P, NP, NP-hard, NP complete.

Subject:.....

Date:.....

NP completeness

algorithm

- polynomial time
  - (i) Linear search - $O(n)$
  - (ii) Binary " - $O(\log n)$
  - (iii) Insertion sort - $O(n^2)$
  - (iv) merge " - $O(n \log n)$
- exponential time
  - (i) 0/1 knapsack - 2^n
  - (ii) Traveling salesman - 2^n
  - (iii) sum of subset - 2^n
  - (iv) Graph coloring - 2^n

(i) P-class : class of problem can be solve in polynomial time

→ run fast (efficient)

Ex: (i) Binary search

(ii) sorting (merge, quick etc)

(iii) shortest path

NOTE: Easy to solve & verify.

there exist an algorithm that can solve the problem in $O(n^k)$, k be constant →

(11) NP class: (Nondeterministic polynomial time)

A problem cannot solve in polynomial time but solution can be verified by polynomial time, even if finding the solution may be slow.

Ex:

- → Traveling salesman problem (TSP)
- → subset sum
- → Hamilton cycle

Note: → Hard to solve
→ ~~verify to~~ Easy to verify

code: (non-deterministic algorithm)

```
nsearch (A, n, key)
{ j=choice(); ↗3
  if (A[j] = key)  ↓7
  {
    write(j);
  } success();
  write(0);
} failure();
```

key=7

A | 5 | 6 | 7 | 8 |

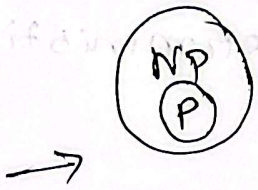
A[3] = 7
↑ [illegible] index [illegible]

[illegible] non deterministic algorithm

TC = O(1)

Subject:.....

Date:.....



NP-hard: A problem is NP-hard if every problem in NP can be reduced to it in polynomial Time, but it doesnot have to be in NP ~~as~~ itself.

Key points:

→ They are very hard problem

→ They may not belong to NP

→ They may not have polynomial time verification.

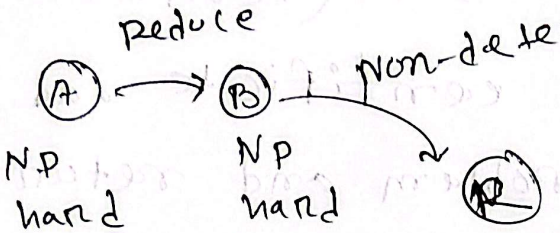
Ex:
- Traveling Salesman problem (optimization version)
- Knapsack (optimization version)
- Halting problem (undecidable)
- No Hamilton cycle.

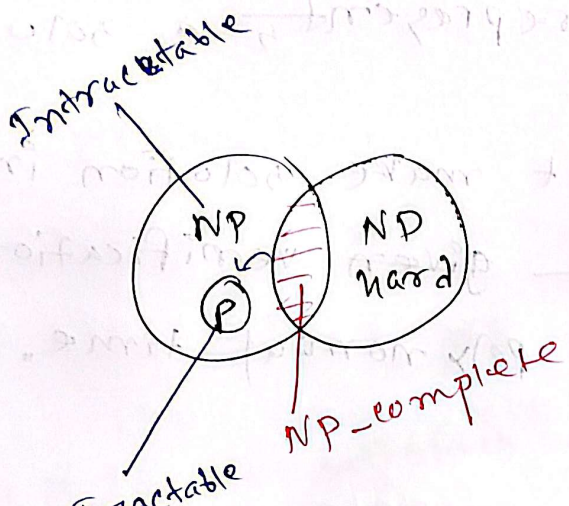
Note:
① Hard to solve
② Hard to verify
③ may not be in NP or P

satisfiability - 2n

Subject:.....

Date :.....





