# Computational Complexity

Frank

March 11, 2022

## 1 Introduction

**Computability**: What can be computed.

**Complexity**: What can be computed efficiently

- time (focus of this course)
- space
- others

From now on, only consider algorithms that halt on all inputs

ex. deciders

## 2 Running time of TMs

### 2.1 Definition

**Def**: Running time of TM is a function $T : \mathbb{N} \to \mathbb{N}$, (input size $\to$ number of steps) s.t. $T(n) =$ max number of steps the TM takes on inputs of size $n$ (worst-case)

Use $O, \Omega, \Theta$

Note that here, our size of input is the most simple case; the length of the string representing the input.

### 2.2 An example

Suppose we have language $L = \{0^k 1^k : k \in \mathbb{N}\}$

The process we will take is as follows:

1. Scan left to right and see if input is the right form
2. Move back to the left, and mark the first 0, scan to the right and see if there is a 1, and mark that too
3. mark a 1 and scan to the left and check for a 0
4. etc...

The running time of the algorithm on input of length $n$ is approximately $T(n) = \Theta(n^2)$

Have approx. $\frac{n}{2}$ sweeps, and on each sweep, we have around $\frac{n}{2}$ cells to sweep through

Here is a more efficient process:

1. Scan left to right and see if input is the right form

2. Mark every other 0's or 1's

The running time will now be

$$\text{Number of sweeps} = O(\log n)$$
$$\implies T_2(n) = O(n \log n)$$

**Fact**: No Single tape TM can decide $L$ in less than $c \cdot n \log n$ time - $o(n \log n)$

**small o**: $f(n) = o(g(n)) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (strictly less than)

2-tape TM for $L = \{0^k 1^k : k \in \mathbb{N}\}$ can be solved efficiently in $O(n)$

The idea is to first sweep and check if the string is in the correct form

Then we want to copy the 0's onto the second tape, then verify that we indeed have the same number of 1's and 0's by moving in opposite directions from the "center"

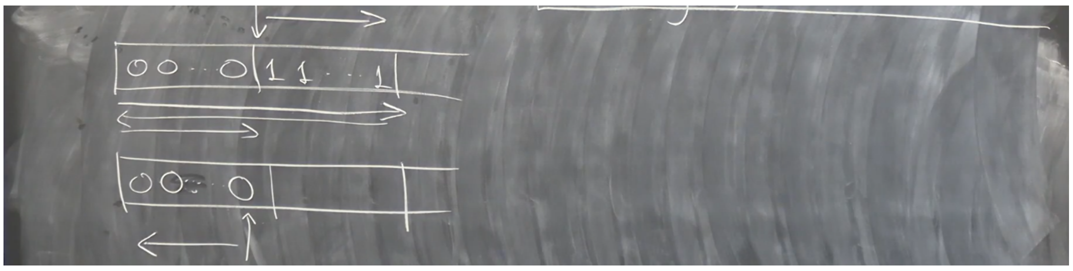


Figure 1: l11

## 2.3 Subtle Problem

Different models of computation have different worst-case times.

For example, 1-tape TM and 2-tape TM have different worst-case times to decide $L$

$\implies$ We need to be able to talk about complexity in an abstract manner (without concern about the underlying model of computation)

We will "lump" things together. For example efficient $\implies$ runs in polynomial time (classes)

## 2.4 Polynomial Time Thesis (polytime) thesis

Attempts to disconnect complexity with the models of computation

A problem is solvable efficiently

$\implies$ $\exists$ a TM that solves it in $O(n^k)$ time, for some constant $k$

Note that we require "bounded from above"

ex. something like $O(n \log n)$ is also polytime

When we say non-polytime, we might think of exponential time - $O(b^n)$, but this isn't entirely true

ex. We can have things like $2^n, 2^{2^n}, 1.0001^n, n^{\log_2 n}$

Note that polynomials are closed under composition:

> $f(x) = x^2$
>
> $g(x) = x^3$
>
> $f(g(x)) = x^6$
>
> This means when we encode some things, we can make sure that the encoding of the input is still polynomial in size.
>
> However, beware that this doesn't work with numbers. You can encode a number with $\log n$ bits, which means the actual proxy we are using might be exponentially larger than our encoding (not polynomial). An example will illustrate this fact below. (seemingly $O(n)$ algorithm is actually exponential in size of input)

