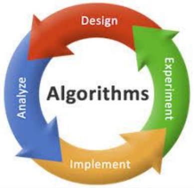
DESIGN AND ANALYSIS OF ALGORITHMS (DAA) (A34EC)

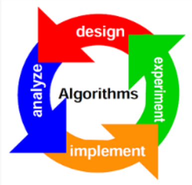
By :-

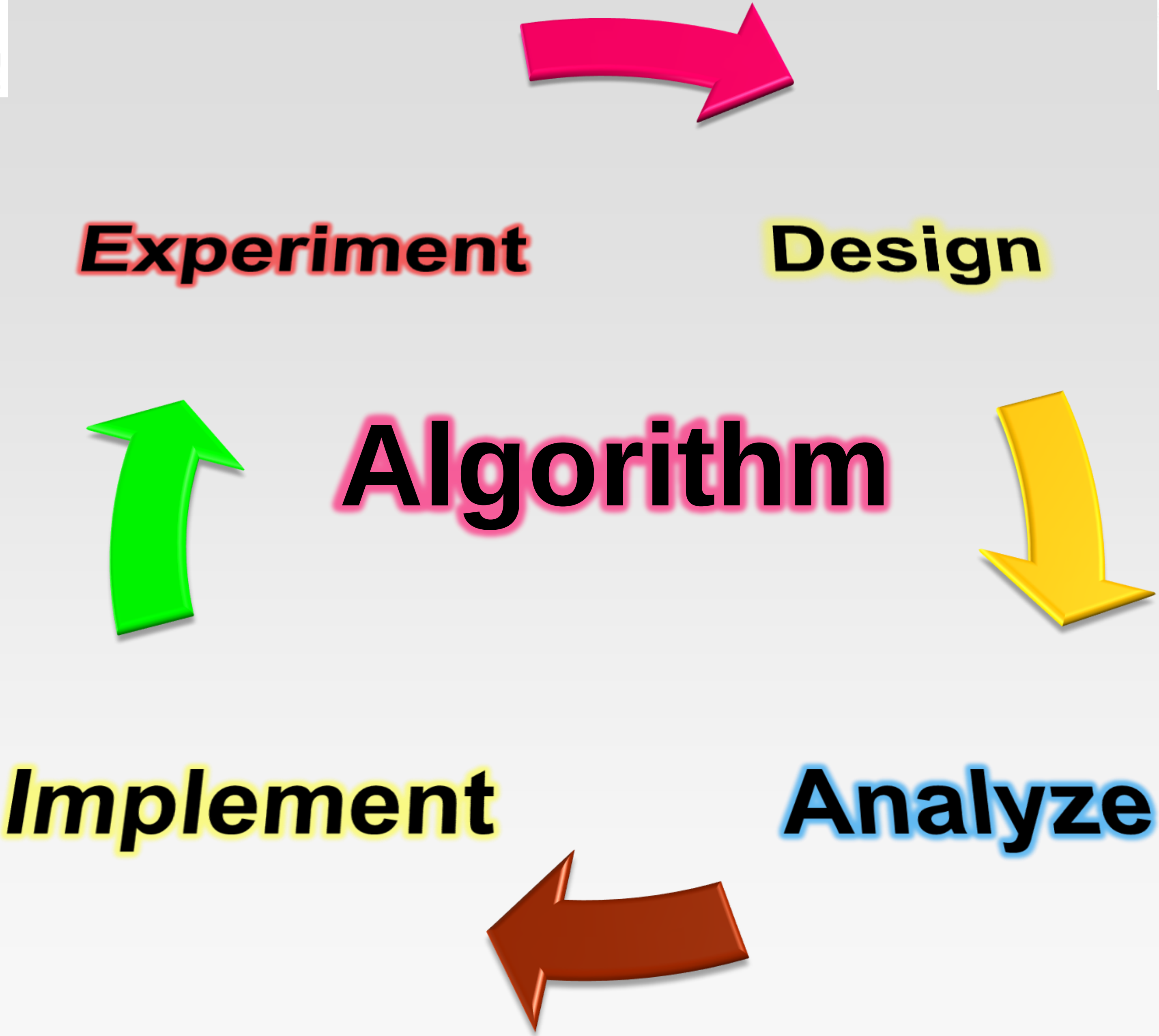
VIJAYKUMAR MANTRI,

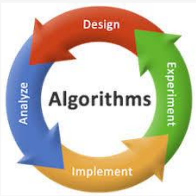
ASSOCIATE PROFESSOR.