Tractable = solveable in polynomial time (easy)

Nontractable = requires exponential time or no efficient algorithm (hard)

Example:

NP: Non deterministic algorithm
two stages procedure :-

① Non deterministic ~~stage~~ (guessing) stage:

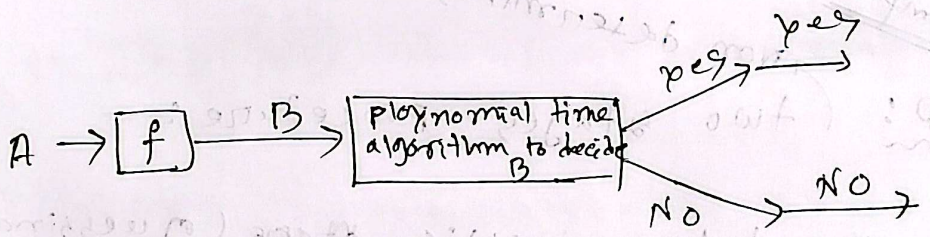
generate an arbitary string that can be thought of as a candidate solution ("certificate")

Subject:.....
Date:.....

(2) Deterministic (verification) stage :-
take the certificate and the instance to the problem and return yes if the certificate represents a solution.

So NP do not make solution in polynomial time but ~~given~~ verification of given solution in polynomial time.

Polynomial reduction algorithm



→ ~~To~~ Poly solve a decision problem A in polynomial time

(1) Use a polynomial time reduction algorithm to transform A into B

(2) run a known ~~algorithm~~ polynomial time for B

(3) use the ~~answer~~ for B as the ans. for A.

Subject:.....
Date :.....

$A(i) = yes \Leftrightarrow B(f(i)) = yes$

NP-completeness

A problem B is NP-complete if

① $B \in NP$

② $A \leq_p B$ for all $A \in NP$

→ If B satisfies only property ② then we say that B is NP-hard

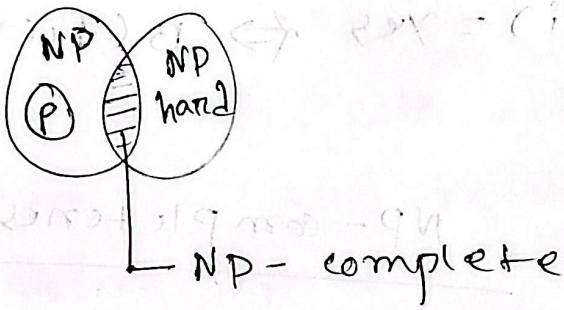
No polynomial time algorithm has been discovered for an NP-complete problem.

Defination: A problem is NP-complete if:

1. It is in NP → its soln can be verified in polynomial time.
2. It is in NP-Hard → every problem in NP can be reduced to it in polynomial time.

Subject:.....

Date :.....



features:

① NP complete problems are special because every NP problem can be reduced to them in polynomial time.

② If any NP-complete problem is solved in polynomial time, then all NP problems can also be solved in polynomial time.

Ex: → SAT (boolean satisfiability)

→ clique problem (deciding if there is a k-clique)

→ vertex cover

Subject:.....

Date:.....

Proving NP-complete

theorem:

If A is NP-complete & $A \leq_p B$

$\Rightarrow$ B is NP-Hard

In addition, if $B \in NP$

$\Rightarrow$ B is NP-complete.

proof: Assume that $B \in P$

since $A \leq_p B \Rightarrow A \in P$ [contradiction]

$\Rightarrow$ B is NP-hard.

P & NP complete problem

① shortest simple path:

- Given a graph $G = (V, E)$, find a shortest path from a source to all other vertices
- polynomial time: $O(VE)$

② longest simple path: Given a graph $G = (V, E)$ find a longest path from

Subject:.....

Date:.....

a source to all other vertices

– NP- complete.

③ Euler tour:

→ visit each edge of G exactly once (may visit vertex multiple times)

→ ~~Poly tom~~ polynomial time O(E)

④ Hamilton cycle

– $G_b = (V, E)$ a connected graph find a cycle that visits each vertex of G_b exactly once.

– NP – complete.

Subject:.....
Date:.....

Application of NP-complete:

Even through NP-complete problem are hard to solve exactly, but they are used in many real life optimization task.

