Algorithm

L-1

• **Algorithm:** step by step procedure to solve a specific problem.

L-2

• **Program:** program is also a step by step procedure to solve a specific problem.

algorithm	program
- abstract concept	- concrete implementation.
- Written in any language	- Written in programming language only.
- developed during design phase	- development during development phase.
- doesn't depent on h/w or OS	- depent on hardware or operating system.
- always analyzed.	- program is always tested.

L-3

Characteristic of an algorithm:

- input: zero or more than one or one
- Output: at least one output.
- finiteness: must terminate in finite time.
- definiteness: An algorithm must be clear and unambiguous
- effectiveness: Should take less time and space to execute

L-4

Significance of Algorithms:

- A bad algorithm can lead to longer execution time.

L-5

- A good algorithm will lead to shorter execution time.

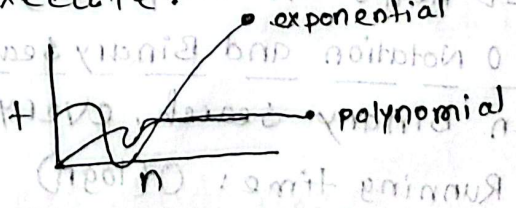
Decidable problems:

- the problem for which efficient algorithm exists.
- takes polynomial time or less to execute.

Undecidable problem:

- the problem for which no efficient algorithm exists.
- takes exponential time to execute.

polynomial vs exponential:



L-6

the nature of undecidable problem:

- it takes exponential time, that need more time

L-7 Analysis of Algorithm are two types:

posteriori vs priori Analysis:

priori Analysis	posteriori analysis
• Estimation of time and memory space required by an algorithm before executing it on the system.	- Calculation of time and memory space required by an algorithm after executing it on the system.
• independent of the programming language.	• Dependent on the programming language.
• Independent of the hardware	• Dependent on the hardware.

L-9

Importance of Algorithm Analysis:

example of linear search:

example of Binary search: faster

L-10

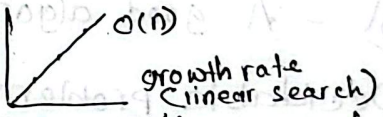
importance of growth rate:

- As the size of the input grows, speed of the binary search grows drastically.
- One example is not enough to analyze algorithms.
- Important to know how fast an algorithm grows with respect to the input size.

* Rate at which running time increases as a function of input is called the growth rate.

L-11

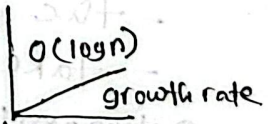
The big O Notation:



- Way to describe the performance of an algorithm with respect to the size.
- In linear search, the number of comparisons is directly proportional to the input size. It's running time is O(n).
- Big O notation doesn't tell speed in seconds
- It helps in comparing the number of operations.

L-12

Big O Notation and Binary search:



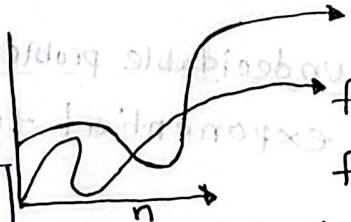
- In binary search, every time half of the list is eliminated.

L-13

- Running time: O(logn)

growth rate comparison of functions:

definition: Assuming f(n) and g(n) are non-negative functions. f(n) = O(g(n)) iff f(n) ≤ c·g(n) for all value of n where n ≥ n₀ and c and n₀ are constants.



f(n) = O(g(n)).

g(n) is the tight upper bound of f(n)

L-14

Pblm-1: Assume that $f(n) = 5n+50$ and $g(n) = n$. Is $f(n) = O(g(n))$?

⇒ Assuming $c = 6$.

$5n+50 = O(n)$

for $c = 6$ and $n \geq 50$

$n = 10$	$f(n) = 100$	$c \cdot g(n) = 60$
$n = 20$	$f(n) = 150$	$c \cdot g(n) = 120$
$n = 50$	$f(n) = 300$	$c \cdot g(n) = 300$
$n = 51$	$f(n) = 305$	$c \cdot g(n) = 306$.

Pblm-2: Assume that $f(n) = 5n+50$ and $g(n) = \log_{10} n$. Is $g(n)$ upper bound of $f(n)$?

⇒ Assuming $c = 1000$

$f(n) = 5n+50$

$c \cdot g(n) = 1000 \log_{10} n$

$n = 10$	$f(n) = 100$	$c \cdot g(n) = 1000$
$n = 10^2$	$f(n) = 550$	$c \cdot g(n) = 2000$
$n = 10^3$	$f(n) = 5050$	$c \cdot g(n) = 3000$
$n = 10^4$	$f(n) = 50050$	$c \cdot g(n) = 4000$.

So, $g(n)$ is not an upper bound of $f(n)$.

L-15

pblm-1: find the upper bound for $f(n) = 3n+8$.

⇒ **Step 1:** identify the dominant term.

$(3n+8)$ here $3n$ is dominant term. 8 is constant.

