UNIT-V DAA

Introduction to Randomization Algorithm:

Probabilistic Analysis

It uses probability for the analysis of our algorithms.

The Hiring Problem.

→ Manager wants to hire an assitant.
- Employement agency - to do the job.

Strategy
- agency will send 1 candidate each day.
- If the current assistant
  - If the candidate is better than the current assistant, hire the candidate.

Algorithm:

```
HIRE(N)
    best = 0        // dummy least qualified.
    for i=1 to N
        if candidate i is better than best
            best = i
```

hire candidate i

low cost - interviewing candidate c_i

Expensive - hiring c_h

~~Total cost.~~

we have n candidates, we hire m of them.

Total cost $= O(nc_i + mc_h)$

worst case

→ If the candidates are in increasing order of quality.

1, 3, 7, 10, 15 (ranks)

✗ ✗ ✗ ✗ ✓

$O(c_h n)$

Probabilistic Analysis.

- Uses probability to analyze the problems.

In order to use prob. analysis, we use knowledge and make certain assumptions about distribution of inputs and then we analyze & compute our average case running time.

candidate i, rank(i)
candidates appear in random order.
Ranks will form uniform random permutation
- n! permutations with equal probability.

Randomized Algorithms

Agency may send candidates in increasing quality, to get more money.
So we need more control.

- Change the model.
- Agency will have list of n candidates.
- Take that list & organize it in some random order.
- Use any random generator function to randomize it.

What is randomized algorithm.

An algorithm is randomized in behaviour if the input is produced by random generator.

If the input is random, running time is expected running time.

Indicator Random variable (IRV)

It is a method to convert between probabilities and expectations.

Let Indicator variable - $I\{A\}$

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{if } A \text{ not occur.} \end{cases}$$

Ex. Coin toss

$$X_h = I\{H\} = \begin{cases} 1 & \text{if Head} \\ 0 & \text{if Tail} \end{cases}$$

* In one flip, expected number of head

$$E[X_h] = E[I\{H\}]$$
$$= 1 \cdot P\{H\} + 0 \cdot P\{T\}$$
$$= 1 \times \frac{1}{2} + 0 \times \frac{1}{2}$$
$$= \frac{1}{2}$$

Lemma

When we are given set S and event A

Let $X_A = I\{A\}$

$$\boxed{E[X_A] = P\{A\}}$$

* In n flip.

$$X := \sum_{i=1}^{n} X_i$$

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

$$= \sum_{i=1}^{n} \frac{1}{2}$$

$$= \frac{1}{2} + \frac{1}{2} + \cdots$$

$$= \frac{n}{2}$$

Analysis of Hiring Problem using IRV

Let X be the random variable = number of times we hire a new assistant.

X_i be IRV for i^{th} candidate.

$$X_i = \begin{cases} 1 & \text{if hired} \\ 0 & \text{if not hired.} \end{cases}$$

$$X = X_1 + X_2 + X_3 + \cdots + X_n$$

$$E[X_i] = P(\text{Candidate is hired})$$

To get hired, i should be better than those who came, 1 to i-1

∴ Candidate i will get hired $= (1/i)$

This is the probability, if the order is random & quality is equally likely.

$$E[X_i] = \frac{1}{i}$$

$$E[X] = E\left[\sum_{i=1}^{n} x_i\right]$$

$$= \sum_{i=1}^{n} E[x_i]$$

$$= \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)$$

↳ log natural

~~$= \log n + O(1)$~~

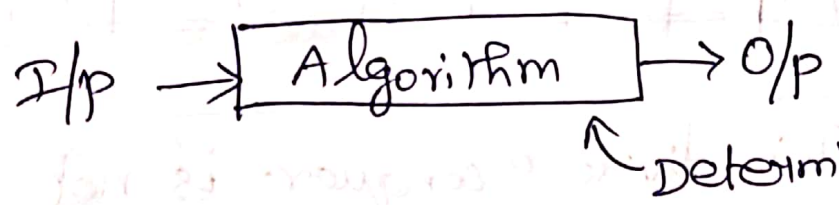
We interviewed n people, but hired ln n.

