14/05/22

ASSIGNMENT-1.

CEN601: ANALYSIS & DESIGN OF ALGORITHMS.

YASH VINAYVANSHI
19BCS081.

Q1 (i) Big oh (O-Notation)

→ Binds a function from Above – Asymptotic upper bound

$O(g(n)) = \{f(n) : \exists c, n_0 > 0 \text{ s.t. } 0 \le f(n) \le cg(n) \ \forall n \ge n_0\}$.

→ Since O Notn describes the upper bound, its used to bound the worst case running time of an algorithm.

(ii) Ω Notation

→ Binds a function from below – Asymptotic lower bound.

$\Omega(g(n)) = \{f(n) : \exists c, n_0 > 0 \text{ s.t. } 0 \le cg(n) \le f(n) \ \forall n \ge n_0\}$.

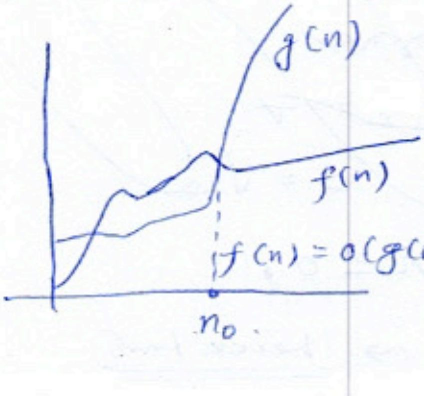
→ Its used to bind the best case running time of an algorithm.

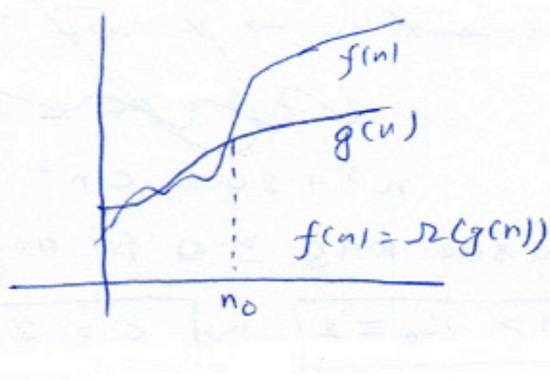
(iii) Θ Notation

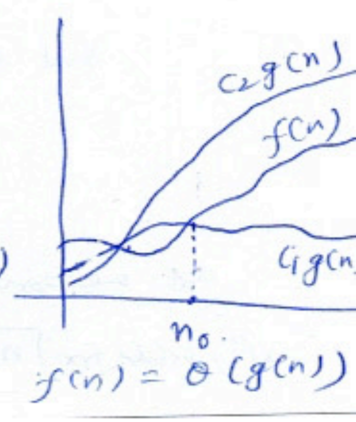
→ Asymptotically binds a function from above & below: sandwiching.

$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0 \text{ s.t. } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \ \forall n \ge n_0\}$

→ Its used to bind the average case running time of an algorithm.







(i) To prove $T(n) = n^3 + 20n + 1$ is $O(n^3)$

Acc. to definition of $O(g(n))$, $T(n) \le c\,g(n)$ for all $n \ge n_0$, $c, n_0 > 0$.

$$n^3 + 20n + 1 \le cn^3 \quad \text{— (1)}$$

$$\frac{n^3}{n^3} + \frac{20n}{n^3} + \frac{1}{n^3} \le c.$$

$$1 + \frac{20}{n} + \frac{1}{n^3} \le c.$$

~~for $n \ge n_0 = 1$, $f(n) = 1 + \frac{20}{1} + \frac{1}{1} = 22$.~~

$$f(n) = 1 + \frac{20}{n} + \frac{1}{n^3} \rightsquigarrow f'(n) = \frac{-20}{n^2} - \frac{3}{n^4}$$

$$f'(n) = 0 \rightarrow \frac{-20}{n^2} - \frac{3}{n^4} = 0$$ No global Minima.

