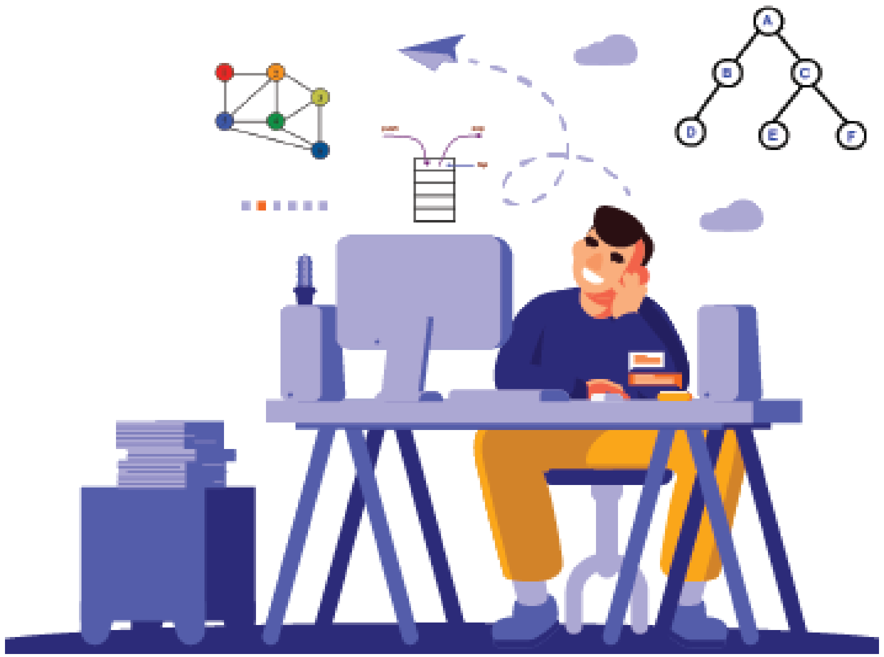
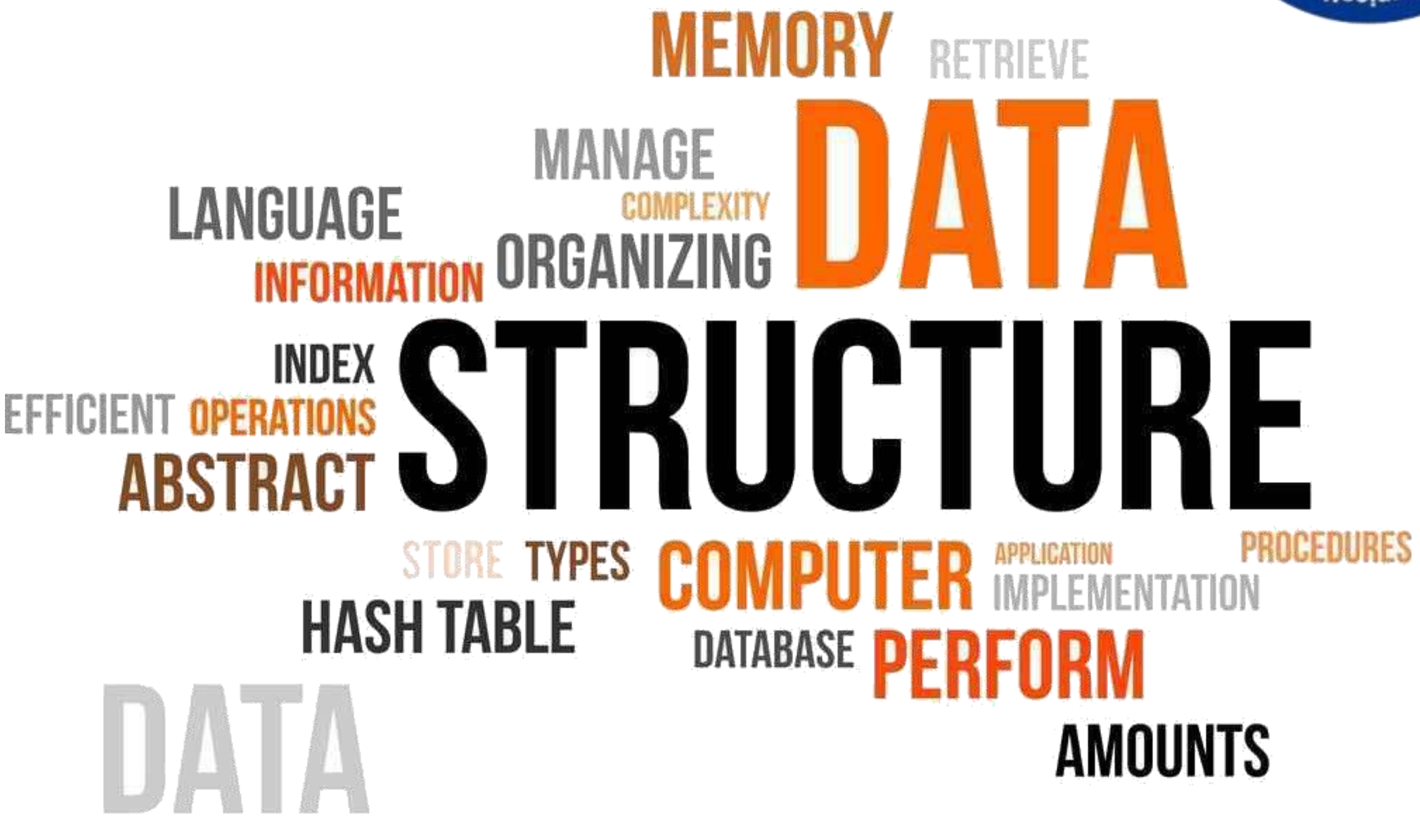
Data Structures