Step 2: choose $g(n)$ according to the dominant term.

n, $n\log n$, n^2, n^3 ..... ← $g(n)$

Step 3: Apply the Big O notation.

now, assuming $c = 4$; is $3n+8 \leq 4n$?

($f(n)$ ≤ $g(n)$)

	$f(n)$	$g(n)$
$n = 7$	29	28
$n_0 \rightarrow n = 8$	32	32
$n = 10$	38	40

yes, $3n+8 \leq 4n$ is true

$\therefore 3n+8 = O(n)$ for $c = 4$ and $n_0 = 8$.

pblm 2: find the upper bound for $f(n) = n^2+10$.

1: dominant term $= n^2$

2: Choose $g(n) = n^2$

3: apply big O notation-

assuming $c = 2$

Is $n^2+10 \leq 2n^2$?

	$f(n)$	$g(n)$
$n = 1$	11	2
$n = 2$	14	8
$n = 3$	19	18
$n = 4$	26	32

So, $n^2+10 \leq 2n^2$ is true,

$\therefore n^2+10 = O(n^2)$ for $c = 2$ and $n_0 = 4$.

L-16

Pblm-1: find the upper bound for $f(n) = 2n^3 - 2n^2$?

⇒ 1: dominant term : $2n^3$

2: choose $g(n) = n^3$.

3: big O defination.

assuming $c = 2$; ∄

Is $\underbrace{2n^3 - 2n^2}_{f(n)} \leq \underbrace{2n^3}_{g(n)}$?

	$f(n)$	$g(n)$
$n_0 \rightarrow n = 1$	0	1
$n = 2$	8	16
$n = 3$	36	54
$n = 4$	96	128

So, $2n^3 - 2n^2 \leq 2n^3$ is true.

∴ $2n^3 - 2n^2 = O(n^3)$ for $c = 2$ and $n_0 = 1$.

Pblm-2: find the upper bound for $f(n) = n^4 + 100n^2 + 35$.

⇒ 1: dominant term = n^4.

2: choose $g(n) = n^4$.

3: Apply Big O notation defination

assuming $c = 2$

Is $n^4 + 100n^2 + 35 \leq 2n^4$

	$f(n)$	$g(n)$
$n = 1$	136	2
$n = 3$	1016	162
$n = 5$	3160	1250
$n = 10$	20035	20000
$n = 11 \rightarrow n_0$	26776	29282

so, $n^4 + 100n^2 + 35 \leq 2n^4$ is true.

$n^4 + 100n^2 + 35 = O(n^4)$ for $c = 2$ and $n_0 = 11$.

L-18

Common Big O Runtime:

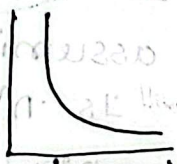
best ↑ better

- $O(\log n)$ also known as log time. Ex: binary search.
- $O(n)$ also known as linear time. Ex: linear search.
- $O(n \log n)$. Ex: quicksort.
- $O(n^2)$ also known as polynomial time. Ex: Selection sort.
- $O(n!)$ also known as exponential time. Ex: traveling sakesman.

L-19

Decrement functions:

- A function in which the denominator is bigger than the numerator. Ex: $\frac{c}{n}, \frac{n}{n^2}, \frac{n}{2^n}, \frac{n^2}{3^n} \ldots\ldots$



Pblm: Arrange the following functions in the order of decrease from slowest to fastest. $\frac{100}{n}, \frac{n}{2^n}, \frac{n}{n^2}, \frac{n^2}{3^n}, \frac{n^3}{3^n}$

⇒ **Step 1:** Simplify the functions.

$\frac{100}{n}, \frac{n}{2^n}, \frac{1}{n}, \frac{n^2}{3^n}, \frac{n^3}{3^n}$.

Step 2: Compare the denominators and arrange:

$$\frac{100}{n}, \frac{1}{n}, \frac{n}{2^n}, \frac{n^2}{3^n}, \frac{n^3}{3^n}.$$

Step 3: Compare the numerators and arrange:

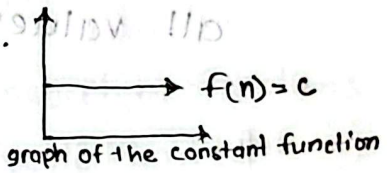
$$\frac{100}{n} > \frac{1}{n} > \frac{n}{2^n} > \frac{n^3}{3^n} > \frac{n^2}{3^n}$$

slowest → fastest.

L-20

Constant functions: A function parallel to x-axis.

Ex: 100, 500392 .... etc

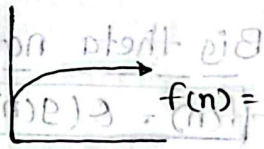


* Constant function is asymptotically bigger than the decrement function.

L-21

Logarithmic functions: A function that grows positively, but with slower growth rate.

Ex: $\log n$, $(\log n)^{10}$, $\log\log n$, $\log\log n$, $\log\log\log n$ ....

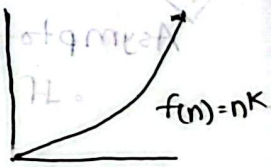


* logarithomic functions is asymptotically bigger than the constant ~~time~~ function.