$\rightsquigarrow$ (1) holds for $\boxed{n \ge n_0 = 1}$, $f(n) = 1 + \frac{20}{1} + \frac{1}{1} = 22$

$\boxed{c \ge 22}$

$\rightarrow$ hence proved.

(ii) To prove $T(n) = n^3 + 20n$ is $\Omega(n^2)$

Acc. to definition of $\Theta(g(n))$, $T(n) = \Omega(g(n))$ if $T(n) \ge c\,g(n)$ for all $n \ge n_0$, $c, n_0 > 0$

$$n^3 + 20n \ge cn^2. \quad \text{— (1)}$$

$$\frac{n^3}{n^2} + \frac{20n}{n^2} \ge c.$$

$$n + \frac{20}{n} \ge c.$$

~~$f(x) = x + \frac{20}{x} \rightsquigarrow f'(x) = 1 - \frac{20}{x^2}$~~

~~$f'(x) = 0$ for local extrema $\rightsquigarrow 1 - \frac{20}{x^2} = 0 \rightarrow x^2 = 20$ $\rightsquigarrow x = \pm\sqrt{20}$~~

~~but since $x > 0 \rightarrow x = \sqrt{20}$.~~

~~$\therefore n^3 + 20 \ge 2\sqrt{20}\,n^2$~~

~~$\therefore n^3 + 20 \ge cn^2$ for $c = \sqrt{20}$~~

as we can see RHS ≥ 0 for any $n \ge 0$,

$\rightsquigarrow$ (1) holds for $\boxed{n \ge n_0 = 1}$ $\rightsquigarrow$ $\boxed{c = 21}$ $\rightsquigarrow$ hence proved

Q2 (i) Recurrence Equation for running time $T(n)$ of Strassen's algorithm for matrix multiplication.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1. \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

using Master's theorem,

$T(n) = aT(n/b) + f(n) \rightsquigarrow a = 7,\ b = 2\ f(n) = n^2$

$n^2 = n^{\log_2 7 - \varepsilon} \rightsquigarrow \varepsilon > 0$ ($\log_2 7 > 2$)

case 1: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$

(ii) Strassen's Algorithm for matrix Multiplication.

```
MatMul-Strassen(A, B)
    n = A.rows.
    Let C be a n x n matrix.
    if n == 1.
        C(n, n) = A(n, n) . B(n, n).
    else
        S1  = B(1..n/2, n/2+1..n) - B(n/2+1...n, n/2+1..n)
        S2  = A(1..n/2, 1..n/2) + A(1..n/2, n/2+1...n).
        S3  = A(n/2+1..n, 1..n/2) + A(n/2+1..n, n/2+1..n).
        S4  = B(n/2+1..n, 1..n/2) - B(1..n/2, 1..n/2).
        S5  = A(1..n/2, 1..n/2) + A(n/2+1..n, n/2+1..n)
        S6  = B(1..n/2, 1..n/2) + B(n/2+1..n, n/2+1..n).
        S7  = A(1..n/2, n/2+1..n) - A(n/2+1..n, n/2+1..n)
        S8  = B(n/2+1..n, 1..n/2) + B(n/2+1..n, n/2+1..n).
        S9  = A(1..n/2, 1..n/2) - A(n/2+1..n, 1..n/2)
        S10 = B(1..n/2, n/2+1..n) + B(1..n/2, 1..n/2)
        P1 = MatMul-Strassen (A(1..n/2, 1..n/2), S1)
        P2 = MatMul-Strassen (S2, B(n/2+1..n, n/2+1..n))
        P3 = MatMul-Strassen (S3, B(1..n/2, 1..n/2))
        P4 = MatMul-Strassen (A(n/2+1..n, n/2+1..n), S4)
        P5 = MatMul-Strassen (S5, S6)
        P6 = MatMul-Strassen (S7, S8)
        P7 = MatMul-Strassen (S9, S10)
        C (1..n/2, 1..n/2) = P5 + P4 - P2 + P6.
        C (1..n/2, n/2+1..n) = P1 + P2
        C (n/2+1..n, 1..n/2) = P3 + P4.
        C (n/2+1..n, n/2+1..n) = P5 + P1 - P3 - P7.
    return C
```

$$A = \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix} \qquad B = \begin{bmatrix} 7 & 8 \\ 8 & 9 \end{bmatrix}$$

$A_{11} = 3$ $\qquad B_{11} = 7$

$A_{12} = 4$ $\qquad B_{12} = 8$

$A_{21} = 5$ $\qquad B_{21} = 8$

$A_{22} = 6$ $\qquad B_{22} = 9$.