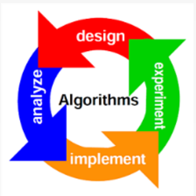
vijay_mantri.it@bvrit.ac.in



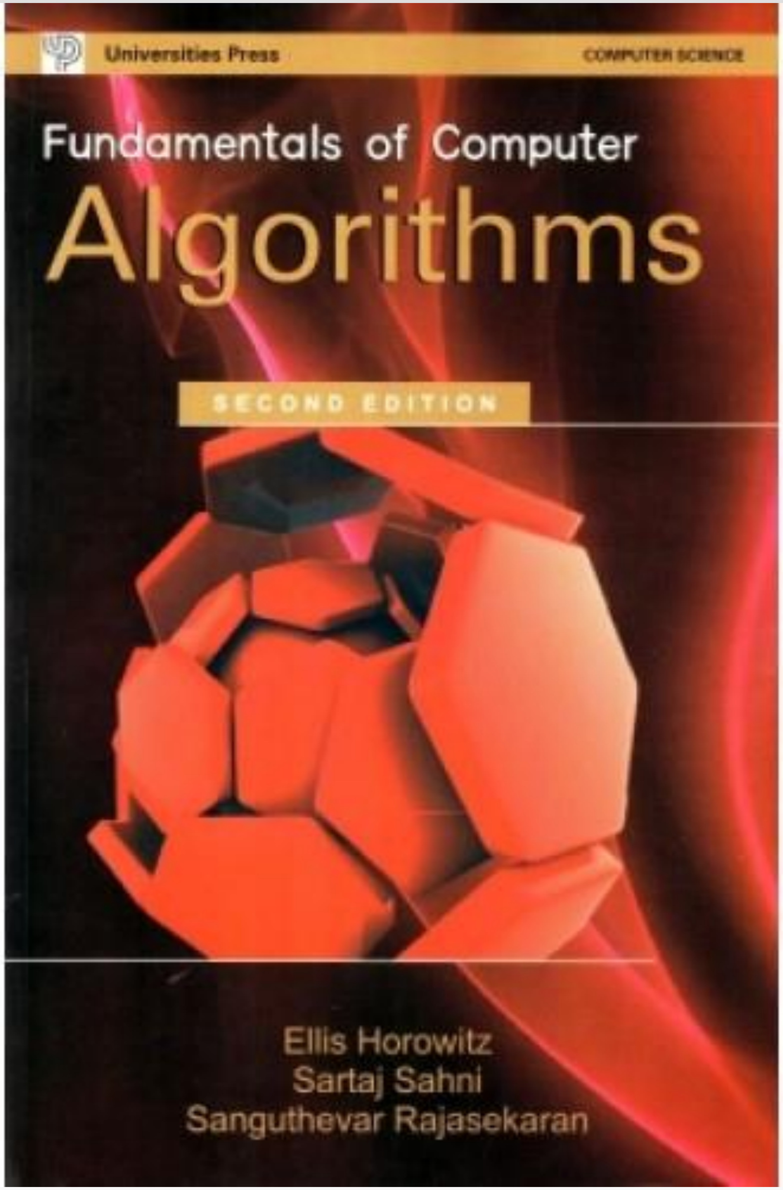



Algorithm
Experiment
Design
Analyze
Implement


Design
Experiment
Implement
Analyze
Algorithms

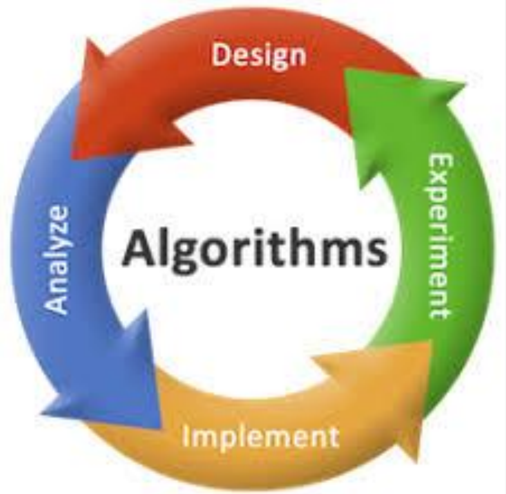

design
experiment
implement
analyze
Algorithms

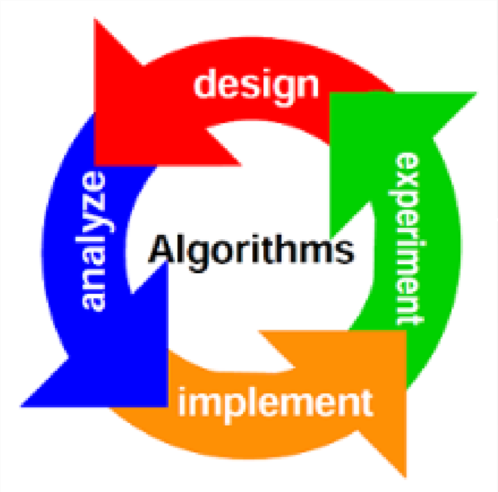
Textbook


Universities Press
COMPUTER SCIENCE
Fundamentals of Computer
Algorithms
SECOND EDITION
Ellis Horowitz
Sartaj Sahni
Sanguthevar Rajasekaran

DAA Unit VI

NP-Hard and NP-Complete Problems





Unit VI Syllabus

NP-Hard and NP-Complete Problems:

- Basic Concepts
- Nondeterministic Algorithms
- NP-Hard and NP-Complete Classes
- Cook’s Theorem.

Basic Concepts

- In this unit, we are going to learn distinction between problems that can be solved by a polynomial time and problems for which no polynomial time algorithm is known.
- It is unexplained phenomenon that for many of the problems we know and study, the best algorithm for their solutions have computing times that cluster into two groups.
- The first group consists of problems whose solution times are bounded by polynomials of small degree like Linear Search $\boldsymbol{O(n)}$, Binary Search $\boldsymbol{O(\log n)}$, Bubble/Insertion sorting $\boldsymbol{O(n^2)}$, Merge Sorting $\boldsymbol{O(n\,log\,n)}$, Matrix Multiplication $\boldsymbol{O(n^3)}$ etc.
- The second group is made up of problems whose best-known algorithms are nonpolynomial (exponential) like Travelling Salesperson $\boldsymbol{O\left(n^2 2^n\right)}$ and 0/1 Knapsack Problem $\boldsymbol{O(2^n)}$, Sum of Subsets $\boldsymbol{O(2^n)}$, Graph coloring $\boldsymbol{O(2^n)}$, Hamiltonian Cycle $\boldsymbol{O\left(n^2 2^n\right), etc}$.

Basic Concepts

- In the quest to develop efficient algorithms, no one has been able to develop a polynomial time algorithm for any problem in second group.
- This is very important because algorithms whose computing times are greater than polynomial (specially time is exponential) very quickly require such vast amount of time to execute that even moderate-size problems cannot be solved.