Course code: IT623





Dr. Rahul Mishra
Assistant Professor
DA-IICT, Gandhinagar

Algorithmic analysis and runtime

- The complexity of the search algorithm is given by the number **C** of comparisons between
  ITEM and **DATA[K].**

- We seek **C(n)** for the worst case and the average case.

Worst case:
The worst case occurs when **ITEM** is the last element in the array **DATA** or is not there.

$$\mathbf{C(n) = n}$$

Accordingly, **C(n) = n** is the worst-case complexity of the linear search algorithm.

Algorithmic analysis and runtime

- We assume that the **ITEM** does appear in the **DATA** and that it is equally likely to occur at any position in the array.

- The number of comparisons can be any of the numbers: **1, 2, 3, …, n**, and each number occurs with the probability **p= 1/n**.

$$
\begin{aligned}
C(n) &= 1.\ 1/n + 2.\ 1/n + \dots + n.\ 1/n \\
&= (1+2+3+ \dots + n).\ 1/n \\
&= n(n+1)/2\ .\ 1/n = (n+1)/2.
\end{aligned}
$$

- This agrees with the intuitive feeling that the average number of comparisons needed to find the location of ITEM is approximately equal to half the number of elements in the DATA list.

Algorithmic analysis and runtime

- The complexity of average case of an algorithm is usually much more complicated to analyses than that of the worst case.

- The probabilistic distribution that one assumes for the average case may not actually apply to real situations.

- Thus, unless otherwise stated or implied, the complexity of an algorithm shall mean the function which gives the running time of the worst case in terms of the input size.

- Moreover, the complexity of the average case for many algorithms is proportional to the worst case.

Growth of function

* Suppose M is an algorithm, and suppose n is the size of the input data.

* The complexity $\boxed{f(n) \text{ of } M}$ increases as $\boxed{n}$ increases.

* It is usually the rate of increase of $\boxed{f(n)}$ that we want to examine.

* This is usually done by comparing $f(n)$ with some standard function

$$\boxed{\log_2 n, \quad n, \quad n\log_2 n, \quad n^2, \quad n^3, \quad 2^n}$$

n \ $g(n)$	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

"Rate of Growth of Standard Functions"

Asymptotic Analysis

- ***Dictionary meaning:*** *"Asymptotic function approaches a given value as an expression containing a variable tends to infinity."*

- We are concerned with how the **running time** of an algorithm increases with the size **of the input in the limit,** **as the size of the input increases without bounds.**

- An algorithm that is asymptotically more efficient will be the best choice for all but very small input.

- The notations we use to describe the asymptotic running time of an algorithm are defined in terms of functions whose domain is the set of natural numbers, **N** = {0,1,2,…}.

- They are used for defined worst-case running-time function **T(n),** which usually is defined only on integer input sizes.

Asymptotic Analysis

- We might extend the notation to the domain of ***real numbers or alternatively*****,** restrict it to a subset of the natural number.