$S_1 = B_{12} - B_{22} = 8 - 9 = -1$

$S_2 = A_{11} + A_{12} = 3 + 4 = 7$

$S_3 = A_{21} + A_{22} = 5 + 6 = 11$

$S_4 = B_{21} - B_{11} = 8 - 7 = 1$

$S_5 = A_{11} + A_{22} = 3 + 6 = 9$

$S_6 = B_{11} + B_{22} = 7 + 9 = 16$

$S_7 = A_{12} - A_{22} = 4 - 6 = -2$

$S_8 = B_{21} + B_{22} = 8 + 9 = 17$

$S_9 = A_{11} - A_{21} = 3 - 5 = -2$

$S_{10} = B_{11} + B_{12} = 7 + 8 = 15$.

$P_1 = A_{11} \times S_1 \cancel{= A_{11} \times B_{12} - A_{11} \times B_{22}} = 3 \times (-1) = -3$

$P_2 = S_2 \times B_{22} = 7 \times 9 = 63$

$P_3 = S_3 \times B_{11} = 11 \times 7 = 77$

$P_4 = A_{22} \times S_4 = 6 \times 1 = 6$

$P_5 = S_5 \times S_6 = 9 \times 16 = 144$

$P_6 = S_7 \times S_8 = -2 \times 17 = -34$

$P_7 = S_9 \times S_{10} = -2 \times 15 = -30$.

$C_{11} = P_5 + P_4 - P_2 + P_6 = 144 + 6 - 63 - 34 = 53$

$C_{12} = P_1 + P_2 = -3 + 63 = 60$

$C_{21} = P_3 + P_4 = 77 + 6 = 83$

$C_{22} = P_5 + P_1 - P_3 - P_7 = 144 - 3 - 77 + 30 = 94$.

$$\therefore \quad \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 8 & 9 \end{bmatrix} = \begin{bmatrix} 53 & 60 \\ 83 & 94 \end{bmatrix}$$

Q 3. Optimal Substructure of Fractional knapsac pblm

Assume that $\underline{X}$ is the optimal solution with value $\underline{V}$ to problem $\underline{S}$ with knapsac capacity $\underline{W}$.

To prove : $X' = X - x_j$ is an optimal solution to subproblem $S' = S - \{j\}$ and the knapsac capacity $W' = W - w_j$.

proof (by contradiction).

Assume X' is not optimal to S' and we've another solution X'' to S' that has higher total value $V'' > V$, then $X'' \cup \{x_j\}$ is a soln with value $V'' + v_j > V' + v_j = V$. This is a contradiction because V is assumed to be optimal in the beginning.

Greedy choice property of fractional knapsac pblm.

Greedy choice prop : Let j be the item with maximum v_i / w_i, then $\exists$ an optimal soln in which you take as much fract of item j as possible.

proof (by contradiction)

- Suppose there exists an optimal solution in which we didn't take as much of item j as possible.
- If the knapsac is not full, add some more ~~item~~ of item j, and we've a higher value soln. (Contradiction)
- we thus assume knapsac is full.
- there must exist some item $k \neq j$ with $\frac{v_k}{w_k} < \frac{v_j}{w_j}$ that's in the knapsac.
- we also must have that not all of j is in knapsac.
- we can therefore take a piece of k, with wt - ϵ out of knapsack & put a piece of j with ϵ weight in knapsack.
- This increases knapsac value by

$$\epsilon \times \frac{v_i}{w_j} - \epsilon \frac{v_k}{w_k} > 0.$$

which is a contradiction to original soln being optimal.

