
{-# BUILTIN INTEGERNEGSUC negsuc #-}

_+_ : ℕ → ℕ → ℕ
zero + n = n
suc m + n = suc (m + n)
{-# BUILTIN NATPLUS _+_ #-}

-- Monus (a∸b = max{a-b, 0})
_∸_ : ℕ → ℕ → ℕ
m ∸ zero = m
zero ∸ suc n = zero
suc m ∸ suc n = m ∸ n
{-# BUILTIN NATMINUS _∸_ #-}

_*_ : ℕ → ℕ → ℕ
zero * n = zero
suc m * n = n + m * n
{-# BUILTIN NATTIMES _*_ #-}

-- Relations of natural numbers
data _≡_ {a} {A : Set a} (x : A) : A → Set a where
  refl : x ≡ x
{-# BUILTIN EQUALITY _≡_ #-}

cong : ∀ {A B : Set} (f : A → B) {x y : A} → x ≡ y → f x ≡ f y
cong f refl = refl

sym : ∀ {A : Set} {x y : A} → x ≡ y → y ≡ x
sym refl = refl

sub : ∀ {A : Set} {x y : A} (P : A → Set) → x ≡ y → P x → P y
sub P refl px = px
```

```
trans : ∀ {A : Set} {x y z : A} → x ≡ y → y ≡ z → x ≡ z
trans refl refl = refl

-- n∸n≡0 : ∀ {n : ℕ} → n ∸ n ≡ zero
-- n∸n≡0 {zero} = refl
-- n∸n≡0 {suc n} = n∸n≡0 {n}

data _≤_ : ℕ → ℕ → Set where
  z≤n : ∀ {n : ℕ} → zero ≤ n
  s≤s : ∀ {m n : ℕ} → m ≤ n → suc m ≤ suc n

inv-s≤s : ∀ {m n : ℕ} → suc m ≤ suc n → m ≤ n
inv-s≤s (s≤s m≤n) = m≤n

≤-refl : ∀ {n : ℕ} → n ≤ n
≤-refl {zero} = z≤n
≤-refl {suc n} = s≤s ≤-refl

≤-trans : ∀ {m n p : ℕ} → m ≤ n → n ≤ p → m ≤ p
≤-trans z≤n _ = z≤n
≤-trans (s≤s m≤n) (s≤s n≤p) = s≤s (≤-trans m≤n n≤p)

n≤suc_n : ∀ {n : ℕ} → n ≤ suc n
n≤suc_n {zero} = z≤n
n≤suc_n {suc n} = s≤s n≤suc_n

-- ∸-≤ : ∀ {m n} → m ∸ n ≤ m
-- ∸-≤ {m} {zero} = ≤-refl
-- ∸-≤ {zero} {suc n} = z≤n
-- ∸-≤ {suc m} {suc n} = ≤-trans (∸-≤ {m} {n}) n≤suc_n

data _<_ : ℕ → ℕ → Set where
  z<s : ∀ {n : ℕ} → zero < suc n
  s<s : ∀ {m n : ℕ} → m < n → suc m < suc n

<→≤ : ∀ {m n : ℕ} → m < n → suc m ≤ n
<→≤ (z<s) = s≤s z≤n
<→≤ (s<s m<n) = s≤s (<→≤ m<n)

<-trans : ∀ {m n p : ℕ} → m < n → n < p → m < p
<-trans z<s (s<s _) = z<s
<-trans (s<s m<n) (s<s n<p) = s<s (<-trans m<n n<p)

-- here tried to make p implicit, but agda fails to infer the type for proof of n-n≡0
_-_ : (n : ℕ) → (m : ℕ) → (p : m ≤ n) → ℕ
(n - zero) (z≤n) = n
(suc n - suc m) (s≤s m≤n) = (n - m) m≤n
```

```
-→≤ : ∀ {n m} → {m≤n : m ≤ n} → (n - m) m≤n ≤ n
-→≤ {n} {zero} {z≤n} = ≤-refl
-→≤ {suc m} {suc n} {s≤s m≤n} = ≤-trans ((-→≤ {m} {n})) (n≤suc_n {m})

n-n≡0 : ∀ {n} → (n - n) (≤-refl {n}) ≡ 0
n-n≡0 {zero} = refl
n-n≡0 {suc n} = n-n≡0 {n}

-suc : ∀ {n m} → {m≤n : m ≤ n} → suc ((n - m) m≤n) ≡ (suc n - m) (≤-trans m≤n n≤suc_n)
-suc {_} {zero} {z≤n} = refl
-suc {suc n} {suc m} {s≤s m≤n} = -suc {n} {m} {m≤n}

n-_n-m≡m : ∀ {m n} → {m≤n : m ≤ n} → (n - ((n - m) m≤n)) (-→≤ {n} {m}) ≡ m
n-_n-m≡m {zero} {n} {z≤n} = n-n≡0 {n}
n-_n-m≡m {suc m} {suc n} {s≤s m≤n} = trans (sym (-suc {n} {(n - m) m≤n})) (cong suc (n-_n-m≡m

-- -suc : ∀ {m n} → {m≤n : m ≤ n} → suc (n - ≤→Fin m≤n) ≡ suc n - ≤→Fin (≤-trans m
-- -suc {zero} {_} {z≤n} = refl
-- -suc {suc m} {suc n} {s≤s m≤n} = -suc {m} {n} {m≤n}

-- n-_n-m≡m : ∀ {m n} → {m≤n : m ≤ n} → n - (≤→Fin (-→≤ {n} {≤→Fin m≤n})) ≡ m
-- n-_n-m≡m {zero} {n} {z≤n} = n-n≡0 {n}
-- n-_n-m≡m {suc m} {suc n} {s≤s m≤n}  = trans (sym (-suc {n} {n - ≤→Fin m≤n} { -→≤
```