(1) Network Design:

- finding optimal routes
- laying min. cost cables
- Designing communication network (related to steiner Tree, Vertex cover, clique)

(2) Scheduling:

- Job scheduling on machine
- Exam timetabling
- workforce / task scheduling (related to graph coloring)

Subject:.....
Date:.....

③ Transportation & logistics:

- Optimal delivary route planning
- Vechile routing
- Traveling salesman problem

④ Cryptography:

→ Many cryptography systems rely on NP - hard assumption
- Hardness of solving Integer factorization, subset sum.

⑤ AI & macro. ML:

- Featurer selection
- clustering
- optimal decision making (related to k-clique, set cover, SAT)

Subject:.....

Date:.....

⑥ Resource Allocation:

- Optimal assignment of limited resources
- knapsack type optimization

⑦ pattern Recognition:

- Image segmentation
- Optimal matching (related to graph matching, eliques)

~~So, At~~ Therefore, NP-complete problems are used whenever optimal decision making is needed but exact solution are too slow, so heuristics or approximation are used.

Chapter-3

Order of growth: The order of growth of an algorithm describes how its running time increases as the input size n grows large.

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 constants.

→ $g(n)$ is strictly upper bound of $f(n)$.

Examples:

$f(n) = n$ and $g(n) = n^2$

$\Rightarrow$ For $c > 0$, $f(n) < c \cdot g(n)$

Assuming $c = 1/8$.

$n < \frac{1}{8} n^2 \Rightarrow 8n < n^2 \Rightarrow 8 < n$

$\Rightarrow n > 8$

$\therefore$ $n < \frac{1}{8} n^2$ is true for $n_0 = 9$

and $c = \frac{1}{8}$.

Here $g(n)$ is strictly greater than $f(n)$

$\therefore f(n) = O(g(n))$.

If, $f(n) = O(g(n))$, then $f(n) = O(g(n))$

Little ω (omega) notation

Assuming $f(n)$ and $g(n)$ are non-negative functions. $f(n) = \omega(g(n))$ 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 constants.

→ $g(n)$ is strictly lower bound of $f(n)$

Examples: $f(n) = n^2$, $g(n) = n$

⇒ for $c > 0$, $f(n) > c \cdot g(n)$

Assume, $c = 1000$

$n^2 > 1000n$ is true

for $n_0 = 1001$.

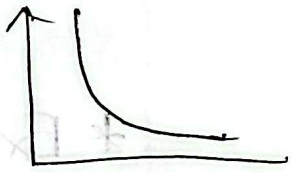
$\therefore f(n) = \omega(g(n))$.

If $f(n) = \omega(g(n))$, then $f(n) = \Omega(g(n))$

Decrement function < Const Function < Log function < Polynomial function < Exponential function.

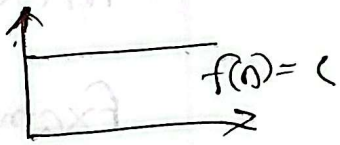
* Decrement functions: denominator > numerator

Example: $\frac{c}{n}$, $\frac{n}{n^2}$, $\frac{n}{2n}$, $\frac{n^2}{3n}$



parallel to x-axis.

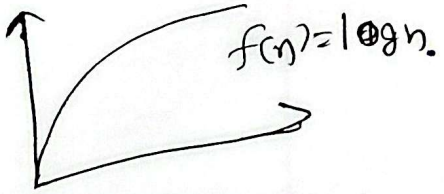
* Constant functions:



Example: 100, 1 billion.