$\therefore$ we can use greedy strategy to design an algorithm for fractional knapsac.

GreedyFractionalKnapsac (v, W, capacity)

```
wttaken = 0
profit = 0
sort items I in decreasing order of v/w        O(nlogn
for each item i in sorted list.                O(n)
    if (wt-taken + w_i ≤ capacity)             O(
        wttaken += w_i
        profit += v_i
    else.                                      O(1)
        remaining = capacity - wt-taken
        profit += remaining * (vi/wi).
        break
return profit
                                               O(1)
                                            T = O(nlogn)
```

I :	1	2	3	4	5	6	7.
W :	2	3	5	7	1	4	1.
V :	10	5	15	7	6	18	3
V/W :	5	1.67	3	1	6	4.5	3.

sorted by V/W

V/w :	6	5	4.5	3	3	1.67	1.
I :	5	1	6	3	7	2	4.
W :	1	2	4	5	1	3	7
V :	6	10	18	15	3	5	7

			1+2	3+4	7+5	12+1
wt taken	0	1	3	7	12	13
profit	0	6	16.	34.	49	52
items taken	{}.	{I5}.	{I5, I1}	{I5, I1, I6}	{I5, I1, I6, I3}	{I5, I1, I6, I7}.

~~13+3 = 16 > capacity = 15.~~

~~$52 + (15-13)\left(\frac{5}{3}\right) = \boxed{53.33}$~~

~~$\{I_5, I_1, I_6, I_3, I_7, \frac{5}{3} \times I_2\}$~~

13 + 3 = 16 > capacity = 15

13 + 2 = 15

$52 + (15-13)\left(\frac{5}{3}\right) = \boxed{55.33}$

$\{ I_5, I_1, I_6, I_3, I_7 \times \frac{5}{3} I_2 \}$

Maximum profit = 55.33 with items I_5, I_1, I_6, I_3, I_7, $5/3\ I_2$ in knapsac.

Q4.

Definitions

Subsequence: Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$, another sequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ is a subsequence of X if there exists a strictly increasing sequence $\langle i_1, i_2, \ldots, i_k \rangle$ of indices in X s.t for all $j = 1, 2, \ldots k$ we have $x_{i_j} = z_j$.

eg. $Z = \langle B, C, D, B \rangle$ is a subsequence of
$X = \langle A, B, C, B, D, A, B \rangle$ with index seq $\langle 2, 3, 5, 7 \rangle$
1 2 3 4 5 6 7.

Common subsequence: A sequence Z is a common subseq. of X & Y if Z is a subsequence of both X & Y.

eg $X = \langle A, B, C, B, D, A, B \rangle$
$Y = \langle B, D, C, A, B, A \rangle$
$Z = \langle B, C, A \rangle$ is a subseq of X & Y.

Problem statement

LCS Pblm: Given two sequences $X = \langle x_1, x_2, \ldots x_m \rangle$ and $Y = \langle y_1, y_2, \ldots y_n \rangle$, find the maximum length subsequence of X & Y.

Problem characterization

(1) LCS has optimal substructure property.

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ &
$Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences
& $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be LCS of X & Y.

1. if $x_m = y_n$ then $z_k = x_m = y_n$ & z_{k-1} is LCS of x_{m-1} & y_{n-1}.