Avg case : $\boxed{O(c_h \ln n)}$ ← Hiring cost

Randomized Quick Sort

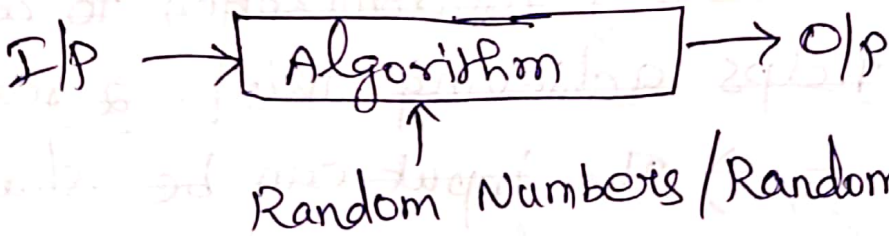
What are Randomized Algorithms?

* Unlike deterministic algorithms that aim to solve a problem correctly and takes fixed no. of steps depending on the size of the input.



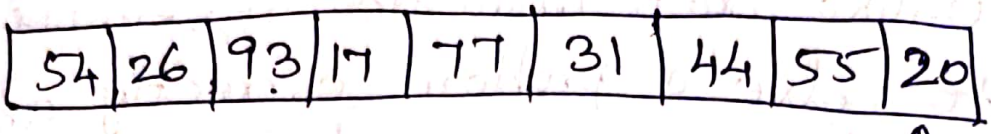
* Randomized Algorithms on the other hand, in addition to the input, attempts random choices during execution.

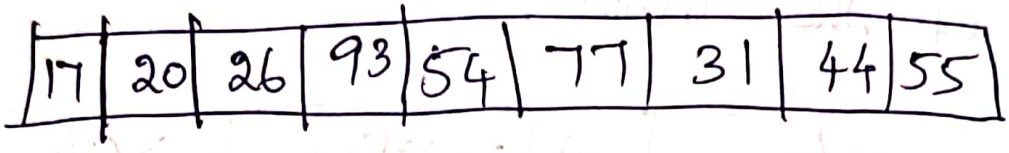
* This leads to change in the behaviour (time & space complexity) with respect to a fixed input.



* This change in behaviour is likely to be more efficient.

* Recall QuickSort.





This divide & conquer is not fair and does not yield $\Omega(n \log n)$ (best case) & yields $O(n^2)$!

* If the pivot is closer to median of inputs, there is a chance of achieving the best case time complexity.

* Introducing randomization to an extent helps achieving this! a randomness

a) The input can be shuffled.

b) The choice of pivot can be done randomly every time, we divide the array.

This can worsen if the i/p is already sorted.

Algorithm:

```
Randomized_Partition (A,P,r)
{ i= Random (P,r)
  exchange A[r] with A[i]
  return partition (A,P,r)
}
Randomized_Quicksort (A, P, r)
{ if (P<r)
  { q = Randomized_Partition (A, P, r)
    Randomized_Quicksort (A, P, q-1)
    Randomized_Quicksort (A, q+1, r)
  }
}
```

Time Complexity = $O(n \log n)$ (Best, worst & Average case)

Rabin karp Algorithm.

- is a string Matching algorithm or pattern Matching algorithm.

If a text & pattern are given, we have to find out, whether the pattern is present in the text or not.

Text

1	2	3	4	5	6
a	a	a	a	a	b

n=6

1+1+1=3 ×

1+1+1=3 ×

1+1+1=3 ×

1+1+2=4 ✓

a-1
b-2
c-3
d-4
e-5
f-6
g-7
h-8
i-9
j-10

Pattern

1	2	3
a	a	b

m=3

1+1+2=4 ← hash code

↑ hash function.

To check whether pattern is present in text, assume values for alphabets. (a-1, b-2 etc).

In pattern, apply value & sum (1+1+2=4)

~~In text~~ Patten size m=3, so in text take first 3 chars & add the values (3). It is not ~~not~~ equal to 4, Then slide

2	3	4
a	a	a

1+1+1=3.