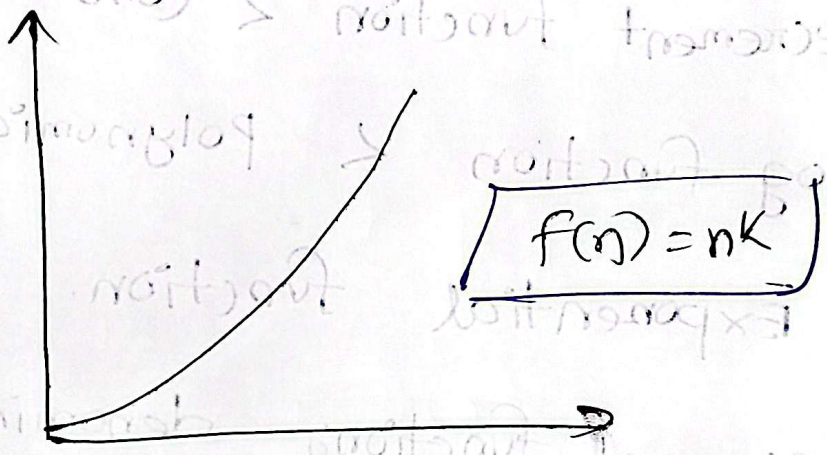
* Log functions: grows positively, but with slower growth rate.

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



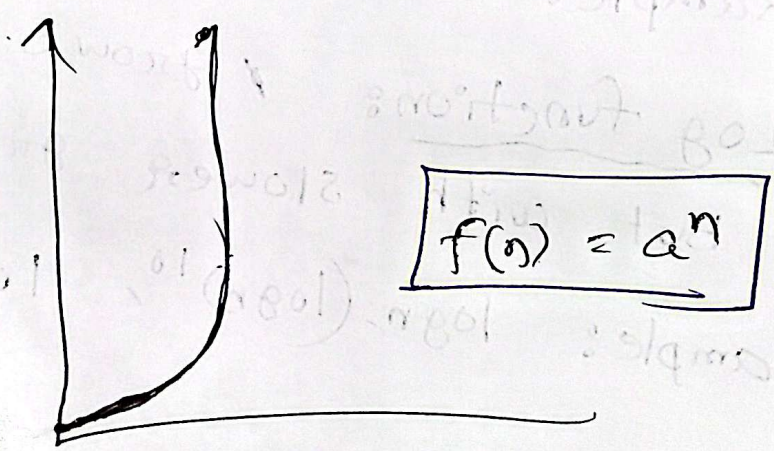
* polynomial function: growth rate increases polynomially.

Example: $n^{0.1}$, $\sqrt{n}$, n^2, $n \log n$, n^k etc.



* Exponential functions: growth rate increases rapidly.

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



Exercises:

3.2-1: Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ notation, prove that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.

$\Rightarrow$ We are to prove that if $f(n)$ and $g(n)$ are asymptotically nonnegative functions, then:

$$\max(f(n), g(n)) = \Theta(f(n) + g(n))$$

By the definition of Θ-notation, we must find positive constants c_1, c_2 and n_0 such that for all $n \geq n_0$.

$$c_1(f(n) + g(n)) \leq \max(f(n), g(n)) \leq c_2(f(n) + g(n))$$

* Upper bound:

Assume $c_2 = 2$ then

$$f(n) \leq c_2 \cdot g(n)$$

$$\max(f(n), g(n)) \leq 2(f(n) + g(n))$$

* Lower bound:

Assume $c_1 = \frac{1}{2}$ then.

$$f(n) \geq c_1 \cdot g(n)$$

$$\max(f(n), g(n)) \geq \frac{1}{2}(f(n) + g(n))$$

for all $n_0 = 1$.

Then for all $n \geq n_0$

$$\frac{1}{2}(f(n) + g(n)) \leq \max(f(n), g(n))$$

$$\leq 2 \cdot (f(n) + g(n)).$$

$\therefore T(n) = O(n^2)$

⊞ Growth Function: Growth of functions are primarily used to describe the complexity of algorithms, often in terms of time or space. These functions help us to understand how the performance of an algorithm changes as the size of the input increases.

⊞ Asymptotic Notation: Asymptotic notations are mathematical tools used to represent the time complexity of algorithms for asymptotic analysis.

There are three asymptotic notations:

1. Big O-notation. (O-notation).
2. Omega Notation (Ω-notation)
3. Theta Notation (Θ-notation)

1. Big-O Notation (O-Notation):

→ Big-O notation represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm.

→ It is the most widely used notation for asymptotic analysis.