L-22

polynomial functions: A function whose growth rate increases polynomially.

Ex: $n^{0.1}$, $\sqrt{n}$, n^2, $n\log n$, n^K, etc.

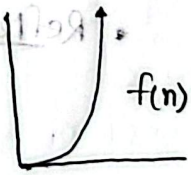


* polynomial function is asymtotically bigger than the logarithmic function.

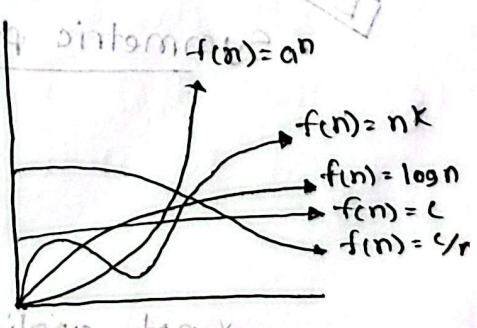
L-23

Exponential functions: A function whose growth rate increases rapidly.

Ex: 2^n, 3^n, $n!$, n^n, a^n, etc....



* Exponential function is asymptotically bigger than the polynomial function.



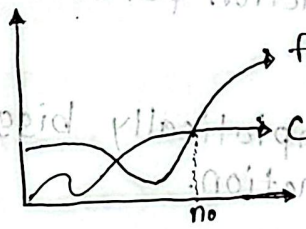
So,

Decrement functions < const functions < log functions < polynomial functions < exponential functions.

L-27

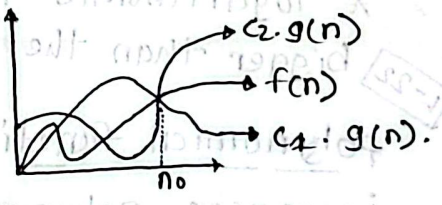
Big Omega Notation: Describe the lower bound of an algorithm.
- used to express the best case running time of an algorithm.

Defination: Assuming $f(n)$ and $g(n)$ are non-negative functions.
$f(n) = \Omega(g(n))$ iff $f(n) \geq c \cdot g(n)$ for some $c > 0$ and for all values of n where $n \geq n_0$ and c and n_0 are constance.



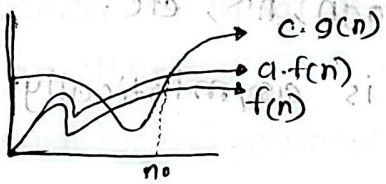
L-28

Big theta notation: Assuming $f(n)$ and $g(n)$ are non-negative functions.
$f(n) = \Theta(g(n))$ iff $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all values of n where $n \geq n_0$ and c_1, c_2 and n_0 are constants.
- $g(n)$ is the asymptotic tight bound of $f(n)$...



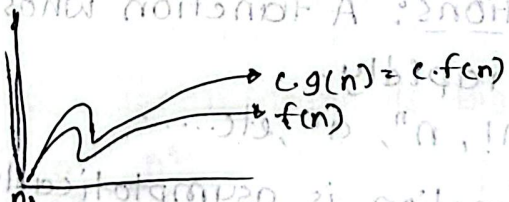
L-30

Asymptotic Notations- general property:
- If $f(n) = O(g(n))$, then $a * f(n) = O(g(n))$.



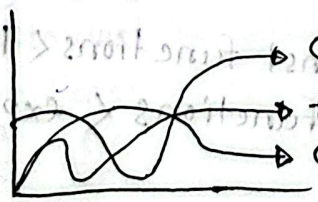
L-31

- **Reflexive property:** If $f(n)$ is given, then $f(n) = O(f(n))$.



L-32

- **Symmetric property:** If $f(n) = \Theta(g(n))$, then $g(n) = \Theta(f(n))$.

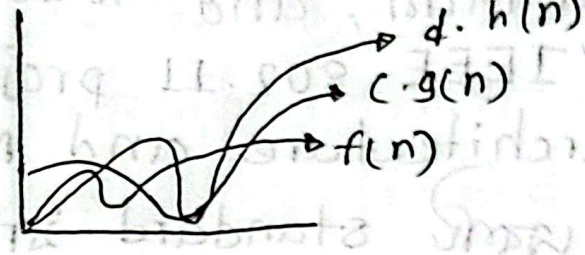


*not applicable for Big O and Big Ω notations.

L-34

∴ ~~Asy~~ Transitive property:

if $f(n) = O(g(n))$, and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.



L-35

- Addition property: $f(n) + g(n) = O(\max(f(n), g(n)))$.
- Multiplication property: $f(n) * g(n) = O(f(n) * g(n))$.

L-36

Little o notation: Assuming $f(n)$ and $g(n)$ are non-negative functions, $f(n) = o(g(n))$ iff $f(n) < c \cdot g(n)$ for all values of $c > 0$ and for all values of n where $n \geq n_0$ and c and n_0 are constant.

- If $f(n) = o(g(n))$, then $f(n) = O(g(n))$.
- if $f(n) = O(g(n))$, then it may or may not be true that $f(n) = o(g(n))$