Again not matching, repeat this till it matches finally a a b . It matches with
4 5 6
1+1+2=4

the pattern. Once it matches, confirm, whether the strings are exactly equal.
Indices - 4, 5, 6)

EX.2

	1	2	3	4	5	6	7	8
Text n=8	a	b	c	d	a	b	c	e

2+3+5=10

	1	2	3
Pattern m=3	b	c	e

2+3+5=10

strings are matching with the indices 6, 7, 8.

$$T(n) = \Theta(n-m+1) \quad \text{(Average time)}$$

EX3

	1	2	3	4	5	6	7	8	9	10	11
Text n=11	c	c	a	c	c	a	a	e	d	b	a

3+3+1=7 (equal but pattern not matching)

3+1+3=7 (")

1+3+3 (")

3+3+1 (")

3+1+1 x

1+1+5=7

	1	2	3
Pattern m=3	d	b	a

4+2+1=7

Hash code is matching but the patterns are not matching. These are called as spurious Hits.

Time = $O(mn)$

To avoid the spurious hits, we have to take a strong hash function given by Rabin Karp.

Pattern $\overset{1}{d}\ \overset{2}{b}\ \overset{3}{a}$

$m=3$

$4\times10^2 + 2\times10^1 + 1\times10^0$

$400 + 20 + 1 = \boxed{421}$

10 is taken as the base value, because in this eg, we have considered alphabets from a to j (1 to 10).

	1	2	3	4	5	6	7	8	9	10	11
Text	c	c	a	c	c	a	a	e	d	b	a

$n=11$ $\quad 3\times10^2+3\times10^1+1\times10^0$

$331 \neq 421$

Next we need to consider $\overset{2}{c}\ \overset{3}{a}\ \overset{4}{c}$

$3\times10^2+1\times10^1+3\times10^0 = 313$

Instead calculate it again and again we can

use rolling hash function ie,

$[[3\times10^2+3\times10^1+1\times10^0]-3\times10^2]\times10+3\times10^0$

$[331-300]\times10+3$

$310+3=313\neq421.$

* 2 4 5
a c c

$313-300=13\times10=130+3=133\neq421$

* 4 5 6
c c a

$133-100=33\times10=330+1=331\neq421$

* 5 6 7
c a a

$331-300=31\times10=310+1=311\neq421$

* 6 7 8
a a d

$311-300=11\times10=110+4=114\neq421$

* 7 8 9
a e d

$114-100=14\times10=140+4=144\neq421$

* 8 9 10
e d b

$144-100=44\times10=440+2=442\neq421$

* q 10 11
d b a

442 − 400 = 42 × 10 = 420 + 1 = 421 equal to 421

Pattern is also matching. (d b a)

It is ~~cal~~ also called as Rabin fingerprint function.

Time

$T(n) = \Theta(n-m+1)$ (Avg case)

$T(n) = \Theta(mn)$ (Worst case)

⇓

But worst case chances are reduced.

And here in calculation of hash function, we got large values

$4 \times 10^2 + 2 \times 10^1 + 1 \times 10^0 = 421$

It may be larger than the int size. So we can use modulo.

$(4 \times 10^2 + 2 \times 10^1 + 1 \times 10^0) \% 2^{31}$ (For 32 bit, 1 bit ⇒ sign bit)

But sometimes it may again generate spurious hits. But we can define our own hash function.

Approximation Algorithm
Vertex Cover Problem

* A set of vertices such that each edge of the graph is incident to at least one vertex of the set, is called the vertex cover.
* Greedy algorithm may or may not produce optimal solution.
* Approximation algorithm does not always guarantee optimal solution but it's aim is to produce a solution which is as close as possible to the optimal solution.

vertex cover problem

* In the mathematical discipline of graph theory, "A vertex cover (node cover) of a graph is a subset of vertices which "covers" every edge.
* An edge is covered if one of its endpoint is chosen.
* In other words, "A vertex cover for a given graph G is a set of vertices incident to every edge in G.".

* The vertex cover problem: What is the minimum size vertex cover in G?

Idea: ~~Greedy Algorithm~~

keep finding a vertex which covers the maximum number of edges.

Step1: Find a vertex v with maximum degree.

Step2: Add v to the solution & remove v and all its incident edges from the graph.

Step3: Repeat until all the edges are covered.