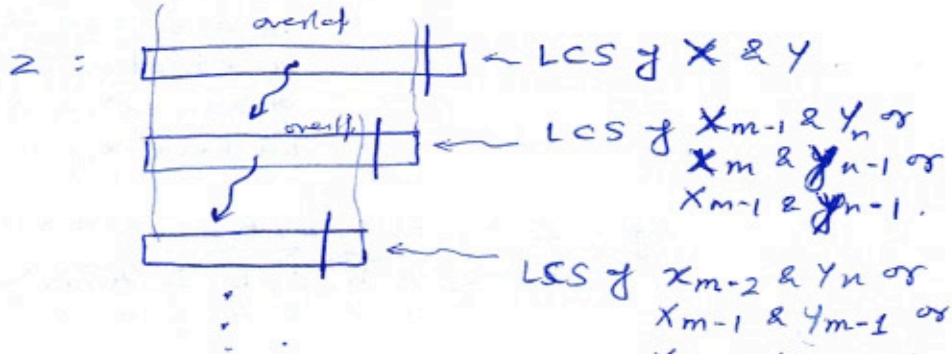
X : $\underline{\quad x_{m-1} \quad}$ A
Y : $\underline{\quad y_{n-1} \quad}$ A
Z : $\underline{\quad z_{k-1} \quad}$ A

2. if $x_m \neq y_n$, then $z_k \neq x_m \Rightarrow$ Z is LCS of X_{m-1} & Y
3. if $x_m \neq y_n$, then $z_k \neq y_n \Rightarrow$ Z is LCS of X & Y_{n-1}

X : $\underline{\quad x_{m-1} \quad}$ A
Y : $\underline{\qquad\qquad}$ B
Z : $\underline{\qquad\qquad}$ B

X : $\underline{\qquad\qquad}$ A
Y : $\underline{\quad y_{n-1} \quad}$ B
Z : $\underline{\qquad\qquad}$ A

(2) LCS has overlapping subproblems property.
LCS of two sequences contains within it the LCS of prefixes of the two sequences.



∴ LCS has only $\Theta(mn)$ distinct subproblems. (m = length of X, n = length of Y)
Here we can apply 2 dimensional bottom up DP to solve the problem efficiently

$\boxed{\text{Dynamic Programming Algo for LCS}}$

```
LCS-Length(X, Y)
   m = X.length
   n = Y.length
   let b[1..m, 1..n] & c[0..m ; 0..n] be new table
   for i = 1 to m
         c[i, 0] = 0.
   for j = 0 to n.
         c[0, j] = 0
   for i = 1 to m.
      for j = 1 to n.
            if xi == yj
                 c[i, j] = c[i-1, j-1] + 1
                 b[i, j] = "↘"
            else if c[i-1, j] ≥ c[i, j-1].
                 c[i, j] = c[i-1, j].
                 b[i, j] = "↓"
            else c[i, j] = c[i, j-1].
                 b[i, j] = "→"
   return c & b.
```

Initial call is
PRINT-LCS(b, X, X.length, Y.length)

```
PRINT-LCS (b, X, i, j).
    if i == 0 or j == 0.
        return
    if b[i, j] = "↘"
        PRINT-LCS(b, X, i-1, j-1).
        print x_i
    else if b[i, j] == "↓".
        PRINT-LCS(b, X, i-1, j).
    else PRINT-LCS (b, X, i, j-1).
```

(i) X = ABCDBCDC
Y = BCDCD.

	j	0	1	2	3	4	5
i		y_j	B	C	D	C	D
0	x_i	0	0	0	0	0	0
1	A	0	A≠B 0	A≠C 0	A≠D 0	A≠C 0	A≠D 0
2	B	0	B=B 1	B≠C 1	B≠D 1	B≠C 1	B≠D 1
3	C	0	C≠B 1	C=C 2	C≠D 2	C=C 2	C≠D 2
4	D	0	D≠B 1	D≠C 2	D=D 3	D≠C 3	D=D 3
5	B	0	B=B 1	B≠C 2	B≠D 3	B≠C 3	B≠D 3
6	C	0	C≠B 1	C=C 2	C≠D 3	C=C 4	C≠D 4
7	D	0	D≠B 1	D≠C 2	D=D 3	D≠C 4	D=D 5
8	C	0	C≠B 1	C=C 2	C≠D 3	C=C 4	C≠D 5

c

		1	2	3	4	5
		B	C	D	C	D
1	A	↓	↓	↓	↓	↓
2	B	↘	→	→	→	→
3	C	↓	↘	→	↘	→
4	D	↓	↓	↘	→	↘
5	B	↘	↓	↓	↓	↓
6	C	↓	↘	↓	↘	→
7	D	↓	↓	↘	↓	↘
8	C	↓	↘	↓	↘	↓

b

X : ABCDBCDC
Y : BCDCD

∴ LCS = BCDCD.

