Q4: Semantics and Type Systems [20 pts]

In this part, you will answer questions about the operational semantics and the type system of Nano, both given in Appendix II at the end of the exam. In each question, mark **all** the answers that apply; it is possible that *none*, *some*, or *all* of the answers are correct.

3.1 Evaluation 1 [5 points]

Which of these evaluation relations are valid according to the operational semantics of Nano?

(A) `[] ; 5 ==> 5` [ ]

(B) `[] ; (\x -> 5) ==> 5` [ ]

(C) `[] ; (\x -> x) (2 + 3) ==> 5` [ ]

(D) `[] ; (\x -> x x) ==> <[], x, x x>` [ ]

(E) `[] ; 1 + (\x -> x) ==> <[], x, 1 + x>` [ ]

3.2 Evaluation 2 [5 points]

Which of the following rules are used in the derivation of the reduction

```
[] ; \x y -> x y  ==>  ???
```

(A) `E-Num` [ ]

(B) `E-Var` [ ]

(C) `E-Add` [ ]

(D) `E-Lam` [ ]

(E) `E-App` [ ]

3.3 Typing 1 [5 points]

Which of the following typing judgments are valid according to the type system of Nano?

(A) `[] |- \x -> x :: Int -> Int` [ ]

(B) `[] |- \x -> x :: Int -> Int -> Int` [ ]

(C) `[] |- \x -> x :: (Int->Int) -> (Int->Int)` [ ]

(D) `[] |- x :: Int` [ ]

(E) `[x: Int] |- x :: Int` [ ]

3.4 Typing 2 [5 points]

Which of the following rules are used in the derivation of the typing judgment

```
[] |- \x y -> x y :: (Int -> Int) -> Int -> Int
```

(A) `T-Num` [ ]

(B) `T-Var` [ ]

(C) `T-Add` [ ]

(D) `T-Lam` [ ]

(E) `T-App` [ ]

Solution

3.1 Reduction 1 [5 points]

(A) `[] ; 5 ==> 5` [X]

(B) `[] ; (\x -> 5) ==> 5` [ ]

(C) `[] ; (\x -> x) (2 + 3) ==> 5` [X]

(D) `[] ; (\x -> x x) ==> <[], x, x x>` [X]

(E) `[] ; 1 + (\x -> x) ==> <[], x, 1 + x>` [ ]

3.2 Reduction 2 [5 points]

```
[] ; \x y -> x y  ==>  <[], x, \y -> x y>
```

(A) `E-Num` [ ]

(B) `E-Var` [ ]

(C) `E-Add` [ ]

(D) `E-Lam` [X]

(E) `E-App` [ ]

3.3 Typing 1 [5 points]

(A) `[] |- \x -> x :: Int -> Int` [X]

(B) `[] |- \x -> x :: Int -> Int -> Int` [ ]

(C) `[] |- \x -> x :: (Int->Int) -> (Int->Int)` [X]

(D) `[] |- x :: Int` [ ]

(E) `[x: Int] |- x :: Int` [X]

3.4 Typing 2 [5 points]

```
[] |- \x y -> x y :: (Int -> Int) -> Int -> Int
```

(A) `T-Num` `[ ]`

(B) `T-Var` `[X]`

(C) `T-Add` `[ ]`

(D) `T-Lam` `[X]`

(E) `T-App` `[X]`