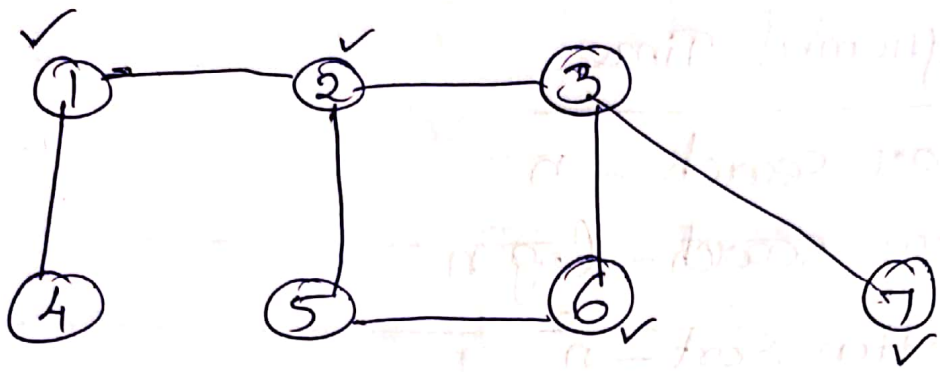
* vertex-cover ~~problem~~ problem is NP-complete.

Alg:

APPROX_VERTEX_COVER

1. $C \leftarrow \phi$;
2. $E' \leftarrow E$
3. while $E' \neq \phi$ do
4. let (u,v) be an arbitrary edge of E'
5. $C \leftarrow C \cup \{(u,v)\}$
6. remove from E' all edges incident on either u or v
7. end while.

Ex: Find vertex cover with minimum size.



Select 1—2

Edges (5,6), (3,6) & (3,7) are not covered by it

select vertex 6 also.

But (3,7) is not covered.

select 1, 2, 6 & 7. Now all the edges are covered. The size is 4.

If we select 1, 3 & 5, all the edges are covered & the size is 3

NP-Hard and NP-complete.

Polynomial Time

Linear search – n

Binary search – $\log n$

Insertion Sort – n^2

Merge sort – $n \log n$

Matrix Multiplication – n^3

Exponential Time

0/1 Knapsack -
Traveling SP -
Sum of subsets -
Graph coloring -
Hamiltonian cycle
Satisfability problem.

} 2^n in common.

For the exponential time algs, we need polynomial time algorithm.

If we can't write polynomial time deterministic alg for exp. time alg, we can

try to write polynomial time nondeterministic alg.

Deterministic: Each line is clear, we know everything clearly.

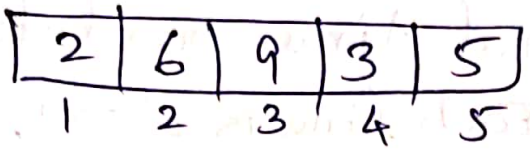
Non-deterministic

Some statements can not be deterministic. In future, somebody else will convert this non-deterministic stmts into deterministic.

Ex:

```
Algorithm Nsearch (A, n, key)
{
    j = choice();          ——— 1
    if (key = A[j])
    {
        write(j);
        success();         ——— 1
    }
    write(0);
    failure();             ——— 1
}                                  O(1)
```

choice() is a non-deterministic stmt. we don't how it chooses.



If key is 9 ⇒ index is 3
But we don't know how it is choosen in constant time.

class P

- set of deterministic algorithms which are taking polynomial time.

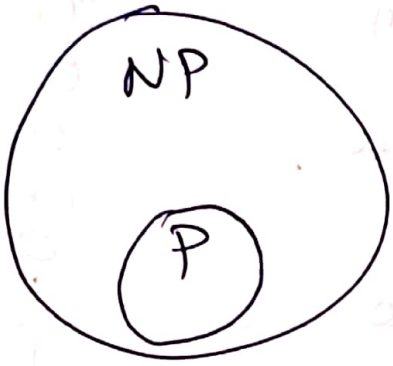
Ex: Linear search, Binary search, MST, Huffman coding. etc.

class NP

- set of non-deterministic algs taking polynomial time.

Ex: ~~(Exponential time algs)~~

For exponential time algs, we try to make polynomial. But we don't know how to make. So we can leave the stmts as non-deterministic. (Ex. Previous one). So alg. becomes non-deterministic.



P is a subset of NP.

The deterministic alg. today were the part of non-deter. alg.

Ex: Merge sort found recently. It becomes deterministic. Previously it was non-deterministic

Non-deter (yesterday) - Deter. (Today)

Non-determ (Today) - Determ (Tomorrow)

CNF- Satisfiability.