Justification for this thesis (note that this thesis isn't as widely accepted as CT)

> **Theoretical**: Makes class of "efficient" algos independent of
>
> - details of computational model (likely exception: non-determinism)
>
>   Since, NTM relies on BFS through the computational tree, which is exponential in size of nodes. (However, we don't know if there is a more efficient way of searching through this tree)
> - details of reasonable encodings of input
>
>   Remember running time is a function of the size of input, so the encodings of the input matter in terms of giving us the size of input
>
>   For example, we can artificially "inflate" the input so that the running time isn't too much larger than the size of input. This is what we mean by "reasonable"; we do not "inflate" the size of input.
>
> **Empirical**:
>
> - Most practical problems with polytime algorithms have low degree polynomials
>
>   For example, $n^{100}$ is not very efficient, but it is better than $2^n$ for some $n \geq 1000$
> - A key observation is that the algorithms we have that searches the entire search space (brute force) is usually exponential - non-polytime
>
>   Insights into the problem will give us a better running time (skip portions of the search space), usually at first is a large polynomial, then refined to a small polynomial; efficient algorithm

## 3 TM vs. Practical Computer (RAC)

Constrast Turing machines against a random access computer (closer to our modern computer)

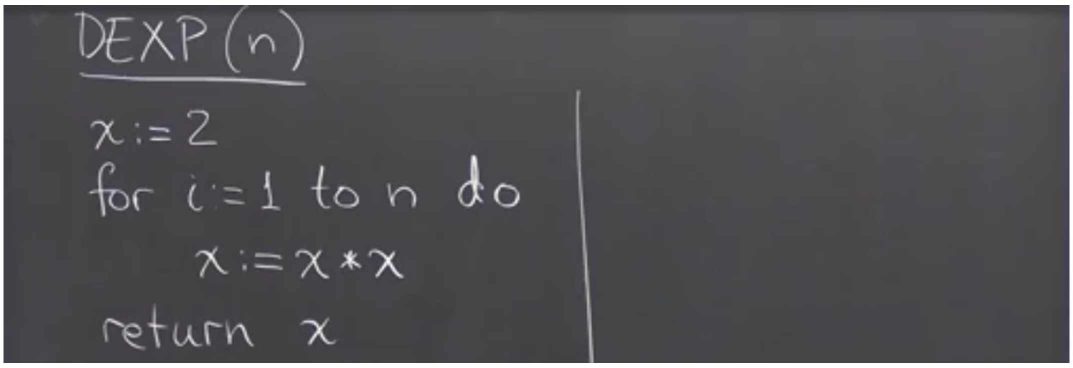
### 3.1 Differences

- Sequential vs. random access - $O(1)$

  To get from cell $a$ to cell $b$ in a TM, we need to sequentially go through all the cells in between the two cells. (this slows down computation by a polynomial amount)

- Sensitivity to size of the values being manipulated.

  For example:



```
DEXP(n)
x := 2
for i := 1 to n do
    x := x * x
return x
```

Figure 2: l12

Computes $2^{2^n}$

$T(n) = \Theta(n)$

At first glance, this may seem like a polytime algorithm, but it is not.

We need $\log_2 n$ bits to represent the input, while the steps we take is $O(n)$, that means the steps we take is exponentially bigger than our size of input

Even worse, consider DEXP(7), which means $2^{128}$. The numbers we multiply together are going to get very very big, which means the actual cost is even worse than exponential. Let's analyze it.

> iteration $i$ - number of bits for $x = 2^{i-1}$ (doubles)
>
> Now, multiplying $m$ bits together takes $O(m^2)$. $\implies 2^{2(i-1)} = 4^{i-1}$ steps
>
> Now, we do this operation $n$ times, giving us $\sum_{i=1}^{n} 4^{i-1} = \sum_{i=0}^{n-1} 4^i = \frac{4^n - 1}{3} = O(4^n)$

We were not taking into account of the huge numbers we are multiplying in our naive approach

If the size of input is polynomial, then using our naive approach would correctly identify that the algorithm is indeed polytime

## 3.2 Informal idea of RAC

We want to see that if we can compute a problem in polytime on a RAC, then we can indeed do the same (perhaps with a higher degree) on a TM.