Examples:

* Linear Search: In linear search, the number of comparisons is directly proportional to the input size.

Number of items	Number of Comparison
10	10
100	100
1 million	1 million
n	n

Running time: O(n).

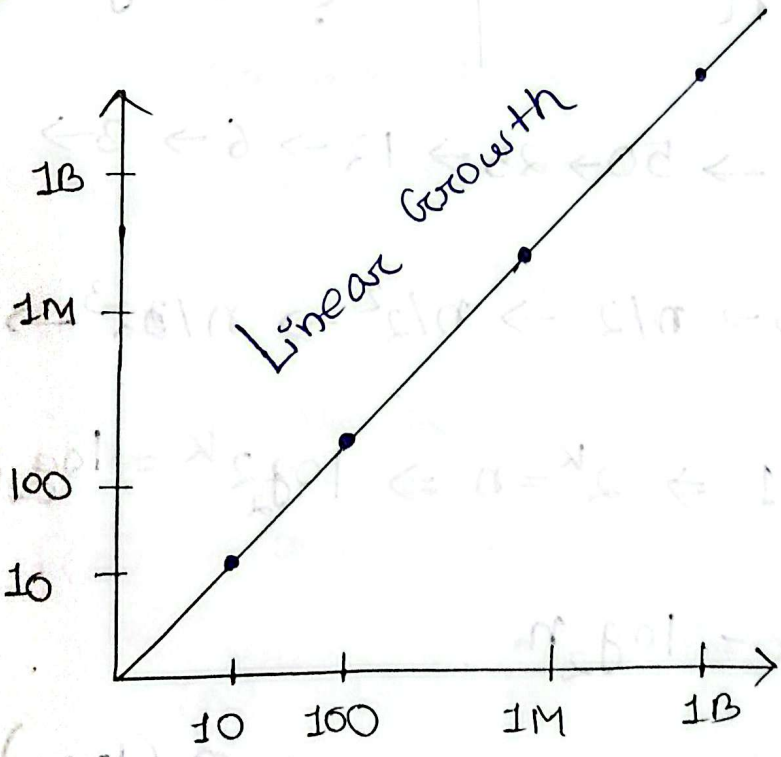


fig: Growth rate of Linear search.

* Binary Search: In binary search, every time half of the list is eliminated.

Number of items	Number of Comparisons
10	4
100	7
1 billion	30
n	? → log n

100 times → 50 → 25 → 12 → 6 → 3 → 1 → 7 Comparison

n items → n/2 → $n/2^2$ → $n/2^3$ → ... $n/2^k$

$$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow \log_2 2^k = \log_2 n$$

$$\Rightarrow k = \log_2 n.$$

∴ Running time : O (log n).

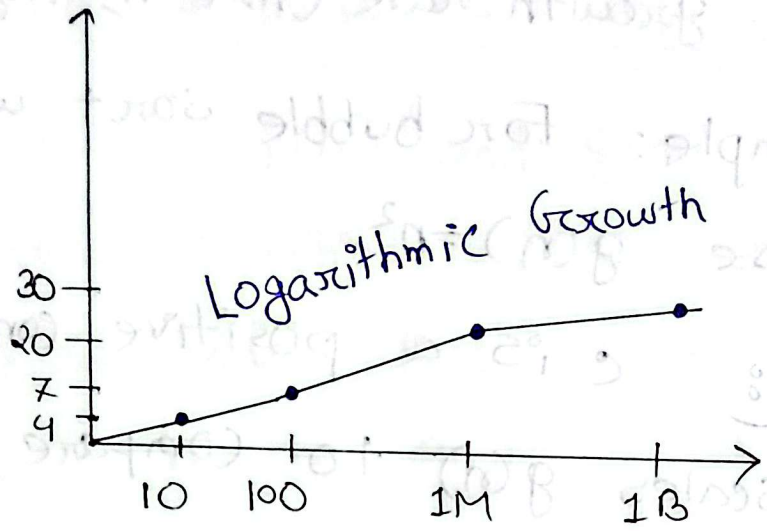


Fig: Growth Rate of Binary Search.

