THE TURING DEGREES AND THEIR LACK OF LINEAR ORDER

JASPER DEANTONIO

Abstract. This paper is a study of the Turing Degrees, which are levels of incomputability naturally arising from sets of natural numbers. I explore the structure of the Turing Degrees, concluding with a proof that there exist Turing Degrees which are not comparable so as to illustrate the nonlinearity of the Turing Degrees.

Contents

1. An Introduction to Computability

Computability Theory is the study of degrees of computability. For this paper, we shall specifically study the computability of sets of the natural numbers. Intuitively, a computable set is a set whose elements can be determined with finitely many precise directions, i.e., with a computer program. Thus, one can determine both what is and what is not in the set in finitely many steps. So, for a set to be computable, the complement of the set must also be computable. For instance, the prime numbers are computable: if we fix $n \in \mathbb{N}$, we simply check every number less than n and greater than 1 to see if that number divides n. So, in $n-2$ steps, we know whether or not n is prime. Thus, the primes are computable; for any natural number, we can, in finitely many steps, tell whether or not the number is prime.

In order to rigorously define this intuitive notion, we shall use a structure called the Turing Machine:

Definition 1.1. Turing Machine
A Turing machine uses five substructures:

- A Tape T: An infinite strip with an infinite line of boxes on it. Each box can contain a symbol.

Date: September 24, 2010.

- Tape Symbols $S = \{s_1, s_2, s_3, ..., s_n\}$: These symbols are the only things which can appear on the tape. There are only finitely many symbols allowed.
- A Reading Head: The reading head is positioned at one box on the tape. It is the only object that can move and perform actions in the Turing machine.
- List of Internal States $Q = \{q_1, q_2, q_3, ...\}$: These internal states are necessary for the reading head to act. At the start of any given step of a computation, in addition to being at a box on the tape, the reading head must be in one and only one of the internal states.
- Action Symbols: These are symbols used to represent the actions which the reading head can take. These actions are: L to move the reading head left, R to move it right, 0 to have it print s_1 to the current place, 1 to print s_2, etc.

The Turing machine T's program consists of finite quadruples q_iSAq_j, where q_i and q_j are internal states, S is a symbol on the tape, and A is an action. The quadruple encodes the following process: if the reading head is at state q_i reading the symbol S, then perform action A and go to state q_j. The Turing machine halts when there is no applicable quadruple. We also require that a Turing machine be consistent, i.e., if q_iSAq_j and $q_iSA'q_k$ both exist, then $A = A'$ and $q_j = q_k$. Because the main difference between any two Turing machines is their program, we shall commonly only define the program of a Turing machine when defining the Turing machine and leave the rest of the definition to the reader to complete.

A Turing machine is like a function; in order to use it, one must give the Turing machine a specific piece of tape as input and then read the tape afterwards to receive the output. In order to have the Turing machine start with input n, we give it a tape with $n + 1$ ones on it and start the machine at the left most one. Then, if the Turing machine ever halts, we count the number of ones. This is the output. Note that this is very similar to a function over the natural numbers; one gives the Turing machine a tape representing a natural number and receive a tape representing a natural number. To denote this function associated with a Turing machine T, we write ϕ_T.

Note that the Turing machine mimics the actions of a computer running a program very closely. Therefore, in order to define computability, we shall use this Turing machine.

Definition 1.2. Computable Functions, Sets, and Relations
We define a **computable function** to be a function $f : \mathbb{N} \to \mathbb{N}$ such that $f = \phi_T$ for some Turing machine T.
We define a **computable set** to be a set $A \subset \mathbb{N}$ such that the characteristic function of A, χ_A, is computable.
Recall that a relation can be considered as a set of n-tuples. Thus, we define a **computable relation** to be a relation R whose corresponding set of n-tuples is computable.

Example 1.3. The zero function is computable
Intuivitely, this is simple: consider the program which just returns 0. In order to show this rigorously, we must create a Turing machine which takes any input and returns zero.

