Department of Computer Science

THEORY OF COMPUTATION
23MT2014

Topic:

TIME MEASURES OF COMPLEXITY

Session - 23

AIM OF THE SESSION

The aim of this course is to familiarize students with the complexity theory, measuring time and space complexity.

INSTRUCTIONAL OBJECTIVES

This Session is designed to:

1. Understand the fundamentals of complexity theory.
2. Explore the Asymptotic notation
3. Analyse the time complexity of a TM

LEARNING OUTCOMES

At the end of this session, you should be able to:

1. Understand the fundamentals of complexity in term of time
2. Explore the limits of algorithm solvability and decidability

COMPUTABILITY VS. COMPLEXITY

Are we guaranteed to get an answer?

Computability

How long do we have to wait for an answer?

Complexity: Time

How much resources do we need to find the answer?

Complexity: Space

TIME COMPLEXITY

- Let's begin with an example:
- Take the language $A = \{0^k 1^k \mid k \geq 0\}$
- Obviously, A is a decidable language but how much time does a **single-tape** Turing Machine need to decide A?

TIME COMPLEXITY

- We examine the following single-tape TM M_1 for A.
- M_1 = "On input string w:

```
1. Scan across the tape and reject if a 0 is found to the right of a 1
2. Repeat 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
       still remain after all the 0s have been crossed off, reject.
       Otherwise, if neither 0s nor 1s remain on the tape, accept."
```

TIME COMPLEXITY : DECIDING $\{0^K 1^K | K \geq 0\}$

- We examine the following single-tape TM M_1 for A.

- M_1 = "On input string w:

1. Scan left to right and confirm no 0's after 1's 2n steps
2. Repeat if both 0s and 1s remain on the tape: $\frac{n}{2}$ steps
3. Scan across the tape, crossing off a single 0 and a single 1. 4n step
4. If neither 0's or 1's remain on tape, accept; otherwise, reject (n steps)

Total steps: $2n + 4n \times \frac{n}{2} + n = 2n^2 + 3n \rightarrow O(n^2)$

CHARACTERIZING RUN-TIME

- If TM M halts on all inputs, there exists f: N→N
- f(n) = max. number of steps on any input of length n
  - M runs in time f(n)
  - M is an f(n) time Turing machine

TIME COMPLEXITY

- We analyze for TM M_1 deciding A how much time it uses.
- The number of steps that an algorithm takes on a particular input may depend on several parameters.
- For simplicity we compute the running time of an algorithm purely as a function of the length of the string.

WORST-CASE COMPLEXITY

- We consider the longest running time of all inputs of a particular length.

AVERAGE-CASE COMPLEXITY

- We consider the average of all running times of inputs of a particular length.

TIME COMPLEXITY

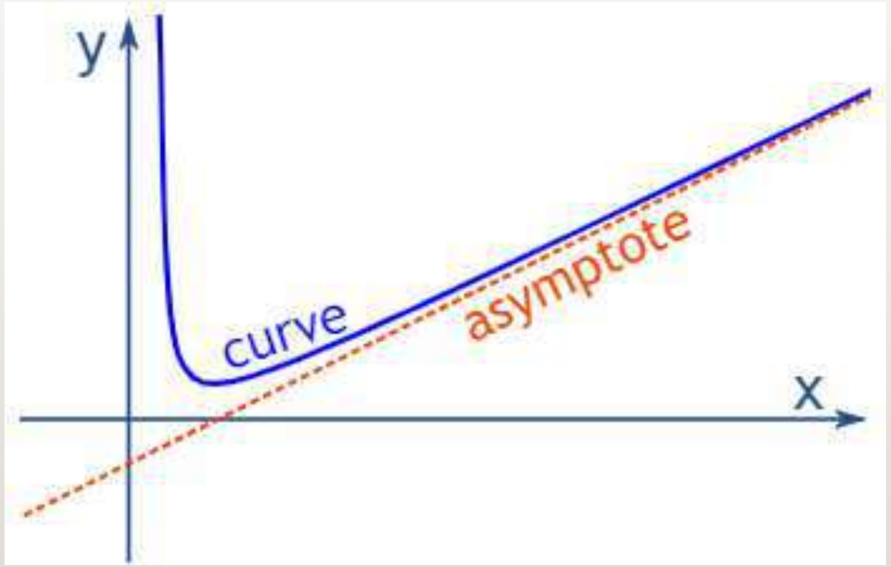
- Let M be a deterministic TM that halts on all inputs.
- The *time complexity* of M is the function $f: \mathbb{N} \rightarrow \mathbb{N}$, where $f(n)$ is the maximum number of steps that M uses on any input length n.

BIG-O AND SMALL-O NOTATION

- Because the exact running time of an algorithm often is a complex expression, we usually just estimate it.
- One convenient form of estimation, ***asymptotic analysis***, we seek to understand the running time of algo. when it is run on large inputs.