- Here we are going to show that many of the problems for which there are no known polynomial time algorithms are computationally related.
- In fact, we establish two classes of problems, namely $\boldsymbol{NP-Hard\ and\ NP-Complete}$.

Basic Concepts

- A problem that is $\boldsymbol{NP-Complete}$ has the property that it can be solved in polynomial time if and only if all other $\boldsymbol{NP-Complete}$ problems can also be solved in polynomial time.
- If an $\boldsymbol{NP-Hard}$ problem can be solved in polynomial time, then all $\boldsymbol{NP-Complete}$ problems can be solved in polynomial time.
- All $\boldsymbol{NP-Complete}$ problems are $\boldsymbol{NP-Hard}$, but some $\boldsymbol{NP-Hard}$ problems are not known to be $\boldsymbol{NP-Complete}$.

Nondeterministic Algorithms

- Generally, algorithms has the property that the result of every operation is uniquely defined.
- Algorithms with this property are called **Deterministic Algorithms.**
- Such algorithms represent the programs which can be executed on a computer.
- In a theoretical framework we can remove this restriction on the outcome of every operation.
- We can allow algorithms to contain operations whose outcomes are not uniquely defined but are limited to specified sets of possibilities.
- The machine executing such operations is allowed to choose any one of these outcomes subject to termination condition to be defined later.

Nondeterministic Algorithms

- These types of Algorithms are called **Nondeterministic Algorithms.**
- To specify such algorithms, we introduce three new functions:
  1. Choice(S) – arbitrarily chooses one of the elements of set S.
  2. Failure( ) – indicates an unsuccessful completion.
  3. Success( ) – indicates an successful completion.
- The assignment statement $\boldsymbol{x := Choice(1, n)}$ could result in $\boldsymbol{x}$ being assigned any one of the integers in the range $[\mathbf{1}, \boldsymbol{n}]$.
- Whenever there is a set of choices that leads to a successful completion, then one such set of choices is always made and the algorithm **terminates successfully**.
- A nondeterministic algorithm **terminates unsuccessfully** if and only if there exists no set of choices leading to a success signal.