Let T be a Turing machine with quadruples: $q_0 1 0 q_1$ and $q_1 0 R q_0$. Now, consider what happens at input n. We input a tape with $n+1$ ones into the Turing machine. At q_0, the starting state, the reading head encounters 1, so we use our first quadruple. So, we write 0 on the tape and move to state q_1. Next, we use our second quadruple because we encounter q_1 and 0. This causes us to move right, where we'll find another 1. Thus, one can see how this process can turn any tape of consecutive ones into a tape containing only zeros, making the output zero, which is precisely what we desire.

How does the intuitive computability described at the beginning of this section relate to Turing computable? One might think that the intuitive idea is more broad. In fact, by the *Church-Turing Thesis*, these two notions are equivalent. Therefore, we shall justify computability from now on using our intuitive notion as opposed to Turing computability.

Next, we consider how many computable functions there are.

Lemma 1.4. *There are countably many computable sets.*

Proof. Every computable set has a unique computable function. Every computable function is expressible as a Turing machine. Every Turing machine has a program which is composed of finitely many quadruples of the form $q_i S A q_j$ where q_i and q_j are internal states, S is a symbol on the tape, and A is an action. There are countably many internal states, finitely many symbols, and finitely many actions (L,R, and one for each symbol). Thus, there are countably many quadruples available and each Turing machine can only choose finitely many of these. Therefore, there are only countably many Turing machines and thus countably many computable sets.
Note that we could also prove this lemma using the intuitive computability discussed earlier with a very similar argument. □

Because there are countably many computable functions, we can put these functions into a one-to-one correspondence with the natural numbers as follows:

Definition 1.5. ϕ_i **and** W_i
We define ϕ_i to be the i-th computable function.
We define W_i to be the domain of ϕ_i.

Using this defintion, we can enumerate all computable functions and domains of these functions, giving us much more control over these computable functions.

2. Computably Enumberable Sets and K_0

Having considered what is computable, we now explore what is not computable. In order to do so, we define a new notation:

Definition 2.1. The pairing function
We define a **pairing function** to be a computable bijection function $<,>: \mathbb{N}^2 \to \mathbb{N}$.

Using these pairing functions, we can examine subsets of $\mathbb{N}^2$ as subsets of $\mathbb{N}$ and thus study their computability. In fact, using multiple pairing functions, for any k, we can consider the computability of subsets of $\mathbb{N}^k$.

Specifically, we study the set $K_0 = \{\langle i, x\rangle | \phi_i(x) \text{ halts }\}$. This set is known as the Halting Problem.

At first one may be confused about why we even consider programs which don't halt. Recall that one can easily write a computer program which does not halt; a simple example is any loop that has no end condition. These are still computer programs and can be encoded as Turing machines. Thus, there exists $m \in \mathbb{N}$ such that ϕ_m is exactly a program which, given every input, loops infinitely. Therefore, in studying the computable functions, we must be aware of functions which do not halt, which is one reason why we shall study K_0.

Theorem 2.2. *K_0 is not computable*

Proof. Assume K_0 is computable. Then, $\chi_{K_0}(i, n) = 0$ or 1 for all $i, n \in \mathbb{N}$, i.e., for all i, n, we can computably tell whether or not $n \in W_i$. Therefore, we can define the function $H : \mathbb{N} \to \mathbb{N}$ as follows:

$$H(i) = \begin{cases} 1 & \chi_{K_0}(i, i) = 0 \\ 0 & \chi_{K_0}(i, i) = 1 \end{cases}$$

H is computable since χ_{K_0} is. Therefore, there exists $e \in \mathbb{N}$ such that $H = \phi_e$ because we can enumerate all the computable functions. We then consider $H(e)$.

$$H(e) = \begin{cases} 1 & \chi_{K_0}(e, e) = H(e) = 0 \\ 0 & \chi_{K_0}(e, e) = H(e) = 1 \end{cases}$$

Either way, $H(e) = 1 \neq 0 = H(e)$, which is a contradiction. This illustrates that K_0 cannot be computable. □

Remark 2.3. Since K_0 is not computable, what is it? It seems like K_0 is almost computable, as we can enumerate the elements of K_0 using the following process: Step through the triples $\langle i, x, s\rangle$, running $\phi_i(x)$ for s steps. If $\phi_i(x)$ halts in s steps, add $\phi_i(x)$ to K_0.
Unfortunately, this is not all we need for K_0 to be computable. The problem is that we cannot computably determine if any pair is not in K_0. On a pair $\langle i, x\rangle$, if $\phi_i(x)$ has not halted in s steps, we don't know if it will halt on the $(s + 1)$-th step or never halt. Thus, we can never state that a pair $\langle i, x\rangle$ is not in K_0 because we cannot look ahead an infinite number of steps to see if $\phi_i(x)$ ever halts. Intuitively, this is why K_0 is not computable.