A RAC looks like the following:

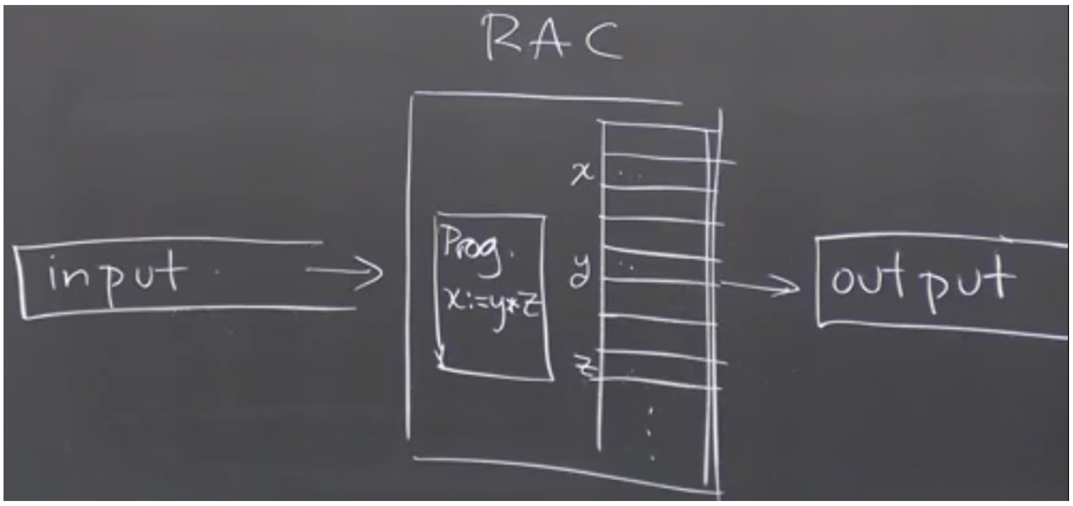


Figure 3: l13

We have a device where we can load programs, inputs. Our device computes using a RAM, which is constant in access time, and delivers output.

We have basic operations such as: +, -, x, /, ... that when applied to $m$-bit operands, takes $O(m^c)$ bit operations (polytime)

**Fact 7.1**: If a computation performed by RAC on input of size $n$ requires a polynomial (in $n$) number of

- memory locations, say $O(n^a)$
- basic operations support by the RAC, say $O(n^b)$

Then it can be performed by a TM in a polynomial (in $n$) number of its own steps

Proof sketch:

if the number of memory locations is $O(n^a)$ (size of input), then say we perform an operation that takes $O(n^c)$, then we have a combined number of steps per basic operation bounded by $O((n^a)^c) \cdot O(n^a) = O(n^{a(c+1)})$ (The additional $O(n^a)$ accounts for sequential access, or sweeping across the tape)

Then entire operation (bounded by $O(n^b)$) takes $O(n^{ab(c+1)})$ TM steps, which is still polynomial.

## 3.3 Input size

Consider bubblesort(), which takes an array of size $n$, but the size of input is actually (more accurately), $n \log n$, because each number needs $\log n$ bits to represent itself. However, note that the two input sizes are still a polynomial apart. So if we can compute a polytime algorithm for one, we can also compute a polytime algorithm for the other.

Now consider a graph $G = (V, E)$, and suppose the graph is represented in adj. list. We usually say that the size of input is $O(n+m)$, however, the more accurate representation is $O((n + m) \log n)$. Again, these representations are a polynomial off.

In practice, most numbers fit in a word of memory, so the "naive" way of measuring input size is actually quite accurate, but it is only when we want to consider cases where input size gets arbitrarily large that we would need to consider how the numbers are represented in the computer

**Summary** of the points:

- Polytime in pseudo-code according to usual conventions (naive) about counting steps and input size $\implies$ polytime in TM

  *caveat, beware of memory size (ex. algorithms that double in memory requirement every step, as we saw before, does not qualify as polytime)
- Note that designing a polytime algorithm does not indicate efficiency, we can always find ways to improve the algorithm run time on a particular machine

# 4 P Class (Decision Problems)

## 4.1 Definition

**Def**: $P$ is the class of languages/decision problem/bin-valued function computable by a TM (deterministic) in polytime - $O(n^k)$

## 4.2 Problems in class P

- SUM

  Instance: $\langle x, y, z \rangle$, $x, y, z \in \mathbb{N}$

  Question: $x = y + z$?