PL(b, X, 8, 5)
↓
PL(b, X, 7, 5)
↓
PL(b, X, 6, 4) → X_5 = 'D'
↓
PL(b, X, 5, 3) → X_4 = 'C'
↓
PL(b, X, 4, 3)
↓
PL(b, X, 3, 2) → X_3 = 'D'
↓
PL(b, X, 2, 1) → X_2 = 'C'
↓
PL(b, X, 1, 0) → X_1 = 'B'
↳ return

X = " POLYNOMIAL "
Y = " EXPONENTIAL "

i		j: 0	1	2	3	4	5	6	7	8	9	10	11
	y_j		E	X	P	O	N	E	N	T	I	A	L
0	x_i	0	0	0	0	0	0	0	0	0	0	0	0
1	P	0	0	0	P=P 1	1	1	1	1	1	1	1	1
2	O	0	0	0	1	O=O 2	2	2	2	2	2	2	2
3	L	0	0	0	1	2	2	2	2	2	2	2	3
4	Y	0	0	0	1	2	2	2	2	2	2	2	3
5	N	0	0	0	1	2	3	3	3	3	3	3	3
6	O	0	0	0	1	2	3	3	3	3	3	3	3
7	M	0	0	0	1	2	3	3	3	3	3	3	3
8	I	0	0	0	1	2	3	3	3	3	4	4	4
9	A	0	0	0	1	2	3	3	3	3	4	5	5
10	L	0	0	0	1	2	3	3	3	3	4	5	6

	E	X	P	O	N	E	N	T	I	A	L
P	↓	↓	↘	→	→	→	→	→	→	→	→
O	↓	↓	↓	↘	→	→	→	→	→	→	→
L	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↘
Y	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
N	↓	↓	↓	↓	↘	→	↘	→	→	→	→
O	↓	↓	↓	↘	↓	↓	↓	↓	↓	↓	↓
M	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
I	↓	↓	↓	↓	↓	↓	↓	↓	↘	→	→
A	↓	↓	↓	↓	↓	↓	↓	↓	↓	↘	→
L	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↘

X : POLYNOMIAL
Y : EXPONENTIAL
LCS : PONIAL

Q5. NP-class

Set of all decision problems solved by a non deterministic machine in polynomial time

or.

It is a complexity class that represents set of all decision problems where the answer is 'yes' have proofs that can be verified in polynomial time.

This means, if someone gives us an instance of the problem and a certificate (sometimes called witness) to the answer being yes, we can check that it is correct in polynomial time

eg. INTEGER FACTORIZATION.

pblm: given integers n & m, is there an integer f with $1<f<m$ such that f divides n?

This is a decision pblm bec answers are yes or no. If someone hands us an instance of the pblm (ie n & m) and an integer f s.t $1<f<m$, and claim that f is a factor of n (the certificate), we can check the answer in polynomial time by performing the division n/f

NP-complete

NPC is a complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP pblm Y to X in polynomial time.

This means that we can solve Y quickly if we know how to solve X quickly. Y is reducible to X, if there is a polynomial time algorithm f to transform instances y of Y to instances $x=f(y)$ of X in polynomial time, with the property that the answer to y is yes, iff the answer to f(y) is yes.

eg. 3SAT (a Boolean satisfiability pblm)

pblm : given a conjunction (ANDs) of 3-clause disjunctions (ORs), ie stmt of form

$(v_{11}$ OR v_{21} OR $v_{31})$ AND
$(v_{12}$ OR v_{22} OR $v_{32})$ AND
... AND
$(v_{1n}$ OR v_{2n} OR $v_{3n})$

Where v_{ij} is a boolean variable or negation of a variable from a finite predefined list $(x_1, x_2, \ldots x_n)$

It can be shown that every NP problem can be reduced to 3SAT: Cook's theorem.