Because K_0 is incomputable, it cannot be generated by any computer. No possible algorithm could give the exact set which is K_0. Yet, K_0 is a relatively standard set in the sense that it is easy for mathematicians to understand and simple for us to construct. This illustrates the hidden difficulty of actually creating many of the sets which mathematicians commonly use.

Definition 2.4. Computably Enumerable
We define a **computably enumerable set** (abbreviated c.e. set) to be a set $A \subset \mathbb{N}$ such that $A = \text{range}(f)$ for some computable function f. Intuitively, this means that the elements of A can be enumerated using f. So, for every $a \in A$, there are $n, s \in \mathbb{N}$ such that, after s steps, $f(n)$ halts and $f(n) = a$.

Example 2.5. K_0 and W_i are c.e.
K_0 is a c.e. set, since, using the process described in Remark 2.3, the elements of K_0 can be enumerated.

Let $g : \mathbb{N}^2 \to \mathbb{N}$ such that $g(x, s)$ does the first s steps of ϕ_i on input x, returning x if ϕ_i halts.
Let $h : \mathbb{N} \to \mathbb{N}^2$ such that h is bijective. Then, if we run $(g \circ h)(n)$ for each $n \in \mathbb{N}$ starting with $n = 1$, we will be able to enumerate the elements of W_i. Therefore, W_i is c.e.

Next we provide a few theorems about computably enumerable sets to give the reader a better understanding of them.

Theorem 2.6. *Every computable set is c.e.*

Proof. Let A be a computable set. We want to show that A is c.e. so we will find a computable f such that $f(\mathbb{N}) = A$.
If $A = \emptyset$, then let f be the Turing machine that just loops infinitely for any input. Then, f never returns any value. Thus, $f(\mathbb{N}) = \emptyset$.
Now, assume $A \neq \emptyset$. Let $a \in A$. Then, since χ_A is computable (because A is a computable set), consider the function $g : \mathbb{N} \to \mathbb{N}$:

$$g(n) = \begin{cases} n & \chi_A(n) = 1 \\ a & \chi_A(n) = 0 \end{cases}$$

Then $g(\mathbb{N}) = A$. g is computable since χ_A is computable. Thus, A is c.e. □

There is another way to describe sets which are c.e., as we illustrate below:

Definition 2.7. Existential Sets
We define an **existential set** to be a set $A \subset \mathbb{N}$ such that for all x, $x \in A$ if and only if there exists y such that $R(x, y)$ holds for some computable relation R.

Example 2.8. K_0 is an existential set.
$\langle i, x\rangle \in K_0$ if and only if there exists $s \in \mathbb{N}$ such that $\phi_i(x)$ halts in s steps. Thus, our relation here is composed of three variables i, x, s. So, $\langle i, x\rangle \in K_0$ exactly when (i, x, s) is in this relation R. Note that R is computable since we only run the algorithm s times on the given input, then see if it has halted or not.

Theorem 2.9. *A set $A \subset \mathbb{N}$ is c.e. iff A is a existential set.*

Proof. First we show that if $A \subset \mathbb{N}$ is c.e., then A is a existential set.
If A is c.e., then A is the range of some computable function f. So,

$$x \in A \iff (\exists s)[f(s) = x]$$

Since f is computable, this relation is computable as well. Therefore, A is existential due to this relation.
Next, we show that if A is a existential set, then $A \subset \mathbb{N}$ is c.e.
If A is an existential set, then we have, for some computable relation R,

$$x \in A \iff (\exists s)[R(s, x)]$$

Define the function $\psi : \mathbb{N} \to \mathbb{N}$ as follows:

$$\psi(x) = \begin{cases} 0 & (\exists s)[R(s, x)] \\ \text{loop infinitely} & \text{otherwise} \end{cases}$$