- PATH

  Instance: $\langle G, s, t \rangle$ w here $G = (V, E)$ is a graph, $s, t \in V$

  Question: Does there exist a $s \to t$ p th in $G$?

- WST

  Instance: $\langle G, wt, b \rangle$, where $G = (V, E)$, $wt : E \to \mathbb{N}$, $b \in \mathbb{N}$

  Question: Does there exist spanning tree of $G$ with $wt \leq b$? (Prim's)

- **Note** that in practice, we want **search problems**, which gives as output the actual object we want, instead of a yes/no answer like decision problems

  Later on, we will prove that the search problem in fact reduces to its decision problem (**self reducibility**)

## 4.3 Problems not in class P

Consider $EXP = \{\langle M, x \rangle : \ M \text{ accepts } x \text{ in } \leq 2^{|x|} \text{ steps } \}$

This problem is decidable, because we can just run $M$ on $x$ for $2^{|x|}$ steps and see if it accepts.

**Thm 7.2**: $EXP$ is not in $P$.

Another way of understanding this is that the best we can do is to run $M$ on $x$ for $2^{|x|}$ steps and see if it accepts, and this process is not polytime.

**Proof**:

Suppose for contra that $EXP \in P$

Define $EXP' = \{\langle M \rangle : \ M \text{ accepts } \langle M \rangle \text{ in } \leq 2^{|\langle M \rangle|} \text{ steps}\}$

$\implies EXP' \in P$ (Use polytime algo for $EXP$)

$\implies D = E\bar{X}P' = \{\langle M \rangle : \ M \text{ does not accept } \langle M \rangle \text{ in } \leq 2^{|\langle M \rangle|} \text{ steps}\}$

Clearly, $D \in P$ (just change yes answers to no answers for our polytime algorithm)

$\implies$ There exists polytime TM $M_D$ that decides $D$

Let $p(n)$ be polynomial running time of $M_D$

$\implies \exists n_0$ s.t. $\forall n \geq n_0$, $p(n) \leq 2^n$

WLOG assume $|\langle M_D \rangle| \geq n_0$ (we can just put junk in the encoding to fulfil this requirement)

- Question: What does $M_D$ do on $\langle M_D \rangle$?
  - Case 1: $M_D$ accepts $\langle M_D \rangle$

    $\implies$ it does so in $\leq 2^{|\langle M_D \rangle|}$ steps

    $\implies$ by definition of $D$, $M_D$ rejects $\langle M_D \rangle$
  - Case 2: $M_D$ rejects $\langle M_D \rangle$

    $\implies$ $M_D$ accepts $\langle M_D \rangle$ in $\leq 2^{|\langle M_D \rangle|}$ steps (by definition)

    $\implies$ $M_D$ accepts $\langle M_D \rangle$ by definition

  Then we have a contradiction

## 5 NP Class

**Def**: NP is class of decision problems computable by polytime NTMs.

Recall that a $P$ class problem can be thought of as a TM, starting at the initial config, and executing a sequence of configs until it reaches a halting config, where the length of that sequence is bounded by some polynomial.

A NTM computes a tree of configurations. If a NTM is a decider, then all configuration paths must halt. The running time of the NTM on a particular input is the height of its computation tree on that input or
the length of the longest computation path for that input.

**Def**: Running time of NTM $M$ is $T : \mathbb{N} \to \mathbb{N}$ s.t. $T(n) =$ max length of a computation path on inputs of size $n$

**Thm 7.3**: $P \subseteq NP$

TM is special case of NTM

**Conjecture 7.4**: $P \neq NP$ $[P \subset NP]$

Highly likely

We can simulate NTM with TM, however, this simulation is not polynomial (BFS of the computation tree).

### 5.1 NP Problems (NP Complete)

- Travelling Salesman - TSP

  Instance: $\langle G, wt, b \rangle$, $G$ is undirected graph, $wt : E \to \mathbb{N}$, $b \in \mathbb{N}$

  Question: There exists tour of $G$ of $wt \leq b$ (tour means cycle that visits each node exactly once)

- Independent (node) set

  Instance: $\langle G, b \rangle$, $G$ is undirected graph, $b$ is bound

  Question: Does there exist $S \subseteq V$ s.t. $|S| \geq b$ and $\forall u, v \in S$, $(u, v) \notin E$