**GOOD LUCK EVERYONE**

**👍🏻💪🏻**

**1a)**

Base Case: dup [] = dup2 [] [] = []

Inductive Hypothesis: ∀xs:[a].dup xs = xs

Inductive Step:

Dup x:xs = dup2 x:xs []

= dup2 xs ([] ++[x])

= dup2 xs ([x])

Aux lemma : ∀xs,ys:[a] dup2 xs ys = dup2 [] ys++xs

Prove over induction on xs

Base case:

Dup2 [] ys = dup2 [] ys ++[]

= dup2 [] ys

Inductive case:

IH : dup2 xs ys = dup2 [] ys++xs

Proof:

Dup2 x:xs ys

= dup2 xs (ys ++[x])

= dup2 [] (ys ++[x] ++xs)

= dup2 [] (ys ++ x:xs)

Therefore proven

= dup2 [] ([x] ++ xs)

= dup2 [] (x:xs)

= x:xs

**b)**

T1: ∀i:Int.∀c:Char.[P(C1 i c)] ^ P(C2) ^ ∀is:[Int].∀t:T1.[P(t) -> P(C3 is t)] -> ∀t1:T1.P(t1)

T2: ∀t:T1.[Q(C4 t)] ^ ∀t1,t2,t3:T2.[Q(t1) ^ Q(t2) ^ Q(t3) -> Q(C5 t1 t2 t3)] -> ∀t2:T2.Q(t2)

T3: ∀b1,b2:Bool.[R(C6 b1 b2)] ^

∀t1,t2:(T3 Bool Bool).[R(t1) ^ R(t2) -> R(C7 t1 t2)] ^

∀t:(T3 Bool Bool).∀bs:[Bool][R(t) -> R(C8 bs t)]

-> ∀t:(T3 Bool Bool).R(t)

Shouldn’t T3 be :

((∀a, b : Bool, R(C6 a b)) ∧ (∀t1, t2 : T3, R(C7 t1 t2)) ∧ (∀as : [Bool], ∀t : T3, R(C8 as t)) → (∀t3 : (T3 Bool Bool), R(t3))

?

**ci)**

[P(Lf)] ^

∀ts:[T].[∀t’:T(t’ ϵ ts -> P(t’)) -> P(Nd ts)]

-> ∀t:T.[P(t)]

**Or**

[P(Lf)] ^

∀ts:[T].[ *All*(ts) -> P(Nd ts)]

-> ∀t:T.[P(t)]

**ii)**

See Pages 12-15 in Generalized Induction Notes

C&D are base cases, A&B are inductive steps

[

∀t,t’,t’’:T[R(t, t’) ^ R(t’, t’’) ^ Q(t,t’) ^ Q(t’,t’’) -> Q(t, t’’)] ^

∀t,t’:T, ∀ts:[T] [R(t,t’) ^ Q(t,t’) -> Q(Nd(t:ts),Nd(t’:ts))] ^

∀ts:[T] [Q(Nd ((Nd []) : ts), Nd ts)] ^

∀ ts,ts’:[T] [Q(Nd ((Nd (Lf : ts)) : ts’), Nd ((Nd ts) : Lf : Lf : ts’))]

] -> ∀t,t’:T[R(t,t’)->Q(t, t’)]

**2a)**

Sorted(a[x..y)) <-> ∀i[x..y-1).(a[i] ≤ a[i+1])

***Or***

Sorted(a[x..y)) <-> ∀i,j[ x ≤ i ≤ j < y -> a[i]≤ a[j]]

**bi)**

I1<-> a!=null ^ a~a0 ^ done <-> Sorted(a[0..a.length))

Or maybe

I1<-> a!=null ^ a~a0 ^ done **->** Sorted(a[0..a.length))

Looks better

**ii)**

Prove INV + !cond -> Post

Given:

1. a!=null INV
2. a~a0 INV
3. done = Sorted(a[0..a.length)) INV
4. done !cond

To Prove:

1. a~a0
2. sorted(a[0..a.length))

Proof:

a follows from (2)

b follows from (3) and (4)

**c)**

Prove INV + cond + code -> INV’

Given:

1. a~a0 INV
2. 0 <= i < a.length INV
3. done -> Sorted(a[0..i + 1)) INV
4. i < a.length - 1 cond
5. done’ = done && ( a[i] <= a[i+1] ) code
6. i’ = i + 1 code
7. a~a’ implicit

To Prove:

1. a’~a0
2. 0 <= i’ < a’.length
3. Done’ -> Sorted(a’[0..i’ + 1))

Proof:

a follows from (1) and (7)

8) 0 <= i < a.length - 1 follows from (2) and (4)

9) 0 <= i + 1 < a.length arithmetic from (8)

10) 0 <= i’ < a.length follows from (9) and (6)

b follows from (10) and (7)

11) assume (done’) and we prove ( Sorted(a’[0..i’ + 1)) )

12) done from (5)

13) a[i] <= a[i+1] from (5)

14)Sorted(a[0..i+1)) from (3) & (11)

15) ∀j[x..i).(a[j] ≤ a[j+1]) from (14) & def. of *Sorted*, q. (a)

16) ∀j[x..i+1).(a[j] ≤ a[j+1]) from (13) & (15)

17) Sorted(a[0..i+2)) from (16) & def. of *Sorted*, q. (a)

18) Sorted(a’[0..i’+1)) from (17), (6), (7)

19) done’ -> Sorted(a’[0..i’+1)) by ass (11) and prove (18)

(c) follows from (19)

**di)**

V2 = a.length -1 - i

**ii)**

Prove Var >= 0 and Var’ < Var

Given:

1. a~a0 INV
2. 0 <= i < a.length INV
3. i’ = i+1 code
4. i < a.length - 1 cond
5. a~a’ implicit

To Prove:

1. a.length - 1 - i >= 0
2. a’.length - 1 - i’ < a.length - 1 - i

Proof:

5) 0 <= i <= a.length - 1 (2) and (4)

6) a.length - 1 - i >= 0 arithmetic (5)

a (6)

7) i’ > i arithmetic (3)

8) -i’ < -i arithmetic (7)

9) a.length - 1 - i’ < a.length - 1 - i arithmetic (8)

b follows from (9) and (5)

**e)**

It’s possible to shuffle and never get an ordered permutation of x

(basically bogo sort in disguise)

**fi)**

B is the kth unique way of ordering a

^

Q: Shouldn’t it be: The slice of the new array b from index 0 up to index k inclusive must be sorted ?

A: the precondition for shuffle includes 0 <= x < (a.length)! So k can’t represent an index, since obviously (a.length)! - 1 isn’t a valid index for a lot of lists. The factorial also hints that shuffle(int []x, intx) and the shuff predicate is related to permutations of the list.

**ii)**

Probably not right:

shuff(a0, a, x) <-> [ ∀k : [0, x) [( shuff(a0, a, k) <-> shuff(a0, a, x) )<-> x = 0]]

Probably still not right:

shuff (a, b, k) -> [shuff(a, b, k) -> b ~ a ∧ ∀j : N [j != k ∧ j < (a.length)! -> !shuff(a, b, j)] ]

Aren’t these last two exactly the same? Both look right to me too.

I think this one makes sense : shuff (a, b, k) = is b the kth permutation of a ?

shuff(a, b, k) <-> [b ~ a ∧ ∀j : N [shuff(a, b, j) -> j = k]]

∀a,b,c ∈ int[], ∀k1,k2 ∈ {0..(a.length)!}

k1 ≠ k2 ^ Shuff (a[..), b[..), k1) ^ Shuff (a[..), b[..), k2) -> b[..) ≠ c[..)

**iii)**

-line 5-

Int[ ] original = a // deep copy

Int ordering = 0

While (!done){

A = shuffle(original, ordering)

Ordering++

-line 11-

There is probably a better solution to f