Nondeterministic Algorithms

- The computing time for Choice, Success and Failure are taken to be $\boldsymbol{O}(\mathbf{1})$.
- A machine capable of executing nondeterministic algorithm is called a nondeterministic machine.
- Although nondeterministic machine do not exists in practice, we see that they provide strong intuitive reasons to conclude that certain problems cannot be solved fast deterministic algorithms.
- **Example 1 - Nondeterministic Search (NSearch)**: Consider the problem of searching for an element $\boldsymbol{x}$ in a given set of elements $\boldsymbol{A}[\mathbf{1}:\boldsymbol{n}], \boldsymbol{n} \geq \mathbf{1}$.
- We have to find an index $\boldsymbol{j}$ such that $\boldsymbol{A}[\boldsymbol{j}] = \boldsymbol{x}\ \boldsymbol{or}\ \boldsymbol{j} = \mathbf{0}\ \boldsymbol{if}\ \boldsymbol{x}$ is not in A.

- A nondeterministic algorithm for this is

```
Algorithm NSearch (Am n, x)
{
j := Choice(1, n); // j is assigned value between 1 and n
  if A[j] = x then
  // By luck, if j is the index of our searching element x
  {
   write (j);   //
 Then display the position of the element
   Success( );         // Non − deterministic algorithm is
  // successfully executed & terminates
  }
   write (0);          // Then display the 0 as position of
   // the element indicating failure.
   Failure( );         // Non − deterministic algorithm is
  // failed and terminates
}
```

- From the way of a nondeterministic computation is defined, it follows that the number **0** can be output if and only if there is no $\boldsymbol{j\ such\ that\ A[j] = x}$.
- Search Algorithm is of nondeterministic complexity $\boldsymbol{O(1)}$.
- Note that since A is not ordered, every deterministic search algorithm is of complexity $\Omega(\boldsymbol{n})$.

- **Example 2 - Nondeterministic Sorting (NSort)** : Let $\boldsymbol{A[i], 1 \le i \le n,}$ be an unsorted array of positive integers. The nondeterministic algorithm $\boldsymbol{NSort(A, n)}$ sorts the numbers into increasing order and then output them.
- An auxiliary array $\boldsymbol{B[1{:}n]}$ is used for convenience.
- The time complexity is $\boldsymbol{O(n)}$.
- Recall that all deterministic sorting algorithms must have a complexity of $\Omega(\boldsymbol{n\ log\ n})$.

- A nondeterministic algorithm for this sorting is

```
Algorithm NSort (A, n)
// Sort n positive integers.
{
  for i := 1 to n do   B[i] := 0; // Initialize array B[ ] with 0
  for i := 1 to n do
  {
    j := Choice(1, n);
// j is assigned a value between 1 and n
    if B[j] ≠ 0 then
      // if j value is already chosen value, then failure
 Failure( );
      B[j] := A[i];
    // Assign B[j] as next element from array A,  A[i]
  }
```

```
for i := 1 to n − 1 do          // Verify the sort order.
    if B[i] > B[i + 1] then
  // If elements chosen not in correct sorted order
 Failure( );  // Algorithm unsuccessfully terminates
   write (B[1: n]);        // If elements are by luck,
  // chosen in correct sorted order, then display.
   Success( );  // Algorithm successfully terminates
}
```

- ***Example***: $n = 4 \qquad A[\ ] = \{5, 9, 7, 2\}$
- ***Successful Case*** : ***Randomly choose*** $j = 2, 4, 3, 1$
  - $B[2] = A[1] = 5$
  - $B[4] = A[2] = 9$
  - $B[3] = A[3] = 7$
  - $B[1] = A[4] = 2$
  - ***Display*** $B[\ ] = \{2, 5, 7, 9\}$

Example 3 - Nondeterministic 0/1 Knapsack Problem :

```
Algorithm NKP (p, w, n, m, r, x)
{
  W := 0;   P := 0; // No objects are chosen yet
  for i := 1 to n do
  {
    x[i] := Choice(0, 1);
  // Object i, is randomly decided to chose or leave
    W := W + x[i] * w[i];
// If Object i is chosen, then add its weight, else add 0
    P := P + x[i] * p[i];          // similarly, If Object i is
   // chosen, then add its profit, else add 0.
  }
```