- The function to which we apply "*asymptotic notation*" will usually characterize the running times of the algorithm.

- In addition, the asymptotic notation can apply to functions that characterize some other aspects of the algorithm (the amount of space they used).

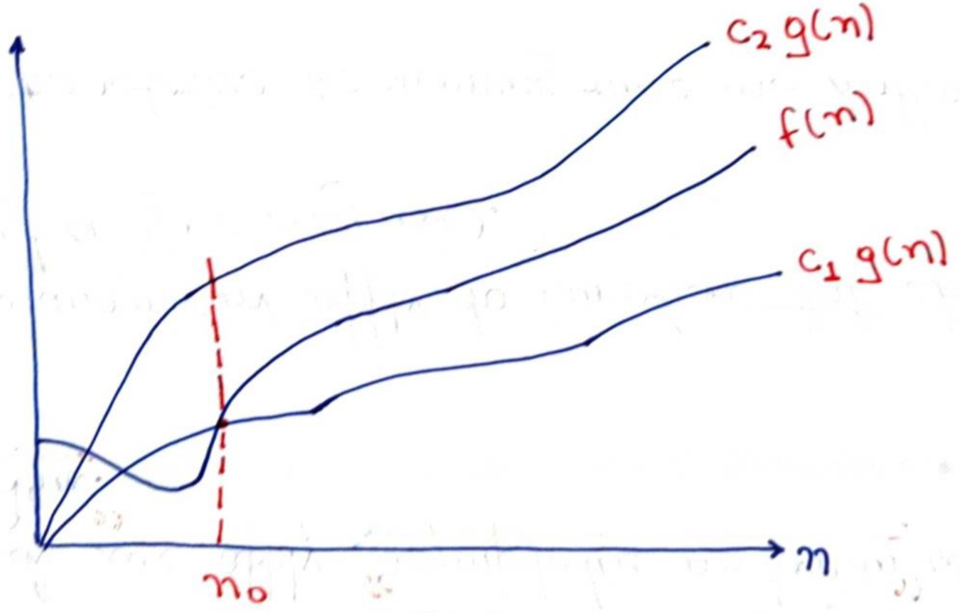
- We often wish to make/ characterize the running time no matter what the input.

- *Asymptotic notations are well suited to characterizing running times no matter what the input.*

Asymptotic Analysis: $\theta - notation$

Let us define what this notation mean. For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions.

* $\boxed{\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \geq n_0 \}.}$



$f(n) = \Theta(g(n))$

Asymptotic Analysis: $\theta - notation$

Θ - Notation

* A function $\boxed{f(n)}$ belongs to the set $\boxed{\Theta(g(n))}$ if there exist positive constants $\boxed{c_1}$ and $\boxed{c_2}$ such that it can be "sandwiched" between $\boxed{c_1 g(n) \text{ and } c_2 g(n)}$, for sufficiently large $\boxed{n}$.
    * → (read as f of n is theta of g of n)

* Since $\Theta(g(n))$ is a set, we can write "$f(n) \in \Theta(g(n))$" to indicate that $f(n)$ is a member of $\Theta(g(n))$.

* Instead, we will usually write "$f(n) = \Theta(g(n))$" to express the same notion.

* An intuitive picture of functions $\boxed{f(n) \text{ and } g(n)}$, where $f(n) = \Theta(g(n))$.

* For all values of n at and to the right of n_0, the value of $f(n)$ lies at or above $\boxed{c_1 g(n)}$ and at or below $\boxed{c_2 g(n)}$.

* $g(n)$ is an asymptotically tight bound for $f(n)$.

Asymptotic Analysis: $\theta - notation$

The definition of θ(g(n)) require that every member $f(n) \in \theta\left(g_{(n)}\right)$ by asymptotically non-negative, that is, that f**(n)** be non-negative whenever **n** is sufficiently large.

Example → $f(n) = 18n + 9$

since $f(n) \geq 18n$ and $f(n) \leq 27n$

for $n \geq 1$

- $g(n) = 3n$

$$c_1 3n \leq \underbrace{18n + 9}_{f(n)} \leq c_2 3n$$

$c_1 = 6 \qquad c_2 = 9$

Asymptotic Analysis: $\theta - \boldsymbol{notation}$

* Let us briefly justify this intuition by using the formal definition to show that

$$\boxed{\tfrac{1}{2}n^2 - 3n = \Theta(n^2)}$$

* To do so, we must determine positive constants c_1, c_2, and n_0 such that

$$\boxed{c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2}$$

for all $n \geq n_0$. Dividing by n^2 yields.

$$\boxed{c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2}$$

* We can make the right-hand inequality hold for any value of $\boxed{n \geq 1}$ by choosing any constant $\boxed{c_2 \geq 1/2}$.