ψ is an algorithm a computer could reproduce given the relation R. But the relation R is computable. So, ψ is also computable. Therefore, the domain of ψ, which is the set A, is equal to W_e for some $e \in \mathbb{N}$. But, as stated in example 2.5, W_e is c.e. Therefore, A is also c.e. □

3. Computing with Oracles and the Turing Equivalence

Not only does Computability Theory study computable and incomputable sets, it also examines computability in relation. Two incomputable sets can have different degrees of incomputability. Intuitively, we want to be able to tell whether or not, given two subset of the natural numbers, A, B, if A is computable using B as a black box or vice versa. Our intuitive program would be able to ask if an element n of $\mathbb{N}$ is in B and make computable choices based on the answer. This is how the black box mechanism would work. Here, B is called an **oracle**. We revise our Turing machine to rigorously allow for this type of relative comparison.

Definition 3.1. Oracle Turing Machine
We define an **oracle Turing machine** to be a Turing machine with an additional type of quadruple, called a query quadruple, which is defined below:
We define a **query quadruple** to be a quadruple of the form $q_i S_k q_j q_l$ where S_k is a tape symbol and q_i, q_j, and q_l are internal states. If the reading head gets to internal state q_i with tape symbol S_k, the reading head counts the number of ones on the tape (say, n). Then, it queries to the oracle, say A, if $n \in A$. If yes, then the reading head goes to state q_j, otherwise it goes to state q_l.
These query quadruples do exactly what we intuitively wanted to do; they allow the Turing machine to ask questions about the oracle and make a decision based upon this input. As stated by the *Relativised Church-Turing Thesis*, again our intuitive definition aligns with our Turing machine defintion of relatively computable. So, while this definition is provided to lend rigor to the argument, we will not reference the actual quadruples of a oracle Turing machine and instead use intuitive programs.

Remark 3.2. As in Lemma 1.4, there are countably many A-computable sets for any $A \subset \mathbb{N}$ (when we create our Turing machine, we have the exact same number of characters, except we can also ask the oracle. But this is only one additional operation, so there are still only countably many programs). Thus, we enumerate these A-computable sets as follows:

Definition 3.3. ϕ_i^A **and** W_i^A
We define ϕ_i^A to be the function computed by the i-th Turing machine with oracle A.
We define W_i^A to be the domain of ϕ_i^A.

Definition 3.4. Relative Computability and Turing Equivalence
Given $A, B \subset \mathbb{N}$, we say A is B**-Turing computable** if A is computable with oracle B. We write this $A \leq_T B$.
We say that A is **Turing equivalent** to B, denoted $A \equiv_T B$, if $A \leq_T B$ and $B \leq_T A$.

Example 3.5. For any computable set A, $A \leq_T B$ for all $B \subset \mathbb{N}$.
Fix $B \subset \mathbb{N}$.
We need to show that A is B-computable, i.e., that there exists a computable function which uses B as an oracle and is the characteristic function of A. But, since A is computable, there already exists a computable function f which is the characteristic function of A. Then, we can let B be an oracle for f, but compute f the exact same way. Thus, $A \leq_T B$.

Example 3.6. For all $e \in \mathbb{N}$, $W_e \leq_T K_0$
If $x \in \mathbb{N}$, $x \in W_e$ if and only if $\langle e, x\rangle \in K_0$. Therefore, W_e is K_0-computable, and $W_e \leq_T K_0$

We can now generalize the notion of the Halting Problem, or K_0, as shown below.

Definition 3.7. The Jump Operator
We define the **jump** of a set $A \subset \mathbb{N}$ to be $A' = \{\langle i, x\rangle : W_i^A(x) \text{ halts}\}$
Specifically, note that $K_0 = \emptyset'$
Also, given that $K_0 = \emptyset'$, we can then take the jump of $K_0 = \emptyset''$. We can continue this process indefinitely, yielding countably many sets, $\{\emptyset^{(n)}\}_{n\in\mathbb{N}}$. Similar to the proof of $\emptyset < \emptyset'$ as seen in Theorem 2.1, for every $n \in \mathbb{N}$, $\emptyset^{(n)} <_T \emptyset^{(n+1)}$. Therefore, each $\emptyset^{(n)}$ is less computable than every $\emptyset^{(m)}$, where $m < n$. So, we have a sequence $\{\emptyset^{(n)}\}_{n\in\mathbb{N}}$ which gets less and less computable without ever stopping.

