Definition 1. A **non-deterministic** TM is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ with $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$, $\delta(q, \gamma) \subset Q \times \Gamma \times \{L, R\}$ (δ is a finite subset) i.e., it can simultaneously branch in different ways

0	1	$\gamma = 1$	...
	q	$\hat{Q}$	

Theorem 1. *Any non-deterministic TM (NDTM) has an equivalent TM*

Proof. Given a NDTM N, we want to design a TM M which accepts precisely the same set of strings as N.
Idea: At each step, iterate over the possibilities given by $\delta(q, \gamma)$.
Design M to have three tapes

1 : Input (never change)	0	1	0	...	1	␣	...
	↓	↓	↓	↓	↓		
2 : Simulation	0	1	0	...	1	␣	...
3 : Keep Track of Branching				...			...

Algorithm

1. Copy input onto tape 2
2. Set $n = 1$.
3. For each setring $a_{i_1}, a_{i_2}, \ldots, a_{i_n}$ in $Q \times \Gamma \times \{L, R\}$ of length n, Simulate the branch of N given by $a_{i_1}, a_{i_2}, \ldots, a_{i_n}$ on tape 2. Accept in N accepts.
4. $n = n + 1$ and repeat 3.

This explores all brances to level one of the tree, then all branches to level two, etc. □

Corollary 1. *A language $A \subset \Sigma*$ is recognized by a NDTM iff A is recognized by a TM.*

Complexity of Algorithms
How can we measure complexity? Introduce Big-O Notation.
Example 1. *Let*

$$NOTPRIME = \{\textit{integers } n \geq 0 \textit{ that are not prime}\}$$

and use our TM defined by this algorithm

0. *Let n be input*
1. *$i = 2$*
2. *Ff i divides n, accept. If $i = n$ reject. Otherwise $i = i + 1$ and repeat step 2.*

The number of steps this takes is around n, or around 2^ℓ where ℓ = length of input in binary (ignoring constants) Now take our NDTM

0. *Let n be the input*
1. *$i = 2$*
2. *Simultaneously check if $2, \ldots, 2^i$ divide n. If yes, accept. If $i = n$ reject. Otherwise $i = i + 1$ and repeat step 2.*

This takes around $\log_2 n$ steps (or around ℓ)

Definition 2. Let f, g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n)$ is $O(g(n))$ if there exists some constant C such that $f(n) \leq Cg(n)$ for all n.
Example 2.

1. $3n + 6$ *is* $O(n)$ *because* $3(n) = 6 \leq 9n$.
2. $7n^3 + 8n + 19$ *is* $O(n^3)$.
3. $2^n + n^{2019}$ *is* $O(2^n)$.

Definition 3. The **running time** of a TM M is

$$f(\ell) = \text{max } \# \text{ of steps } M \text{ takes on an input of length } \ell$$

Definition 4. Let $\mathbf{P} = \{A \subset \{0, 1, \} * | \exists$ a TM M which recognizes A with run time $O(n^k)$ for some $k\}$ (in the space of binary strings). In other words, the running time $f(n)$ of M is bounded by a polynomial
Definition 5. Let $\mathbf{NP} = \{A \subset \{0, 1, \}* | \exists$ a NDTM N which recognizes A with run time $O(n^k)$ for some $k\}$ (in the space of binary strings).
Theorem 2. $P \subset NP$
Conjecture 1. $P = NP$

This is the central question of the course.