Time complexity

- Here we will consider *elements of computational complexity theory* – an investigation of the time (or other resources) required for solving computational problems.

- We introduce a way of measuring the time used to solve a problem. Then we will classify problems according to the amount of time required.

- We will see that certain decidable problems require enormous amounts of time and how to determine when you are faced with such a problem.

- Let consider an example of a TM $M_1$ which decides the language $A = \{0^k1^k : k \geq 0\}$.

\[ M_1 = \text{``on input } w:\]

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat the following if both 0s and 1s remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*.”

- How much time does a single type TM need to decide $A$?

- We count the number of steps that algorithm uses on a particular input as a function of the length of the string representing the input.

- We consider *worst case analysis*, i.e., the longest running time of all inputs of a particular length.
Asymptotic notation: big-$O$ and small-$o$

• **Def. 1:** Let $M$ be a TM that halts on all inputs. The **running time** or **time complexity** of $M$ is the function $f: \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$. We say $M$ runs in time $f(n)$ and $M$ is an $f(n)$ time Turing machine.

• **Def. 2:** Let $f$ and $g$ be two functions $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$. Say that $f(n) = O(g(n))$ if positive integers $c$ and $n'$ exist so that for every $n \geq n'$, $f(n) \leq c \cdot g(n)$. We say that $g(n)$ is an **upper bound** for $f(n)$ (or **asymptotic upper bound**).

  Intuitively, this means that $f$ is less than or equal to $g$ for sufficient large $n$ if we disregard differences up to a constant factor. $O$ represents that constant; constant is hidden under $O$.

  \[
  \text{If } f(n) = 5n^3 + 2n^2 + 22n + 6, \text{ then } f(n) = O(n^3) \text{ or } f(n) = O(n^4), \text{ but } f(n) \neq O(n^2). \\
  \text{If } f(n) = 3n \log_2 n + 5n \log_2 \log_2 n + 2, \text{ then } f(n) = O(n \log n). \\
  \text{If } f(n) = O(n^2) + O(n), \text{ then } f(n) = O(n^2). \\
  \]

• Other examples of run-time: $2^{O(n)}$, $O(1)$, $n^{O(1)} (= 2^{O(\log n)})$. Bounds of the form $n^c$ for $c > 0$ are called **polynomial bounds**. Bounds of the form $2^{(n^\delta)}$ ($\delta > 0$) are called **exponential bounds**.

• **Def. 3:** Let $f$ and $g$ be two functions $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$. Say that $f(n) = o(g(n))$ if for any real $c > 0$, a number $n'$ exists so that for every $n \geq n'$, $f(n) < c \cdot g(n)$.

  • Examples: $\sqrt{n} = o(n)$, $n = o(n \log \log n)$, $n \log \log n = o(n \log n)$, $n \log n = o(n^2)$, $n^2 = o(n^3)$, but $f(n) \neq o(f(n))$.

  \[
  \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.
  \]
Analyzing Algorithms

Let’s analyze the algorithm we gave for the language \( A = \{0^k1^k : k \geq 0\} \).

\[ M_1 = \text{on input } w:\]

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Repeat the following if both 0s and 1s remain on the tape.
3. Scan across the tape, crossing off a single 0 and a single 1.
4. If 0s still remain after all the 1s have been crossed off, or if 1s remain after all the 0s have been crossed off, reject. Otherwise, if neither 0s nor 1s remain on the tape, accept.”

- **Stage 1:** verifies that input is of form 0*1* in 2n steps. Hence \( O(n) \) steps.
- **Stages 2,3:** each scan uses \( O(n) \) steps, at most \( n/2 \) scans. Hence \( O(n^2) \) steps.
- **Stage 4:** at most \( O(n) \) steps.
- Hence, the total time of \( M_1 \) on input of length \( n \) is \( O(n) + O(n^2) + O(n) = O(n^2) \).

- **Def. 4:** Let \( t : \mathbb{N} \rightarrow \mathbb{N} \) be a function. Define the time complexity class, \( TIME(t(n)) \), to be

\[ TIME(t(n)) = \{L : L \text{ is a language decided by an } O(t(n)) \text{ time Turing machine}\}. \]

- We have \( A \in TIME(n^2) \). Is there a machine that decides \( A \) asymptotically more quickly?
\[ A = \{0^k1^k : k \geq 0\} \in TIME(n \log n) \]

**Why does \( M_2 \) decide \( A \)?**

- on every scan performed in stage 4, the total number of 0s (of 1s) remaining is cut in half and any remainder is discarded.
- in stage 3 we check whether the parities of \# of 0s and \# of 1s are the same.

**Running Time:**

- **All Stages** take \( O(n) \) steps.
- **Stages 1 and 5** are executed once.
- **Stages 2,3,4** are executed at most \((1+\log n)\) time.

Hence, the total time of \( M_2 \) on input of length \( n \) is

\[ O(n) + (1+\log n)(O(n) + O(n) + O(n)) + O(n) = O(n \log n). \]

So, \( A \in TIME(n \log n) \). This result cannot be further improved on a single tape TM.
Linear time two-tape Turing machine for $A$. 