Nondeterministic Algorithms - Example

$\quad\quad if\ \big((W > m)\ or\ (P < r)\big)\ then$

$//If\ selected\ objects\ weight\ is\ more\ than\ Knapsack$

$//capacity\ or\ selected\ objects\ won't\ give\ us$

$//maximum\ profit\ \ r.$

$\quad Failure(\ );$

$//\ Algorithm\ unsuccessfully\ terminates.$

$\quad\quad else\ \ Success(\ );$

$//\ Otherwise\ Algorithm\ successfully\ terminates.$

$\}$

- The time complexity is $\boldsymbol{O(n)}$.
- If q is the input length using binary representation, the time is $\boldsymbol{O(q)}$.

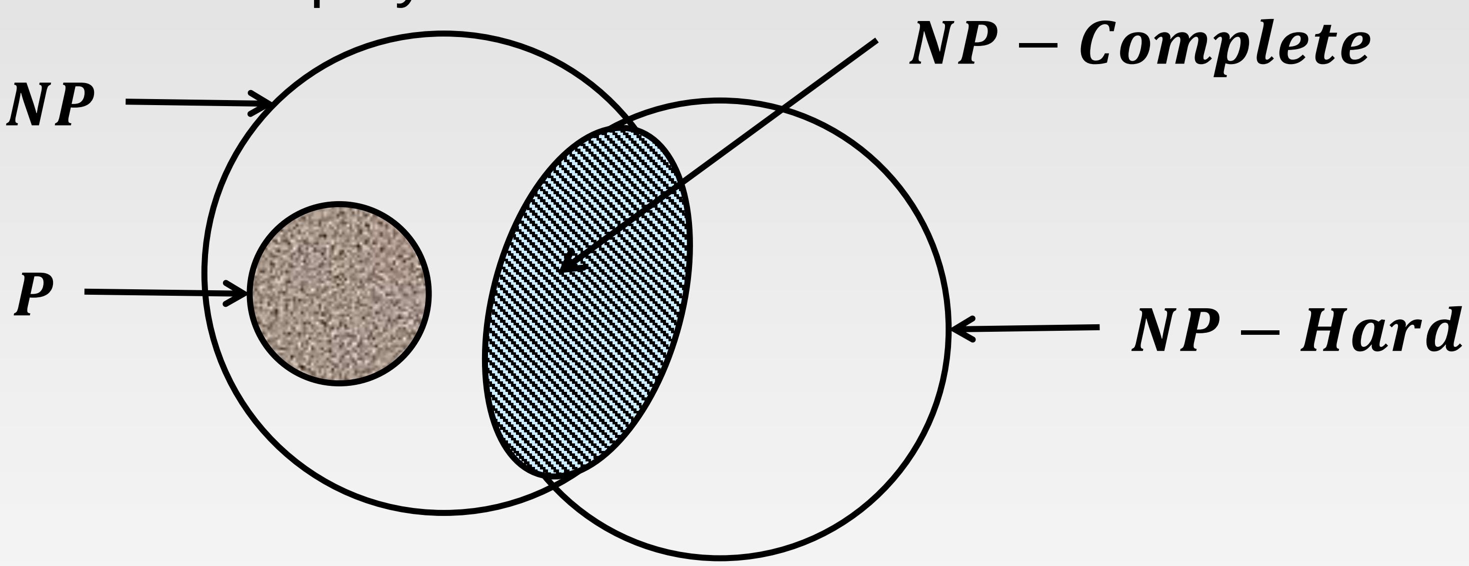
NP-Hard and NP-Complete Classes

Definition $\boldsymbol{P,\ NP,\ NP-Hard\ and\ NP-Complete}$

- In measuring the complexity of an algorithm, we use the input length as parameter.
- An algorithm $\boldsymbol{A\ is\ of\ Polynomial\ Complexity}$ if there exists a polynomial $\boldsymbol{p}(\ )$ such that the computing time of $\boldsymbol{A\ is\ O(p(n))}$ for every input of size $\boldsymbol{n}$.
- **P** is the set of all decision problems solvable by deterministic algorithms in polynomial time.
- **NP** is the set of all decision problems verifiable by Nondeterministic Algorithms in polynomial time.
- If a problem & solution is given, algorithm should be able to tell correct or incorrect in polynomial time.
- Since deterministic algorithms are just a special case of nondeterministic, we conclude that $\boldsymbol{P \subseteq NP}$.

