

Time Complexity

- In computability theory, we were interested in the question: “which problems are solvable?”. In (time) complexity theory, the question is: “which problems are solvable within some time bound?”.
- **Definition:** If M is a Turing machine and $w \in \Sigma^*$, $t_M(w)$ is the number of steps taken by M on input w (if M doesn't halt, then $t_M(w) = \infty$), and for $n \in \mathbb{N}$,

$$T_M(n) = \max\{t_M(w) \mid |w| = n\}$$

$T_M(n)$ is the worst-case time complexity of M on inputs of size n .

- **Definition:** For a function $f : \mathbb{N} \mapsto \mathbb{N}$,

$$\text{DTIME}(f) = \{L \mid \exists \text{ a TM } M \text{ with } L(M) = L \text{ s.t. } T_M(n) \in O(f(n))\}$$

is the set of languages decidable in time $O(f)$.

- Since, after all, we are not so interested in Turing machines as in “algorithms”, it is important to ask how the time complexity of a Turing machine relates to our “real-world” notion of complexity.
- In the case of computability we had the Church-Turing thesis, which states that Turing machines are equivalent to our intuitive notion of “algorithms”.
- In complexity, it is not so clear. Consider the language $L = \{0^n 1^n \mid n \in \mathbb{N}\}$. A typical algorithm for this problem would move through the input once, counting the number of 0's and the number of 1's, and compare the two counters at the end. The time for this algorithm is $O(n \log n)$ (the $\log n$ factor comes from having to increment a counter that holds about $\log n$ bits). However, a typical solution on a one-tape TM would require $\Omega(n^2)$. This $O(n \log n)$ solution can also be implemented on a two-tape TM. So if we consider such small classes as “linear time” or “quadratic time” then these models are not equivalent. On the other hand, the differences are not large in the big scheme of things: in the preceding example the complexity of a one-tape TM differed by less than a linear factor. This motivates the following definition.
- **Definition: (Polynomial-time)**

$$\mathbf{P} = \bigcup_{k>0} \text{DTIME}(n^k)$$

- **Extended Church-Turing thesis:** Any language that is “efficiently decidable” by an “algorithm” is decidable by a Turing machine in polynomial time.

- The extended thesis is not as widely believed as the original thesis; for example, it is cast into doubt by developments in quantum computing. However, no counter-examples are known, and it does appear to hold for traditional notions of computation.

Examples of languages in P

- **Graph connectivity.** Given a directed graph $G = (V, E)$ and two vertices $s, t \in V$, decide whether there exists a directed path from s to t in G .

$$\text{STCON} = \{\langle G, s, t \rangle \mid G = (V, E), s, t \in V, \text{ there is a path from } s \text{ to } t \text{ in } G\}$$

STCON can be solved in linear time by DFS or BFS on a computer. In the Turing machine model the following algorithm is easier: assume the input graph G is represented as a list of edges (u, v) .

1. Make a list of all the vertices v_1, v_2, \dots, v_n
2. Mark s in the list (with some special symbol)
3. While $\exists(u, v) \in E$ such that u is marked but not v , mark v
4. Accept iff t is marked

As an exercise, try and determine the running time of this algorithm on a one-tape TM.

- **Shortest-Path.** Given a directed graph $G = (V, E)$, two vertices $s, t \in V$, and a number $d \in \mathbb{N}$, determine whether the length $d_G(s, t)$ of the shortest path from s to t is equal to d .

$$\text{SHORTEST-PATH} = \{\langle G, s, t, d \rangle \mid G = (V, E), s, t \in V, d_G(s, t) = d\}$$

- **Maximum matching.** A matching in a graph $G = (V, E)$ is a subset $M \subseteq E$ such that for every $v \in V$, there is at most one edge in M that contains v (it is called a matching because it pairs up vertices, but some vertices may not have a partner).

$$\text{MAX-MATCHING} = \{\langle G, t \rangle \mid G \text{ has a matching of size at least } t\}$$

- **Primality.** Given a number $n \in \mathbb{N}$, decide whether n is prime.

$$\text{PRIMES} = \{\langle n \rangle \mid n \in \mathbb{N} \text{ is prime}\}$$

An important result by Agrawal, Kayal and Saxena (2002): $\text{PRIMES} \in \mathbf{P}$. It's important to understand why the "obvious" algorithm does not work. The obvious algorithm is: for $i = 2, \dots, \sqrt{n}$ check if i divides n . If an i is found that divides n then reject; if no such i is found then accept. This algorithm requires time $\Omega(\sqrt{n})$. But what is the input size? The input is

a number $n \in \mathbb{N}$, which if written in binary requires only $O(\log n)$ bits. So if $N = \log n$ is the input size,

$$\begin{aligned}\sqrt{n} &= n^{1/2} \\ &= (2^{\log n})^{1/2} \\ &= 2^{N/2}\end{aligned}$$

i.e. the running time is exponential in the size of the input.

