**Assignment#2 Key; Due February 3 at start of class**

Let set **A** be non-empty recursive, **B** be re non-recursive and **C** be non-re. Using the terminology **(REC)** **recursive**, **(RE) non-recursive recursively enumerable**, **(NR)** **non-re**, categorize each set below, saying whether or not the set can be of the given category and justifying each answer. You may assume, for any set **S**, the existence of comparably hard sets
**SE = {2x|x∈S}** and **SD = {2x+1|x∈S}**. The following is a sample of the kind of answer I require:

**Sample.) A ∩ C = { x | x ∈ A and x ∈ C }**

**REC: Yes. If A = {0} then A ∩ C = ∅ or {0}, each of which is in REC.**

**RE: Yes. Let A = ℵE = { 2x | x ∈ ℵ }; let C = TOTD ∪ HALTE then A ∩ C = HALTE which is in RE**

**NR: Yes. If A = ℵ then A ∩ C = C, which is in NR.**

Be careful: For your problems, some options (**REC**, **RE** or **NR**) may not be possible. If so, you must justify why this is so.

**a.) A – B = { x | x ∈ A and x ∉ B } // Set difference**

**REC: Yes. Let A = {0}, B = HALT, then A – B = ∅ or A – B = {0}, depending on whether ϕ0(0)↑ or ϕ0(0)↓. In either case the resulting set is in REC**

**RE: No. If A – B is RE but non-recursive then its complement
~(A – B) = ~(A ∩~B) = (~A ∪ B) could not be RE, for if it were then A – B is recursive as both it and its complement are RE. However, ~A ∪ B is the union of two RE sets (~A is recursive and B is RE) and so A – B is RE and thus A – B is recursive, not just RE.**

**NR: Yes. Let A = ℵ, B = HALT, then A – B = { x | x ∉ HALT }. This set is co-re, non-re and so is in NR.**

**b.) min(A, B) = { min(x , y) | x ∈ A and y ∈ B } // Minimum**

**REC: Let A = {0}, B = HALT, then min(A, B) = {0} , which is in REC**

**RE: Let A = ℵE, B = HALTD, then min(A, B) = A∪ B = ℵE ∪ HALTD , which is in RE**

**NR: This is not possible. To see this, we need just show that we can semi-decide min(A, b). By definition x ∈ min(A, B) iff ∃ a∈A, b∈B such that x = min(a, b). But, this implies we can limit our search to values of a,b such 0≤a≤x and 0≤b≤x, so we have bounds on each value. For each x, check if A includes x. If so, run the enumerating function for B to see if any value greater than x shows up. If any does, answer “yes.” If A does not contain x, run the enumerating function for B looking for x. If x shows up, then run tests to see if A contains any value greater than x. If both of these tests are satisfied answer “yes.” In all cases other than those for which we answer “yes” we would diverge. This provides a semi-decision procedure and hence max(A,B) cannot be in NR..**

**c.) A ⊕ C = { x | x ∈ A or x ∈ C but x ∉ A ∩ C } // Set exclusive union**

**REC: This is not possible. To see this, assume A ⊕ C is in REC and let x be an arbitrary element of ℵ. Since A ⊕ C and A are both in REC, we can check to see if x is a member of either or both. A simple case analysis shows that C will also need to be in REC.
case 1: x is in A ⊕ C and x is in A. In this case we can state that x cannot be in C.
case 2: x is in A ⊕ C and x is not in A. In this case we can state that x must be in C.
case 3: x is not in A ⊕ C and x is in A. In this case we can state that x must be in C.
case 4: x is not in A ⊕ C and x is not in A. In this case we can state that x cannot be in C.
This means that if A is in REC and A ⊕ C is in REC, then C is also in REC. Thus, C cannot be in NR.**

**A shorter approach is that if A and A ⊕ C are both in REC then so is A ⊕ (A ⊕ C), since the REC sets are closed under exclusive or. However, since exclusive or is associative
A ⊕ (A ⊕ C) = (A ⊕ A) ⊕ C = C and so C is in REC.**

**RE: Let A = ℵ, C = ~HALT, then A ⊕ C = HALT, which is in RE**

**NR: Let A = {0}, C = ~HALT, then A ∪ C = ~HALT∪{0} or ~HALT- {0}, each of which is in NR**

Note:

**TOT = { x | ∀ ϕx (y) ↓ }**. These are the indices of the set of algorithms.

**HALT = { <x,y> | ϕx (y) ↓ }**. This is the set of pairs of procedures and input for which the given procedure halts.

The set **SE**, for any set **S**, is defined as **{2x | x ∈ S }**

The set **SD**, for any set **S**, is defined as **{ 2x+1 | x ∈ S }.**

The complexities of **S**, **SE** and **SD** are the same. That is, all three are either in **REC**, **RE** or **NR**.