Lemma 3.8.
(1) Relative computability is a transitive relation, as is Turing equivalence.
(2) The Turing equivalence is well defined, i.e., if $A \leq_T B$ and $A \equiv_T C$ and $B \equiv_T D$, then $C \leq_T D$

Proof. (1)Let $A \leq_T B$ and $B \leq_T C$. Then, for any $n \in \mathbb{N}$, we can computably determine whether or not $n \in A$ using answers to finitely many questions "Is $m_0 \in B$?"', "Is $m_1 \in B$?"',..."Is $m_k \in B$?"'. Since each of these questions can be computably answered using answers to finitely many questions "Is $p_0 \in C$?"', "Is $p_1 \in C$?"', ... "Is $p_l \in C$?"', we can computably determine whether or not $n \in A$ using answers to finitely many questions about elements of C. Thus, A is C-computable, and $A \leq_T C$.
As Turing equivalence is based upon relative computability, it follows immediately that if $A \equiv_T B$ and $B \equiv_T C$, then $A \equiv_T C$.
(2)If $A \leq_T B$, then there exists a computable function f using B as an oracle which computes A. So, for any $n \in \mathbb{N}$, $f(n)$ B-computably returns a 1 or a 0 if $n \in A$ or $n \notin A$, respectively. Similarly, since $A \equiv_T C$ and $B \equiv_T D$, $C \leq_T A$, $A \leq_T B$, and $B \leq_T D$. Therefore, by part 1 of this lemma, $C \leq_T D$. □

Example 3.9. Let $L = \{e : W_e \neq \emptyset\}$. Then, $K_0 \equiv_T L$.
First we show that $K_0 \leq_T L$.
Consider the pair $\langle i, n\rangle$. We want to know L-computably if $\langle i, n\rangle \in K_0$, i.e., if n is in the domain of ϕ_i. In order to do so, we computably define a function ϕ_i' as follows:

$$\phi_i'(k) = \begin{cases} \text{Loop infinitely} & k \neq n \\ \phi_i(n) & k = n \end{cases}$$

Because this program ϕ_i' is Turing computable (due to the fact that $\phi_i(n)$ is), there exists an $e \in \mathbb{N}$ such that $\phi_e = \phi_i'$. So, W_e is the domain of ϕ_i'. For every natural number x that is not equal to n, $\phi_i'(x)$ does not halt, so $x \notin W_e$. If $\phi_i(n)$ halts, then $\{n\}$ is the domain of ϕ'. Therefore, $W_e \neq \emptyset$, and so $e \in L$. Otherwise $W_e = \emptyset$, so $e \notin L$. So, to determine if $\langle i, n\rangle \in K_0$, we need only ask if e is in L. Thus, $K_0 \leq_T L$.
Next, we show that $L \leq_T K_0$
Note that $x \in L$ if and only if there exists $\langle n, s\rangle$ such that $\phi_x(n)$ halts in s steps. Therefore, L is an existential set. By Theorem 2.7, there exists $e \in \mathbb{N}$ such that $W_e = L$. Since $W_e \leq_T K_0$ for all e, $L \leq_T K_0$.
Therefore, $L \leq_T K_0$ and $K_0 \leq_T L$, so $K_0 \equiv_T L$.

Remark 3.10. Since $L \equiv_T K_0$ and $\emptyset <_T K_0$, $\emptyset <_T L$. Therefore, L is not computable. Yet, we can compute L from K_0. This illustrates the power of relative computability. It allows us to use the otherwise-inaccessible information of K_0 to computably generate incomputable sets. So, we could say that K_0 has some computing ability, or information, stored in its set. In fact, any incomputable set A has some of this information, as there are other incomputable sets which can be A-computed. Since any two Turing equivalent sets can generate each other computably, any two Turing equivalent sets have the same amount of information. Thus, we shall shift our study to these sets of Turing equivalent sets so as to better understand these varying degrees of information.