* We can make the left-hand inequality hold for any value of $\boxed{n \geq 7}$ by choosing any constant $\boxed{c_1 \le 1/14}$.

Asymptotic Analysis: $\theta - notation$

* By choosing $c_1 = 1/14$, $c_2 = 1/2$, and $n_0 = 7$, we can verify that $\boxed{\frac{1}{2}n^2 - 3n = \Theta(n^2)}$.

* Certainly, other choices for the constants exist, but the important thing is that some choice exists.

* These constants depend on the function $\boxed{\frac{1}{2}n^2 - 3n}$; a different function belonging to $\boxed{\Theta(n^2)}$ would usually requires different constants.

* We can also use the formal definition to verify that $\boxed{6n^3 \neq \Theta(n^2)}$. Suppose for the purpose of contradiction that $\boxed{c_2}$ and $\boxed{n_0}$ exist such that

$$\boxed{6n^3 \leq c_2 n^2} \quad \text{for all } n \geq n_0.$$

* But then dividing by $\boxed{n^2}$ yield $\boxed{n \leq c_2/6}$, which cannot possibly hold for arbitrarily large n, since c_2 is constant.

Asymptotic Analysis: $\theta - notation$

* The lower-order terms of an asymptotically positive function can be ignored in determining asymptotically tight bounds because they are insignificant for large n.

* When n is large, even a tiny fraction of the highest-order term suffices to dominate the lower-order terms.

* Thus, setting c_1 to a value that is slightly smaller than the coefficient of the highest-order term and setting c_2 to a value that is slightly ~~smaller than the coefficient~~ larger permits the inequalities in the definition of Θ-notation to be satisfied.

$$\boxed{f(n) = an^2 + bn + c}$$

where $a, b,$ and c are constants and $a > 0$.

* The lower-order terms and ignoring the constant yields $f(n) = \Theta(n^2)$.

* To show the same thing, we take the constants $\boxed{c_1 = a/4,\ c_2 = 7a/4,\ \text{and}\ n_0 = 2}$.

$$\boxed{n_0 = 2 \cdot \max\left(|b|/a, \sqrt{|c|/a}\right)}$$

Asymptotic Analysis: $\theta - \boldsymbol{notation}$

We can verify that $0 \leq c_1 n^2 \leq an^2 + bn + c \leq c_2 n^2$ for all n>= n0.

* In general, for any polynomial $\boxed{p(n) = \sum_{i=0}^{d} a_i n^i}$, where the a_i are constants and $a_d > 0$

* We have $p(n) = \Theta(n^d)$.

* Since any constant is a degree-0 polynomial, we can express any constant function as $\Theta(n^0)$, or $\Theta(1)$.

* We shall often use the notation $\Theta(1)$ to mean either a constant or a constant function with respect to some variable.

Asymptotic Analysis: $O - notation$

* The Θ-notation asymptotically bounds a function from above and below.

* When, we have only an asymptotic upper bound, we use O-notation

* For a given function $g(n)$, we denote by $O(g(n))$ (pronounced "big-oh of g of n" or sometimes just "oh of g of n") the set of functions.

$$\boxed{O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \geq n_0\}}$$

* Precise definition : "Schaum's outlines"

Suppose $f(n)$ and $g(n)$ are functions defined on positive integers with the property that $f(n)$ is bounded by some multiplier of $g(n)$ for almost all n. That is, suppose there exist a positive integer n_0 and a positive number c such that, for all $n > n_0$, we have