*f(n): f(n) represents the actual time or space complexity of an algorithm as a function of input size n.

→ It could be something like:-

* $f(n) = 3n + 5$ → Linear time
* $f(n) = n^2 + 2n + 1$ → Quadratic time

*g(n): g(n) is the simplified comparison function that describes a

general growth rate (like n, $n \log n$, n^2).

→ Example: For bubble sort, we choose $g(n) = n^2$.

* $c \cdot g(n)$: c is a positive constant that scales $g(n)$ to compare with $f(n)$.

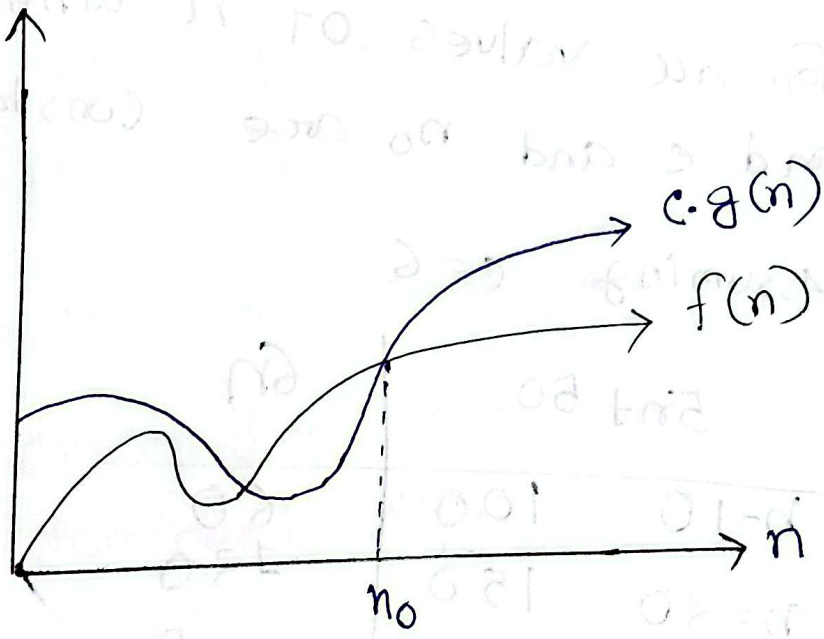
→ $c \cdot g(n)$ is the upper bound (or lower bound, depending on notation) used in asymptotic analysis.

For example:

→ If $f(n) = 3n + 5$, we might say $f(n) \leq c \cdot n$ for some constant $c \geq 4$ and large enough n.

* Formal Definition of Big-O-Notation:

→ Assuming $f(n)$ and $g(n)$ are non-negative functions. $f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n where $n \geq n_0$ and c and n_0 are constants.



→ $g(n)$ is asymptotically bigger than $f(n)$ or, after some point $g(n)$ is always bigger than $f(n)$. $g(n)$ is upper bound of $f(n)$.

Big-O Notation Problems:

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

⇒ According to the definition of Big O notation:

$f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n where $n \geq n_0$ and c and n_0 are constants.

Assuming $c = 6$

	$5n+50$	$6n$
$n=10$	100	60
$n=20$	150	120
$n=50$	300	300
$n=51$	305	306

$5n+50 = O(n)$ for $c = 6$ and $n \geq 50$

$n_0 = 51$

Problem:2 Find the upper bound for $f(n)$

$= n^2 + 10$

⇒ Step-1: Identify the dominant term

	n^2	10
$n=10$	100	10
$n=100$	10000	10
$n=1000$	1000000	10

n^2 is the dominant term.

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

$n^2, n^3, 2^n \ldots$

↳ $g(n)$

Step-3: Apply the Big O definition.

According to the definition of Big O Notation:

$f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n where $n \geq n_0$ and c and n_0 are constants.

Assuming $c = 2$

	n^2+10	$2n^2$
$n=1$	11	2
$n=2$	14	8
$n=3$	19	18
→ $n=4$	26	32

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

Problem: 3 Find the upper bound for $f(n) = 2^n + 3n^3$.