$M3 = \text{“on input } w:\n$ 

1. Scan across the tape and reject if a 0 is found to the right of a 1.
2. Scan across the 0s on tape 1 until the first 1. At the same time, copy the 0s onto tape 2.
3. Scan across the 1s on tape 1 until the end of the input. For each 1 read on tape 1, cross off a 0 on tape 2. If all 0s are crossed off before all the 1s are read, reject.
4. If all the 0s have now been crossed off, accept. If any 0s remain, reject.”

- Clearly, this is a decider for $A$. Running time is clearly $O(n)$.

**Summary:**
- We presented a single tape TM $M2$ that decides $A$ in $O(n \log n)$ time.
- We mentioned (w/o proof) that no single tape TM can do it more quickly.
- Then we presented a two-tape TM $M3$ that decides $A$ in linear time.
- Hence, the complexity of $A$ depends on the model of computation selected.

- This shows an important difference between *complexity theory* and *computability theory*.

- In *computability theory*, The Church-Turing thesis implies that all reasonable models of computation are equivalent, i.e., they decide the same class of languages. In *complexity theory*, the choice of model affects the time complexity of languages.
Complexity relations among models: Multi-tape TM

\[ \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}^k, \]
\[ \delta(q, a_1, ..., a_k) = (r, b_1, ..., b_k, L, R, ..., L) \]

Theorem 1: Let \( t(n) \) be a function, where \( t(n) \geq n \). Then every \( t(n) \) time multi-tape TM has an equivalent \( O(t^2(n)) \) time single-tape TM.

- We have seen before how to convert a multi-tape TM \( M \) to an equivalent single tape TM \( S \), that simulates it.

- Let \( M \) be a \( k \)-tape TM that runs in \( t(n) \) time. We will show that simulating each step of the multi-tape TM uses at most \( O(t(n)) \) steps of the single-tape TM. Hence the total time used is \( O(t^2(n)) \).

- \( S \) simulates the effect of \( k \) tapes by storing their information on its single tape.

- It uses new symbol \( \# \) as a delimiter to separate the contents of the different tapes.

- \( S \) must also keep track of the locations of the heads. It does so by writing a tape symbol with a dot above it to mark the place where the head on that tape would be.
Multi-tape TM vs. Single-tape TM

S=“On input $w = w_1 w_2 ... w_n$ :

1. First $S$ puts its tape into the format that represents all $k$ tapes of $M$. The formatted tape contains

$$\# w_1 w_2 ... w_n \# \# \# \# \# \# ... \#$$

2. To simulate a single move, $S$ scans its tape from the first $\#$, which marks the left-hand end, to the $(k+1)$st $\#$, which marks the right-hand end, in order to determine the symbols under the virtual heads. Then $S$ makes a second pass to update the tapes according to the way that $M$’s transition function dictates.

3. If at any point $S$ moves one of the virtual heads to the right onto a $\#$, this action signifies that $M$ has moved the corresponding head onto the previously unread blank portion of that tape. So $S$ writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost $\#$, one unit to the right. Then it continues the simulation as before.

Running Time:

- **Stage 1** takes $O(n)$ steps and is executed once.
- **Stages 2, 3:** $S$ simulates each of the $t(n)$ steps of $M$, using $O(t(n))$ steps.

- The length of the active portion of $S$’s tape determines how long $S$ takes to scan it.
- A scan of the active portion of $S$’s tape uses $O(t(n))$ steps. (Why???)

- Hence, the total time of $S$ on input of length $n$ is

$$O(n) + t(n) \times O(t(n)) = O(t^2(n)).$$
Complexity relations among models: Non-deterministic TM

• A non-deterministic TM is a decider if all its computation branches halt on all inputs.

• **Def 5:** Let $N$ be a non-deterministic TM that is a decider. The **running time** of $N$ is the function $f: N \rightarrow N$, where $f(n)$ is the maximum number of steps that $N$ uses on any branch of its computation on any input of length $n$.

\[
\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}
\]

\[
\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})
\]

**Theorem 2:** Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time non-deterministic TM has an equivalent $2^{O(t(n))}$ time deterministic TM.

• We have seen before that any non-deterministic TM $N$ has an equivalent deterministic TM $D$, that simulates it.
Non-deterministic TMs vs. ordinary TMs

• The simulating deterministic TM $D$ has three tapes.
  • Tape 1 always contains the input string and is never altered.
  • Tape 2 maintains a copy of $N$’s tape on some branch of its non-deterministic computation.
  • Tape 3 keeps track of $D$’s location in $N$’s non-deterministic computation tree.

• Every node in the tree can have at most $b$ children, where $b$ is the size of the largest set of possible choices given by $N$’s transition function.

• Tape 3 contains a string over $\Sigma_b = \{1,2,\ldots,b\}^*$. Each symbol in the string tells us which choice to make next when simulating a step in one branch in $N$’s non-deterministic computation. This gives the address of a node in the tree.

• On an input of length $n$, every branch of $N$’s non-deterministic computation tree has a length of at most $t(n)$. Hence the total number of leaves in the tree is at most $b^{t(n)}$.

• The total number of nodes in the tree is less that twice the maximum number of leaves, i.e. is bounded by $O(t(n))$. Hence the running time of $D$ is $O(t(n)b^{t(n)}) = 2^{O(t(n))}$.

• $D$ has three tapes. Converting it to a single tape TM $S$ at most squares the running time.

• So, the running time of $S$ is $(2^{O(t(n))})^2 = 2^{O(2t(n))} = 2^{O(t(n))}$. 