Asymptotic Analysis: $O - notation$

$$\boxed{|f(n)| \le c\,|g(n)|}$$

This is also written as: $\boxed{f(n) = O(g(n))}$

It is read as "f(n) is of order g(n)." For any polynomial P(n) of degree m, we show in solved ~~that~~ $P(n) = O(n^m)$; e.g.;

$$\boxed{8n^3 - 576n^2 + 832n - 28 = O(n^3)}$$

We can also write

$$\boxed{f(n) = h(n) + O(g(n)) \text{ when } f(n) - h(n) = O(g(n))}$$

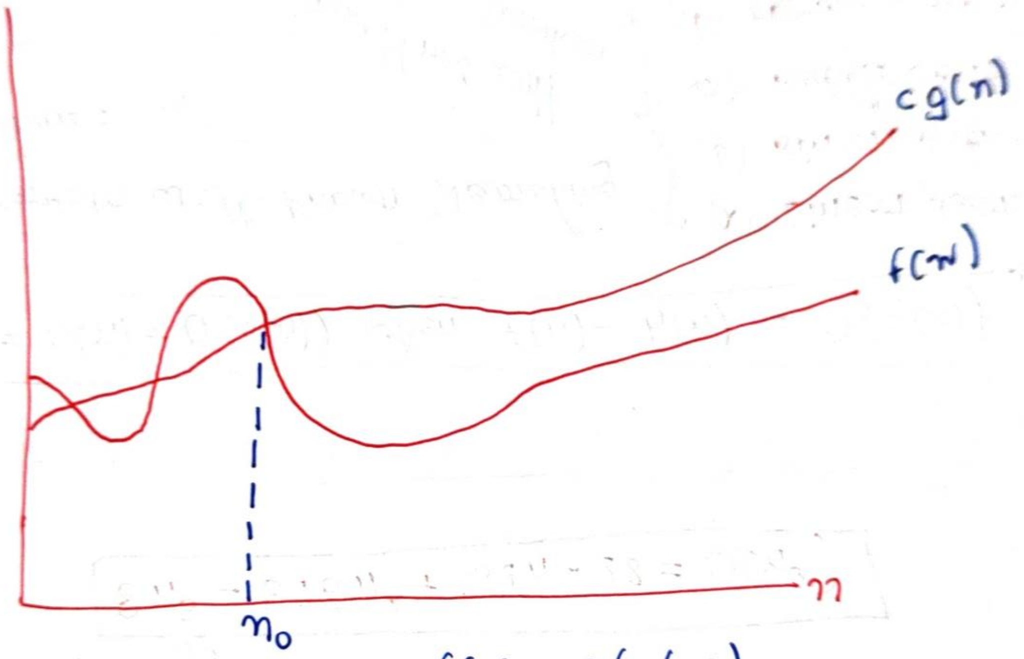
The complexity of certain well-known searching and sorting algorithms:

(a) Linear search: $O(n)$
(b) Binary search: $O(\log n)$
(c) Bubble sort: $O(n^2)$
(d) Merge sort: $O(n \log n)$.

These algorithms will be discussed in further lectures.

Asymptotic Analysis: $O - notation$

* We use O-notation to give an upper bound on a function, to within a constant factor.



$f(n) = O(g(n))$

* For all values n at and to the right of n_0, the value of the function f(n) is on or below c g(n).

Asymptotic Analysis: $O-notation$

* We write $\boxed{f(n) = O(g(n))}$ to indicate that a function $\boxed{f(n)}$ is a member of the set $\boxed{O(g(n))}$.

* $\boxed{f(n) = \Theta(g(n))}$ implies $\boxed{f(n) = O(g(n))}$, since Θ-notation is a stronger notation than O-notation.

* Written set-theoretically, we have $\boxed{\Theta(g(n)) \subseteq O(g(n))}$.

* Our proof that any quadratic function $\boxed{an^2+bn+c}$ where $a>0$ is in $\Theta(n^2)$ also shows that any such quadratic function is in $O(n^2)$.

* A surprizing aspect:

  when $a>0$, any linear function $\{an+b\}$ is $O(n^2)$

* It is easily verified by taking $c = a + |b|$ and $\boxed{n_0 = max(1, -b/a)}$.

Asymptotic Analysis: $O - notation$

* Using O-notation, we can often describe the running time of an algorithm merely by inspecting the algorithm's overall structure.

* For example, the doubly nested loop structure of insertion sort algorithm immediately yields an $O(n^2)$ upper bound on the worst-case running time:

* Since O-notation describes an upper bound, when we use it to bound the worst-case running time of an algorithm, we have a bound on the running time of the algorithm on every input.

* $O(n^2)$ bound on worst-case running time of insertion sort also applies to its running time on every input.

* When we say "the running time is $O(n^2)$", we mean that there is a function $f(n)$ that is $O(n^2)$ such that for any value of n, no matter what particular input of Size n