Definition 3.11. Turing Degrees
(1) We define the **Turing Degree** of a set $A \subset \mathbb{N}$, denoted $\deg(A)$, to be the set $\{X \subset \mathbb{N} : X \equiv_T A\}$.
(2) We define $\deg(\emptyset)$ to be the Turing degree of the computable sets.
(3) We define $\mathcal{D}$ to be the set of all Turing degrees, i.e., $\mathcal{D} = \{\deg(A) : A \subset \mathbb{N}\}$
(4) We define the operator $\leq$ for Turing Degrees A and B as follows:
$A \leq B$ if there exists $X \in A$ and $Y \in B$ such that $X \leq_T Y$. Note that for all $X' \in A$, $Y' \in B$, $X' \leq_T Y'$ by Lemma 3.8 so long as $X \leq Y$. So, when $A \leq B$, every set in A is B-computable.

Example 3.12. The Least Turing Degree
We show that $\deg(\emptyset)$ is the least Turing Degree, i.e., that $\deg(\emptyset) \leq \deg(A)$ for all $A \subset \mathbb{N}$.
Let X be a computable set and $A \subset \mathbb{N}$. Then, $X \leq_T A$ by Example 3.4. Therefore, since $X \in \deg(\emptyset)$, $\deg(\emptyset) \leq \deg(A)$ forall $A \subset \mathbb{N}$.

Theorem 3.13. *$\mathcal{D}$ has uncountably many degrees.*

Proof. For ease of notation in this proof, for any set A, let $\text{car}(A)$ be the cardinality of A.
We prove this theorem by contradiction. Assume that $\mathcal{D}$ has countably many degrees.
Then, note that for every $A \subset \mathbb{N}$, $A \in \deg(B)$ for some $B \subset \mathbb{N}$. Therefore,

$$\text{car}(\mathcal{P}(\mathbb{N})) \leq \text{car}(\mathcal{D}) \times \sup\{\text{car}(\deg(A)) : A \in \mathbb{N}\}$$

As stated in Remark 3.2, for any set $A \subset \mathbb{N}$, the set of A-computable sets is countable. So, $\sup\{\text{car}(\deg(A)) : A \in \mathbb{N}\}$ is countable since each $\deg(A)$ is countable. Thus, $\mathcal{D} \times \sup\{\deg(A) : A \in \mathbb{N}\}$ must be countable, as a countable set cross a countable set is still countable. Then, $\mathcal{P}(\mathbb{N})$ must also be countable. This is a contradiction, as $\mathcal{P}(\mathbb{N})$ is uncountable. Therefore, $\mathcal{D}$ must have uncountably many degrees. □

Corollary 3.14. *$\mathcal{D}$ has no greatest element.*

Proof. We prove this by contradiction. Assume that $\mathcal{D}$ has a greatest element. Then, there exists $A \in \mathcal{D}$ such that $B \leq A$ for all $B \in \mathcal{D}$. Let $X \in A$. The set of X-computable sets is countable. Therefore, the set of Turing degrees less than A is also countable. However, the set of Turing degrees less than A is $\mathcal{D}$. By the above theorem, $\mathcal{D}$ is uncountable. Thus, we arrive at a contradiction. So, $\mathcal{D}$ cannot have a greatest element. □

4. The Turing Degrees are not Linearly Ordered by the Relation $\leq$

In order to prove that the Turing Degrees are not linearly ordered by $\leq$, we will first need some more definitions.

Definition 4.1. Strings
(1) We define a **string**, generally denoted σ or τ, to be a finite sequence of 0's and 1's. For example 01001010 is a string. Note that a string can also be considered as the beginning of a characteristic function of a set. For example, for some set A, the string 01001010 would designate that $0 \notin A$, $1 \in A$, $2 \notin A$, $3 \notin A$, etc.
(2) We define the **length** of a string σ, denoted $|\sigma|$, to be the number of entries in σ. For example, $|010101| = 6$.
(3) We define the concatenation of two strings σ and τ, denoted $\sigma \frown \tau$, to be the string σ followed by τ. For example, if $\sigma = 01010$ and $\tau = 11111$, $\sigma \frown \tau = 0101011111$.
(4) We say that τ is an **extension** of σ, denoted $\sigma \subseteq \tau$, if in every place in which σ is defined, τ has the same value, and $|\sigma| \leq |\tau|$. For example, 0100010 is an extention of 010.

