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)] ^ $\forall t_1,t_2,t_3$:T2.[$Q(t_1)$ ^ $Q(t_2)$ ^ $Q(t_3)$ -> Q(C5 t_1 t_2 t_3)] -> $\forall t_2$:T2.$Q(t_2)$
T3: $\forall b_1,b_2$:Bool.[R(C6 b_1 b_2)] ^
$\forall t_1,t_2$:(T3 Bool Bool).[$R(t_1)$ ^ $R(t_2)$ -> R(C7 t_1 t_2)] ^
∀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, P(C6 a b)) ∧ (∀t1, t2 : T3, P(C7 t1 t2)) ∧ (∀as : [Bool], ∀t : T3, P(C8 as t)) →
(∀t3 : (T3 Bool Bool), P(t3))
?

ci)

$[P(Lf)] \wedge$
$\forall ts:[T].[\forall t':T(t' \in ts \rightarrow P(t')) \rightarrow P(Nd\ ts)]$
$\rightarrow \forall t:T.[P(t)]$

Or

$[P(Lf)] \wedge$
$\forall ts:[T].[\ \mathit{All}(ts) \rightarrow P(Nd\ ts)]$
$\rightarrow \forall t:T.[P(t)]$

ii)

See Pages 12-15 in Generalized Induction Notes
C&D are base cases, A&B are inductive steps

[
$\forall t,t',t'':T[R(t, t') \wedge R(t', t'') \wedge Q(t,t') \wedge Q(t',t'') \rightarrow Q(t, t'')]$
$\forall t,t':T,\ \forall ts:[T]\ [R(t,t') \wedge Q(t,t') \rightarrow Q(Nd(t:ts),Nd(t':ts))] \wedge$
$\forall ts:[T]\ [Q(Nd\ ((Nd\ []) : ts), Nd\ ts)] \wedge$
$\forall\ ts,ts':[T]\ [Q(Nd\ ((Nd\ (Lf : ts)) : ts'), Nd\ ((Nd\ ts) : Lf : Lf : ts'))]$
] $\rightarrow \forall t,t':T[R(t,t') \rightarrow Q(t, t')]$

2a)

$Sorted(a[x..y)) \leftrightarrow \forall i[x..y-1).(a[i] \le a[i+1])$

Or

$Sorted(a[x..y)) \leftrightarrow \forall i,j[\ x \le\ i \le j < y \rightarrow a[i] \le a[j]]$

bi)

$I_1 \leftrightarrow a!=null \wedge a\sim a_0 \wedge done \leftrightarrow Sorted(a[0..a.length))$

ii)

Prove INV + !cond -> Post

Given:

1)	a!=null	INV
2)	$a\sim a_0$	INV
3)	done = Sorted(a[0..a.length))	INV
4)	done	!cond

To Prove:

a) $a\sim a_0$
b) sorted(a[0..a.length))

Proof:

a	follows from (2)
b	follows from (3) and (4)

c)

Prove INV + cond + code -> INV'

Given:

1) $a \sim a_0$ 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

Asian eyes are small, im sorry. XD

To Prove:

a) $a' \sim a_0$
b) 0 <= i' < a'.length
c) 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 (16), (6), (7)

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

(c) follows from (19)

di)

V_2 = 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:

a) a.length - 1 - i >= 0
b) 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)]]

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]]

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