$\Rightarrow$ Dominant term : 2^n Assuming $g(n) = 2^n$

According to the definition of Big O notation:

$f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$

for all values of n where $n \geq n_0$ and c and n_0 are constants.

Assuming $c = 2$

$2^n + 3n^3 \leq 2^{n+1}$

	$2n + 3n^3$	2^{n+1}
$n = 10$	4024	2048
$n = 11$	6041	4096
$n = 14$	24616	32768
$n_0 \rightarrow n = 14$		

$\therefore 2^n + 3n^3 = O(2^n)$ for $c = 2$ and $n_0 = 14$.

Problem:4 Find the upper bound for $f(n) = 300$.

$\Rightarrow$ Dominant term : 300 Assuming $g(n) = 1$

According to the definition of Big O Notation : $f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n where $n \geq n_0$ and c and n_0 are constants.

Assuming $c = 300$

$300 \leq 300$. is true.

$\therefore 300 = O(1)$ for $c = 300$ $n_0 = 1$

2. Big Omega Notations (Ω-notation)

→ Omega Notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

Examples:

* Linear Search: Best case: item is at the first position.

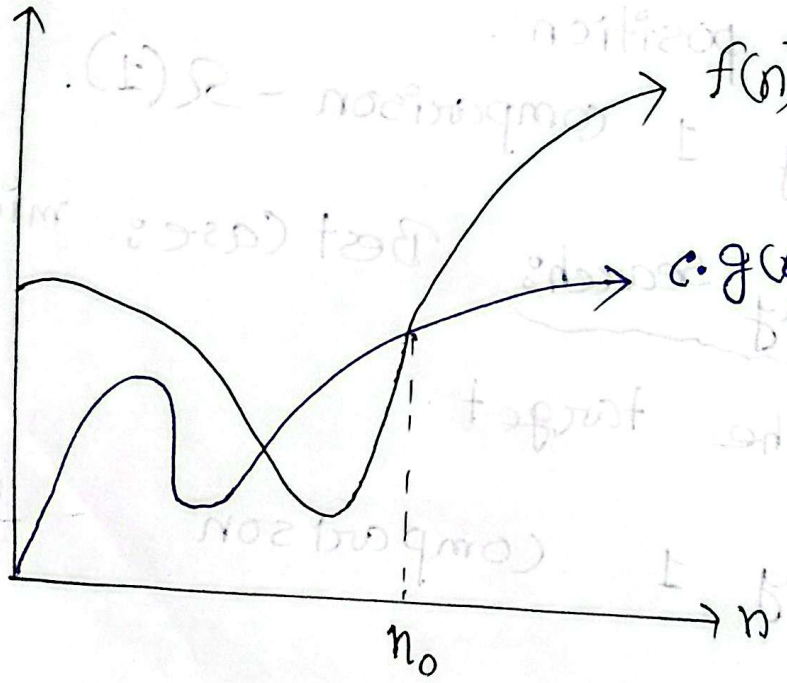
→ Only 1 comparison - $\Omega(1)$.

* Binary Search: Best Case: middle element is the target.

→ Only 1 comparison - $\Omega(1)$.

* Formal Definition of Big-Omega Notation:

→ 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 $n \geq n_0$ and c and n_0 are constants.



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

→ $c \cdot g(n)$ is the lower bound of $f(n)$.

Problem:5 Find the lower bound for $f(n) = 10n^2 + 5$?

⇒ Dominant Term : $10n^2$ Assuming $g(n) = n^2$

According to the definition of Big Omega notation:

$f(n) = \Omega(g(n))$ 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 constants.

Assuming $c = 10$

$10n^2 + 5 \geq 10n^2$ is true for all $n \geq 1$

$\therefore 10n^2 + 5 = \Omega(n^2)$ for $c = 10$ and $n_0 = 1$

3. Big-Theta Notation (Θ-notation):

→ Big Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm it is used for analyzing the average-case complexity of an algorithm.