Definition 4.2. The Use of a Computation
Let $\mathcal{A}$ be an expression whose computation involves a finite set of queries $n_0, ..., n_k$ to oracle A. We define the **use** of $\mathcal{A}$ to be the maximum of the set $\{n_i : 0 \leq i \leq k\}$. Intuitively, the use is the largest element of A one needs to compute $\mathcal{A}$. Since every computation is only finitely many steps long, there is always a use for an expression.

Finally, we prove that the Turing Degrees are not linearly ordered by the operation $\leq$:

Theorem 4.3. *The Turing Degrees are not linearly ordered - that is, there exist $C, E \in \mathcal{D}$ such that neither $C \leq E$ nor $E \leq C$.*

Proof. In order to show that there exist $C, E \in \mathcal{D}$ such that neither $C \leq E$ nor $E \leq C$, we will construct two sets, $A, B \subset \mathbb{N}$, such that A is not B-computable and B is not A-computable. Given these sets A and B, let $C = \deg(A)$ and $E = \deg(B)$. Next, assume, without loss of generality, that $C \leq E$. Then, as illustrated in Definition 3.11.4, $A \leq_T B$, which would be a contradiction. Therefore, neither $C \leq E$ nor $E \leq C$. So, if we can construct A and B, then the theorem is proven.
We construct $A, B \subset \mathbb{N}$ such that A is not B-computable and B is not A-computable. Recall that there are countably many A-computable sets and countably many B-computable sets. So, to ensure that A is not B-computable and B is not A computable, we will satisfy the following relations:

$$R_{2i} : \chi_A \neq \phi_i^B$$
$$R_{2i+1} : \chi_B \neq \phi_i^A$$

With these relations satisfied, A cannot be B-computable because A is not any of the B-computable sets. Similarly, B cannot be A-computable.
To construct A and B, we shall use sequences of strings $\{\sigma_i\}_{i\in\mathbb{N}}$ and $\{\tau_i\}_{i\in\mathbb{N}}$, where $\sigma_0 \subset \sigma_1 \subset \sigma_2 \subset \ldots$ and $\tau_0 \subset \tau_1 \subset \tau_2 \subset \ldots$
Then, we shall define the characteristic functions of A and B by:

$$\chi_A = \bigcup_{i\in\mathbb{N}} \sigma_i$$

$$\chi_B = \bigcup_{i \in \mathbb{N}} \tau_i$$

In order to satisfy the relations, at stage $i+1$, we will define σ_{i+1} and τ_{i+1} in such a way as to ensure that the i-th requirement is satisfied.

The Construction

Stage 0:
Define $\sigma_0 = \tau_0 = \emptyset$.

Stage $i + 1 = 2e + 1$:
We specifically handle R_{2e} and leave R_{2e+1} to the reader as the process is exactly the same.
Assume $\sigma_0 \subset \sigma_1 \subset \sigma_2 \subset \ldots \subset \sigma_i$ and $\tau_0 \subset \tau_1 \subset \tau_2 \subset \ldots \tau_i$ are already defined.
Let $x_i = |\sigma_i|$
We look for a string $\tau \supset \tau_i$ such that $\phi_e^\tau(x_i)$ halts.

CASE I: If τ exists
Then computably choose a τ and let $k = \phi_e^\tau(x_i)$. Then define:

$$\sigma_{i+1} = \sigma_i \frown (1 - k)$$

$$\tau_{i+1} = \tau$$

We show that $\phi_e^B(x_i) = \phi_e^\tau(x_i)$ by contradiction.
Assume $\phi_e^B(x_i) \neq \phi_e^\tau(x_i)$. Then, since τ is the beginning of the characteristic function of B, τ and B agree up to $|\tau|$. Therefore, there must exist $k \in \mathbb{N}$ such that $k > |\tau|$ and the k-th value of B changes the output of $\phi_e^B(x_i)$ so that it is not equal to $\phi_e^\tau(x_i)$. Then, $\phi_e^\tau(x_i)$ must query the k-th value of τ as well. However, as τ is not defined at this value, $\phi_e^\tau(x_i)$ could not halt. This would be a contradiction of our initial assumption, which states that $\phi_e^\tau(x_i)$ halts.
Therefore, $\phi_e^B(x_i) = \phi_e^\tau(x_i)$, and so $\phi_e^B(x_i) = \phi_e^\tau(x_i) = k \neq (1 - k) = \chi_A(x_i)$. The characteristic functions of ϕ_e^B and A disagree on input x_i and thus A cannot be the i-th B-computable set. Therefore, the relation R_{2e} is satisfied.