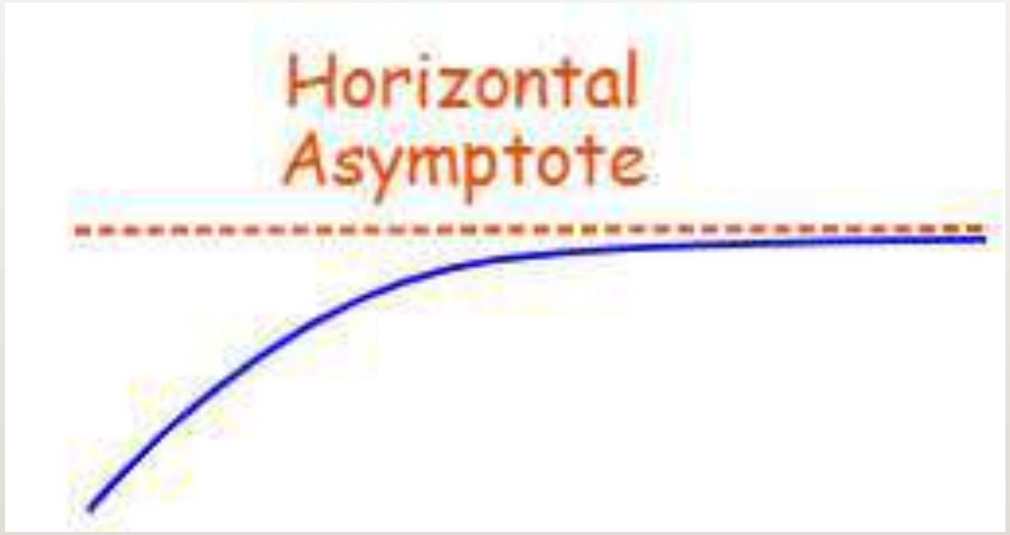
WHAT IS AN ASYMPTOTE?

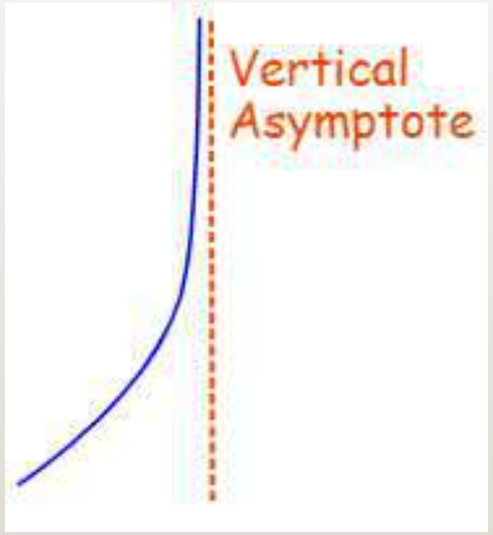
- An *asymptote* is a line that the curve of a function approaches, as either x-values or y-values head off towards infinity.

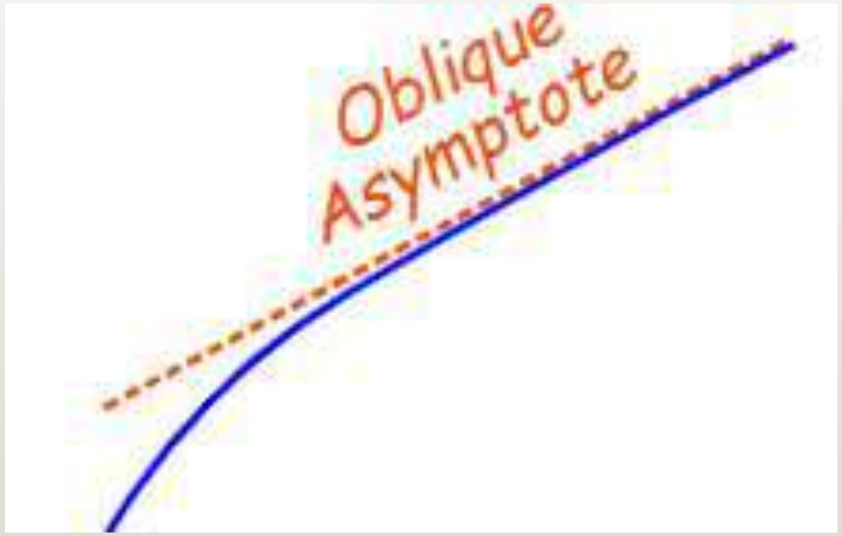


WHAT IS AN ASYMPTOTE?

- An *asymptote* is a line that a curve approaches, as it heads towards infinity. Here are the types.







BIG-O AND SMALL-O NOTATION

- In this analysis we consider only the highest order term of the expression for the running time of the algorithm, disregarding both the coefficient of that term and any lower order terms.

BIG-O NOTATION

- Consider the function

$$f(n) = 6n^3 + 2n^2 + 20n + 45$$

The *asymptotic notation* or *big-O notation* for describing this relationship is $f(n) = O(n^3)$.

BIG-O NOTATION

- Let f and g be functions $f, g: \mathbb{N} \rightarrow \mathbb{R}^{+}$.
- We say that $f(n) = O(g(n))$ if positive integer c and n_0 exist such that for every integer $n > n_0$

$$f(n) \leq cg(n).$$

When $f(n) = O(g(n))$ we say that $g(n)$ is upper bound for $f(n)$.