NPC pblms are important because if a deterministic polynomial time algorithm is found to solve any of them, then every NP pblm is solvable in polynomial time.

NP-Hard

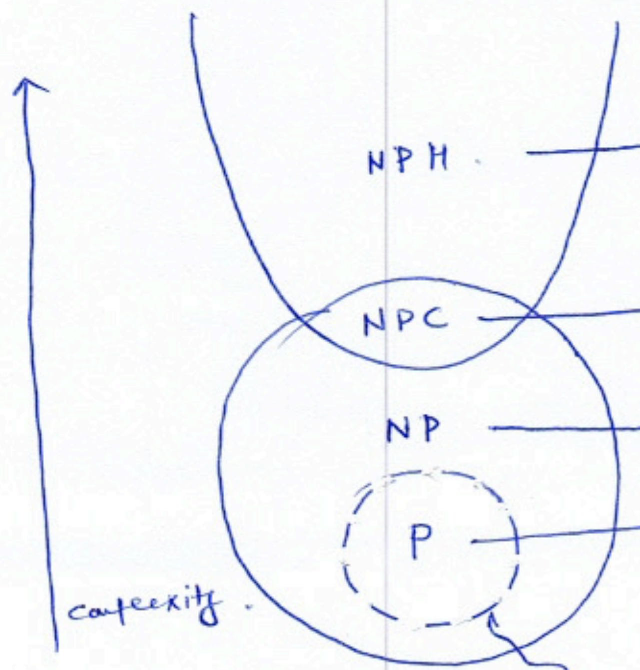
These are the problems which are at least as hard as the NP-complete problems. All NPH pblms do not have to be NP, and they do not have to be decision problems.

~~The precise definition is, a pblm is~~

A problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time.

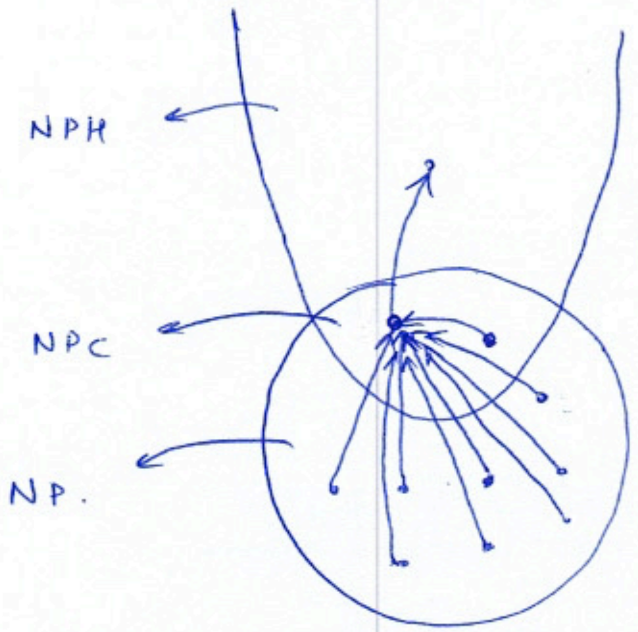
But since any NPC pblm can be reduced to any other NPC in polynomial time, all NPC pblms can be reduced to any NPH problem in polynomial time. Then, if there is a soln to one NPH pblm in polynomial time, there is a soln to all NP pblms in polynomial time.

eg. HALTING PROBLEM
pblm : given a program P and input I, will it halt?



	verifiable in polynomial time?	solvable in polynomial time?
NPH	if in NPC ~~yes~~/No	unknown
NPC	yes.	unknown
NP	yes.	No/~~yes~~ if in P
P	yes	yes.

porous boundary bec. over the years many NP pblms are proven to be in P (eg. primality testing)



$i \rightarrow j$ indicates i is reducible to j in polynomial time.