1.

分析：

出现`<->`，先用`split`分为两个方向的证明。

`->`：`intros`变量和假设，假设中有析取，又可以分出两个分支。第一个分支，我们可以证明结论中析取的`left`，用关键词`exists`，然后直接用到假设`Px`即可。第二个分支，`right`，`Qx`。

`<-`：思路大体与上面相似。

代码：

```
Theorem dist_exists_or : forall (X:Type) (P Q : X -> Prop),
  (exists x, P x \/ Q x) <-> (exists x, P x) \/ (exists x, Q x).
Proof.
  intros. split.
  - intros [x [Px | Qx]].
    * left. exists x. apply Px.
    * right. exists x. apply Qx.
  - intros [[x Px] | [x Qx]].
    * exists x. left. apply Px.
    * exists x. right. apply Qx.
Qed.
```

运行结果：

```
Theorem dist_exists_or : forall (X:Type) (P Q : X -> Prop),
  (exists x, P x \/ Q x) <-> (exists x, P x) \/ (exists x, Q x).
Proof.
  intros. split.
  - intros [x [Px | Qx]].
    * left. exists x. apply Px.
    * right. exists x. apply Qx.
  - intros [[x Px] | [x Qx]].
    * exists x. left. apply Px.
    * exists x. right. apply Qx.
Qed.
```

2.

分析：

按照要求构造一个”基线条件“和一个“递归”即可。

代码：

```
Inductive CE : nat -> nat -> Prop :=
  | CE_0 : CE 0 2
  | CE_S2 (n : nat) (m : nat) (H : CE n m) : CE (S(S n)) (S(S m)).

Example test_CE : CE 4 6.
Proof. apply CE_S2. apply CE_S2. apply CE_0.
Qed.
```

运行结果：

```
Inductive CE : nat -> nat -> Prop :=
  | CE_0 : CE 0 2
  | CE_S2 (n : nat) (m : nat) (H : CE n m) : CE (S(S n)) (S(S m)).

Example test_CE : CE 4 6.
Proof. apply CE_S2. apply CE_S2. apply CE_0.
Qed.
```

3.

分析：

通过使用`inversion`从假设`CE (S (S n)) (S (S m))`中得到假设`CE n m`，再`apply`即可。inversion的作用是如果你有一个假设，该假设声明两个构造式相等，并且两个构造式形式相同。`inversion`可以得出两个构造式的参数也必须相同，并且它通过以上信息试图`rewrite`证明目标。

代码：

```
Theorem CE_SS: forall n m, CE (S (S n)) (S (S m)) -> CE n m.
Proof. intros. inversion H. apply H2.
Qed.
```

运行结果：

```
Theorem CE_SS: forall n m, CE (S (S n)) (S (S m)) -> CE n m.
Proof. intros. inversion H. apply H2.
Qed.
```

4.

分析：

对`<->`采用`split`。再对链表作`induction`。然后对各个分支通过`simpl`、`left/right`、`destruct`等方法进行化简即可。反方向也类似。

代码：

```
Theorem In_app_iff : forall A l l' (a:A),
  In a (l++l') <-> In a l \/ In a l'.
Proof.
  intros. split.
  - induction l as [ | h t].
    * simpl. intro H. right. apply H.
    * simpl. intros [H | H].
      + left. left. apply H.
      + apply IHt in H. destruct H.
        { left. right. apply H. }
        { right. apply H. }
  - induction l as [ | h t].
    * intros [H | H].
      + simpl. inversion H.
      + simpl. apply H.
    * intros [H | H].
      + simpl. inversion H.
        { left. apply H0. }
        { right. apply IHt. left. apply H0. }
      + simpl. right. apply IHt. right. apply H.
Qed.
```

运行结果：

```
Theorem In_app_iff : forall A l l' (a:A),
  In a (l++l') <-> In a l \/ In a l'.
Proof.
  intros. split.
  - induction l as [ | h t].
    * simpl. intro H. right. apply H.
    * simpl. intros [H | H].
      + left. left. apply H.
      + apply IHt in H. destruct H.
        { left. right. apply H. }
        { right. apply H. }
  - induction l as [ | h t].
    * intros [H | H].
      + simpl. inversion H.
      + simpl. apply H.
    * intros [H | H].
      + simpl. inversion H.
        { left. apply H0. }
        { right. apply IHt. left. apply H0. }
      + simpl. right. apply IHt. right. apply H.
Qed.
```

5.

分析：

思路跟4较为类似。综合应用已经学习的各个tactic即可。

代码：

```
Fixpoint All {T : Type} (P : T -> Prop) (l : list T) : Prop :=
  match l with
    | [] => True
    | h :: t => P h /\ All P t
  end.


Theorem All_In :
  forall T (P : T -> Prop) (l : list T),
    (forall x, In x l -> P x) <->
    All P l.

```

```
Proof.
  intros. split.
  - induction l as [ | h t].
    + simpl. reflexivity.
    + intros. simpl. split.
      * apply H. simpl. left. reflexivity.
      * apply IHt. intros. apply H. simpl. right. apply H0.
  - induction l as [ | h t].
    + intros. inversion H0.
    + intros. simpl in H0. simpl in H.
      destruct H as [Ph APt].
      destruct H0 as [hx | Ixt].
      * rewrite <- hx. apply Ph.
      * apply IHt. apply APt. apply Ixt.
Qed.
```

运行结果：

```
Fixpoint All {T : Type} (P : T -> Prop) (l : list T) : Prop :=
  match l with
    | [] => True
    | h :: t => P h /\ All P t
  end.

Theorem All_In :
  forall T (P : T -> Prop) (l : list T),
    (forall x, In x l -> P x) <->
    All P l.
Proof.
  intros. split.
  - induction l as [ | h t].
    + simpl. reflexivity.
    + intros. simpl. split.
      * apply H. simpl. left. reflexivity.
      * apply IHt. intros. apply H. simpl. right. apply H0.
  - induction l as [ | h t].
    + intros. inversion H0.
    + intros. simpl in H0. simpl in H.
      destruct H as [Ph APt].
      destruct H0 as [hx | Ixt].
      * rewrite <- hx. apply Ph.
      * apply IHt. apply APt. apply Ixt.
```