BIG-O NOTATION

- Intuitively, $f(n) = O(g(n))$ means that f is less than or equal to g if we disregard differences up to a constant.

BIG-O NOTATION

- Ex. Let $f_1(n) = 5n^3 + 2n^2 + 22n + 6$

We can write $f_1(n) = O(n^3)$ i.e., $g(n) = n^3$

Let $c = 6, n_0 = 10.$ Then according to definition,

$$f_1(n) \leq cg(n)$$

i.e., $5n^3 + 2n^2 + 22n + 6 \leq 6n^3$ for every $n_0 \geq 10.$

BIG-O NOTATION

- The big-O interacts with logarithms in a particular way.
- While using logarithms, we must use the base, as in $x = log_2 n$ which means $2^x = n$.
- Changing the base b changes the value of $log_b n$ by a constant factor, owing to the identity $n = \frac{log_2 n}{log_2 b}$.

BIG-O NOTATION

- Thus, when we write $f(n) = O(\log n)$, specifying the base is no longer necessary.

Ex: Let $f_2(n) = 3n\, log_2\, n + 5n\, log_2\, log_2\, n + 2.$

In this case $f_2(n) = \mathrm{O}(\mathrm{n} \log n)$ because $\log n$ dominates $\log \log n.$

BIG-O NOTATION

- Big-O also appears in arithmetic expressions such as the expression $f(n) = O(n^2) + O(n)$.
- In that case each occurrence of the O symbol represents a different suppressed constant.
- In this case $f(n) = O(n^2)$ because $O(n^2)$ term dominates $O(n)$.

BIG-O NOTATION

- When O symbol occurs in an exponent, as in the expression $f(n) = 2^{O(n)}$, the same idea applies. This expression represents an upper bound of 2^{cn} for some constant c.

BIG-O NOTATION

- The expression $f(n) = 2^{O(\log n)}$occurs in some analysis.
- Using the identity $n = 2^{log_2 n}$and thus that $n^c = 2^{clog_2 n}$, we see that $2^{O(\log n)}$represents an upper bound of n^c for some c.
- The expression $n^{O(1)}$represents the same bound in a different way, because the expression $O(1)$represents a value that is never more than a fixed constant.

BIG-O NOTATION

- Frequently we drive bounds of the form n^c for $c > 0$.
- Such bounds are called ***polynomial bounds***.
- Bounds of the form $2^{(n^\delta)}$ are called ***exponential bounds*** when $\delta \in \mathbb{R}$ and $\delta > 0$.

SMALL-O NOTATION

- Let f and g be functions $f, g : \mathbb{N} \rightarrow \mathbb{R}^{+}$.
- Say that $f(n) = o\big(g(n)\big)$ if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

In other words, $f(n) = o(g(n))$ means that, for any real number $c > 0$, a number n_0 exists, where $f(n) < cg(n)$ for all $n \geq n_0$.

SMALL-O NOTATION

- Check these out:
- $\sqrt{n} = o(n)$
- $n = o(n \log \log n)$
- $n \log \log n = o(n \log n)$
- $n \log n = o(n^2)$
- $n^2 = o(n^3)$

CLASS OF LANGUAGES

- Let $t: \mathbb{N} \to \mathbb{R}^+$ be a function. Define the *time complexity class*, $\boldsymbol{TIME(t(n))}$, to be the collection of all languages that are decided by an $O(t(n))$ time Turing Machine.

The language $A = \{0^k1^k | k \geq 0\} \in TIME(n^2)$ because M_1 decides A in time $O(n^2)$ and $TIME(n^2)$ contains all languages that can be decided in $O(n^2)$ time.

TIGHTER BOUND FOR $\{0^K 1^K\}$ ON TM

- Scan across tape, reject if 0 found right of 1
- Repeat as long as some 0s and 1s remain
  - Scan across tape and confirm total number of 0s and 1s is even
  - Scan again, crossing off every other 0 starting with the first 0 and every other 1, starting with first 1
- If no 0s and no 1s left, accept; else reject

COMPUTING THE TIGHTER BOUND

Each step cuts input size in half. Number of times to cut n in half: $\log_2 n$

- Scan across tape, reject if 0 found right of 1 — 2n steps
- Repeat as long as some 0s and 1s remain — log2 n loops
- Scan across tape and confirm number of 0s and 1s is even — 2n steps
- Scan again, crossing off every other 0 starting with the first 0 and every other 1, starting with first 1 — 2n steps
- If no 0s and no 1s left, accept; else reject — n steps

Total Steps: $2n + \log_2 n\,(2n+2n) + 2n = 4n + 4\,n \log 2n = O(\,n \log n)$

COMPLEXITY ON 2-TAPE TURING MACHINE COMPUTED

- Scan across tape and reject if 0 right of 1 n steps
- Scan across 0s on tape 1, writing each onto tape 2n steps
- Scan across the 1s on tape 1, removing a 0 from tape 2 for each 1 k
- If run out of 0s while still reading 1s, reject n-k
- If 0s left after 1s finished, reject n – k steps
- If no 0s left, accept

Total Steps: 3n = O(n)