Non-determinism

- A non-deterministic Turing machine (NTM) is like an ordinary TM, except it has a transition function

$$\delta : Q \times \Gamma \mapsto \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

where $\mathcal{P}()$ denotes the power set (set of all subsets).

- An NTM can have several “next” configurations: e.g. $AaqbB \vdash_M Aab'q'B$ if $(q', b', R) \in \delta(q, b)$. If $(q'', b'', L) \in \delta(q, b)$ then $AaqbB \vdash_M Aq''ab''B$ is also true.
- A computation of M on input w is a sequence C_1, C_2, \dots of IDs, such that
 1. $I_M(w) = C_1$
 2. $C_i \vdash_M C_{i+1}$, for $i = 1, 2, \dots$
 3. If the sequence ends with some C_k , then either C_k is accepting or C_k is rejecting.

Note that there can be many possible computations of M on w (if you draw them, they look like a tree)

- An NTM M accepts $w \in \Sigma^*$ if there *exists* an accepting computation of M on w
- **Definition:** The time $t_M(w)$ required by an NTM M on input w is the maximum length of any computation of M on w . For $n \in \mathbb{N}$,

$$T_M(n) = \max\{t_M(w) \mid |w| = n\}$$

- **Definition:** For a function $f : \mathbb{N} \mapsto \mathbb{N}$,

$$\text{NTIME}(f) = \{L \mid \exists \text{ an NTM } M \text{ with } L(M) = L, \text{ s.t. } T_M(n) \in O(f(n))\}$$
- **Definition (Non-deterministic polynomial time):**

$$\text{NP} = \bigcup_{k>0} \text{NTIME}(n^k)$$

The Big Question

- The big question: is $\mathbf{P} = \mathbf{NP}$? The answer to this question is not known.
- Perhaps a more relevant question: why does anyone care? The answer is that there are a large number (literally thousands) of very important real-world problems that are known to be in \mathbf{NP} , but are not known to be in \mathbf{P} (i.e. no deterministic polynomial-time algorithm is known for these problems). If $\mathbf{P} = \mathbf{NP}$ then it is possible to obtain “fast” algorithms to solve these problems. If $\mathbf{P} \neq \mathbf{NP}$ then at least we would know that it is not a failure on the part of algorithm designers that is preventing a solution to these problems, but some kind of inherent “hardness”.
- Here is an example of a language that is in \mathbf{NP} , but is not known to be in \mathbf{P} :

$$\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S \subseteq \mathbb{N} \text{ is finite,} \\ \exists S' \subseteq S \text{ s.t. } \sum_{n \in S'} n = t \}$$

- Here is a non-deterministic algorithm for SUBSET-SUM:
 1. Guess a subset $S' \subseteq S$
 2. Compute the sum of the elements in S'
 3. If the sum is equal to t then accept, else reject

The non-determinism appear in the first step. By “guess a subset”, we mean the following: the Turing machine contains a transition which allows it to write a 0 and move right, and another transition which allows it to write a 1 and move right. It repeats this $|S|$ times, so that at the end it has a string $b_1, \dots, b_{|S|}$ of bits. This string corresponds to a subset of S (if $b_i = 1$ then the i -th element is included in the subset, otherwise the i -th element is not included).

- If $\langle S, t \rangle \in \text{SUBSET-SUM}$ then M has an accepting computation (the one where it correctly guesses the set S'). On the other hand, if $\langle S, t \rangle \notin \text{SUBSET-SUM}$ then no matter what subset M guesses, it will reject.
- The running time of the algorithm as written is $O(|S| \log N)$, where N is the largest number in S . As the input size n satisfies $n \geq |S|$ and $n \geq \log N$ (since every element requires at least one bit in the input, and the largest element requires at least $\log N$ bits), this is time $O(n^2)$.