NP-Hard and NP-Complete Classes

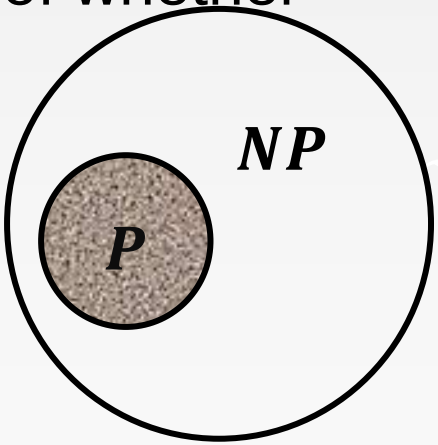
- A problem is $\boldsymbol{NP-Hard}$, if any problem (in NP) can be reduced to NP Problem in polynomial time.
- A problem is $\boldsymbol{NP-Complete}$, if any problem (in NP) can be reduced to it in polynomial time and it is also in NP.



- This means, the set of $\boldsymbol{NP-Complete}$ problems is the set formed by intersect of $\boldsymbol{NP\ and\ NP-Hard}$.
- The main ambiguity in computer science field is whether $\boldsymbol{P = NP\ or\ P \neq NP}$.

NP-Hard and NP-Complete Classes

- It is possible that for all problems in NP, there exists polynomial time deterministic algorithms that have remained undiscovered?
- This seems unlikely, at least because of the tremendous effort that has already been extended by so many people on these problems.
- A proof that $P \neq NP$ is just as elusive and seems to require as yet undiscovered techniques.
- But as with many famous unsolved problems, they serve to generate other useful results, and the question of whether $NP \subseteq P$ is no exception.
- The figure displays the relationship between P and NP assuming $P \neq NP$.



Cook's Theorem

- S. Cook formulated the following question : Is there any single problem in $\boldsymbol{NP}$ such that if we showed it to be in P, then that would imply that $\boldsymbol{P = NP}$?
- Cook answered his own question in the affirmative with the following theorem, known as Cook's Theorem.
- **Cook's Theorem :** Satisfiability is in $\boldsymbol{P}$ if and only if $\boldsymbol{P = NP}$
- $\boldsymbol{NP - Hard\ and\ NP - Complete}$: A Problem L is in $\boldsymbol{NP - Hard}$, if and only if satisfiability reduces to L **(satisfiability** $\propto \boldsymbol{L}$)
- A problem L is $\boldsymbol{NP - Complete}$ if and only if L is $\boldsymbol{NP - Hard}$ and $\boldsymbol{L \in NP}$.
- The satisfiability problem is determine whether a formula is true for some assignment of truth values to the variables.

Cook's Theorem

- **Satisfiability : Nondeterministic approach**
- Consider a problem with 4 variables $x_1, x_2, x_3, and\ x_4$
- Let conjunctive normal form of the problem is

$$E = (x_1 \vee \bar{x}_3) \wedge (\bar{x}_2 \vee x_3 \vee \bar{x}_4) \wedge (x_2 \vee \bar{x}_4)$$

- This function E, will be true if and only if every ax term is true.
- The expression E is valid only when if it is true for all true values of its variables $x_1, x_2, x_3, and\ x_4$.
- This can be written using Nondeterministic Algorithm

Nondeterministic Satisfiability Algorithm :

```
Algorithm Eval (E, n)
//Determine whether the propositional formula E
//is Satisfiable  the variables are  x1, x2,  x3, ...., xn.
{
    for i := 1 to n do
    // Choose a truth value assignment.
        x[i] := Choice(false, true);
   // Variable x[i], is randomly chosen 0 or 1
   if (E(x1, x2,  x3, ...., xn) = 1) then // If chosen values
// of all variables x[ ] satisfies the given Expression E
    Success( );   // Algorithm successfully terminates.
        else
    Failure( );
    // Otherwise,  Algorithm unsuccessfully terminates
 }
```

Nondeterministic Satisfiability Algorithm

- The above algorithm chooses one of the $\mathbf{2}^{\boldsymbol{n}}$ possible assignments of truth values to $(\boldsymbol{x_1}, \boldsymbol{x_2},\ \boldsymbol{x_3}, \ldots., \boldsymbol{x_n})$ and verifies that $\boldsymbol{E}(\boldsymbol{x_1}, \boldsymbol{x_2},\ \boldsymbol{x_3}, \ldots., \boldsymbol{x_n})$ is true of not for that assignment in nondeterministic approach.

Best of Luck for Your Mid and Final Examinations