CASE II: If τ does not exist
Then define

$$\sigma_{i+1} = \sigma_i \frown 0$$

$$\tau_{i+1} = \tau_i \frown 0$$

I claim that, regardless of how τ_n, $n > i$, are picked, $\phi_e^B(x_i)$ will not halt. Given that $\phi_e^B(x_i)$ does not halt, our relation is satisfied because $\phi_A(x_i) = 0$ but $\phi_e^B(x_i)$ is undefined. So A is not the i-th B-computable set, as the characteristic functions disagree at input x_i. Thus, the relation R_{2e} is satisfied.

Proof of Claim
We prove this claim by contradiction. Assume that $\phi_e^B(x_i)$ halts.
Let w be greater than $|\tau_i|$ and the use of $\phi_e^B(x_i)$. Then, for τ equal to the first w elements of B, $\phi_e^B(x_i) = \phi_e^\tau(x_i)$ because for every element of B used to compute $\phi_e^B(x_i)$, τ has the exact same elements. Note also that $\tau \supset \tau_i$. We have found

a τ such that $\phi_e^\tau(x_i)$ halts, so we cannot be in Case II. Therefore, we arrive at a contradiction. By this contradiction, $\phi_e^B(x_i)$ cannot halt. □

Corollary 4.4. *In fact, we can find $A, B \subset \mathbb{N}$ such that $A, B \leq_T K_0$ and A is not B-computable and B is not A-computable. So, not even the computably enumerable sets are linearly ordered.*

Proof. We need to verify that the construction in the above theorem can be carried out computably using an oracle V which is K_0-computable (because then $A, B \leq_T V \leq_T K_0$).
Note that the only part of the above construction which is not computable is in determining whether or not there exists τ such that τ extends τ_i and $\phi_e^\tau(x_i)$ halts. Therefore, we need to be able to tell V-computably whether or not τ exists.
Note that τ exists if and only if there exists a pair $\langle s, \tau\rangle$ such that $\tau \supset \tau_i$ and $\phi_e^\tau(x_i)$ halts in s steps. Define

$$V = \{\langle \pi, \pi'\rangle : (\exists \langle s, \tau\rangle)[\tau \supset \pi \text{ and } \phi_e^\tau(|\pi'|) \text{ halts in } s \text{ steps}]\}$$

Then, τ exists at stage $e+1$ if and only if $\langle \tau_e, \sigma_e\rangle \in V$
Note that V is an existential set, so V is equal to some W_e, $e \in \mathbb{N}$, as illustrated by Theorem 2.7. Then, by Example 3.4, $W_e \leq_T K_0$, so $V \leq_T K_0$. Therefore, $A, B \leq_T K_0$. □

Conclusion

The above theorem and associated corollary give us a very interesting result: there exist sets which are not Turing comparable. Note that neither of the two sets A and B generated are computable (if, say A, was computable, then by Example 3.5, $A \leq_T B$, which is a contradiction). This means that both A and B are sets which hold some computing information. Since each A and B hold computing information, but A is not B-computable and B is not A-computable, both A and B have some computing information which the other does not. Thus, there must be at least two different kinds of this computing information: a set does not have just an amount of computing information but also a kind. The above corollary is surprising because it shows that there are different kinds of information even in the computably enumerable sets. Thus, this division of information is immediate.

Acknowledgments. It is my pleasure to thank my mentors, Damir Dzhafarov and Eric Astor, for their help in my studies. They guided me towards useful books, helped me write this paper, and weathered my questions which were both frequent and at times more obscure than the concept I had questions about.

References

[1] Barry Cooper. Computability Theory. Chapman and Hall/CRC. 2004.
[2] Michael Sipser. Introduction to the Theory of Computation. PWS Publishing Company. 1997.