Example:

* Linear search:

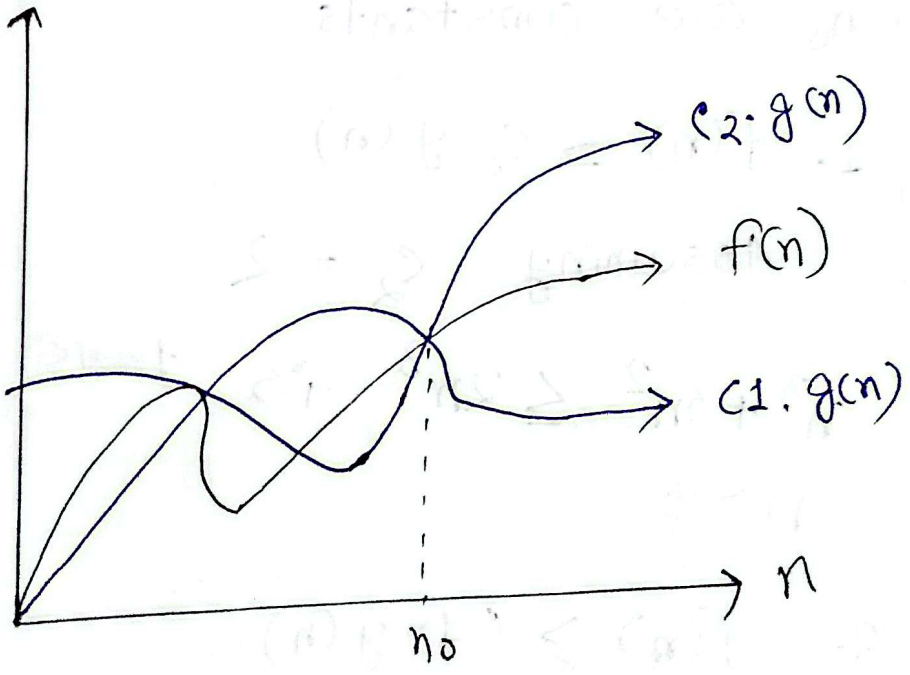
→ Worst case: $O(n)$

→ Best case: $\Omega(1)$

→ But on average, we check half the array - $\Theta(n)$.

Formal Definition of Big-Theta Notation :

→ Assuming $f(n)$ and $g(n)$ are non-negative functions. $f(n) = \Theta(g(n))$ iff $c1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ 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 $f(n)$.

Problem: 6 Show that $f(n) = n^3 + 3n^2 = \Theta(n^3)$.

⇒ According to the definition of Big Theta Notation:

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

1. $f(n) \leq c_2 \cdot g(n)$

Assuming $c_2 = 2$

$n^3 + 3n^2 \leq 2n^3$ is true for $n \geq 3$

2. $f(n) \geq c_1 \cdot g(n)$

Assuming $c_1 = 1$

$n^3 + 3n^2 \geq n^3$ is true for $n \geq 1$.

$\therefore n^3 + 3n^2 = \Theta(n^3)$ for $c_1 = 1$, $c_2 = 2$ and $n_0 = 3$.

Problem:7 Assume that $f(n) = 5n + 50$ and $g(n) = \log_{10} n$. Is $g(n)$ upper bound of $f(n)$?

⇒ According to the definition of Big O notation: $f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n where $n \geq n_0$ and c and n_0 are constants.

Assuming $c = 1000$

	$5n+50$	$1000 \log_{10} n$
$n = 10$	100	1000
$n = 10^2$	550	2000
$n = 10^3$	5050	3000
$n = 10^4$	50050	4000

$\therefore g(n)$ is not an upper bound of $f(n)$.

Problem: 8 Find the upper bound for $f(n) = 3n + 8$.

$\Rightarrow$ The dominant term : $3n$

Assuming $g(n) : n$

According to the Big O- notation:

$f(n) = O(g(n))$ iff $f(n) \leq c \cdot g(n)$ for all values of n where $n \geq n_0$ and c an n_0 are constants

Assuming $c = 4$

$3n + 8 \leq 4n$?