$x_i = \{x_1, x_2, x_3\}$

$$CNF = \underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{C_1} \wedge \underbrace{(\bar{x} \vee x_2 \vee \bar{x}_3)}_{C_2}$$

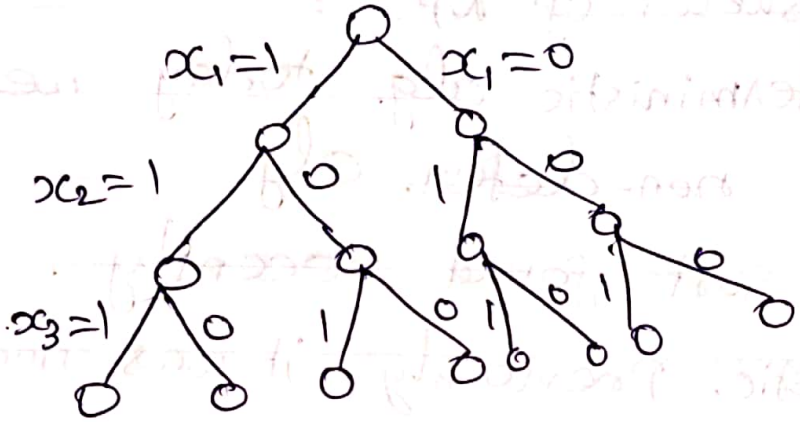
Satis problem is to find, for what values of x_i this formula is true.

Time = $2^3 \Rightarrow 2^n$

Exponential Time.

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

State space Tree



Parse from Root to leaves give solution

Ex:

0/1 Knapsack

$P = \{10, 8, 12\}$ $n=3$

$w = \{5, 4, 3\}$ $m=8$

$x_i = \{0/1, 0/1, 0/1\}$

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
⋮		

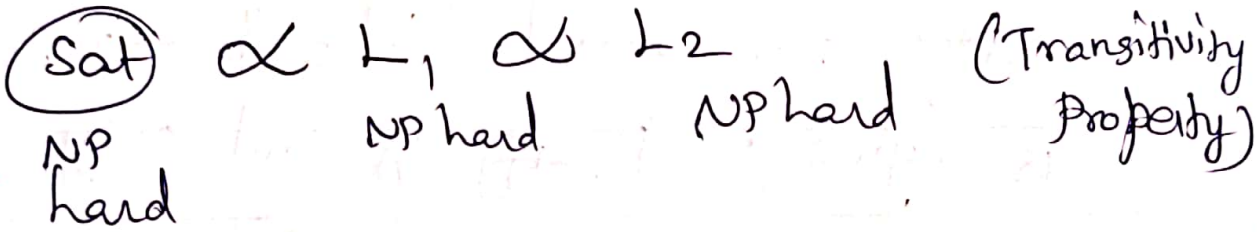
This is similar to the satisfiability ptlm. so if we get soln. for this, we get for that also.

NP hard

Satisfiability is a base pblm for converting exp → poly time.

It is NP hard. If sat is reduced to 0/1 knapsack, then 0/1 knap is also NP hard

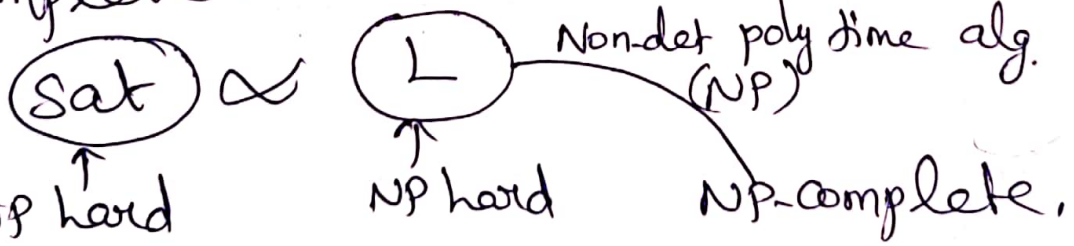
$I_1 \Rightarrow$ Instance



(Transitivity Property)

NP Complete

we have a non-deterministic polynomial time alg for satisfiability pblm. So it is NP-complete.

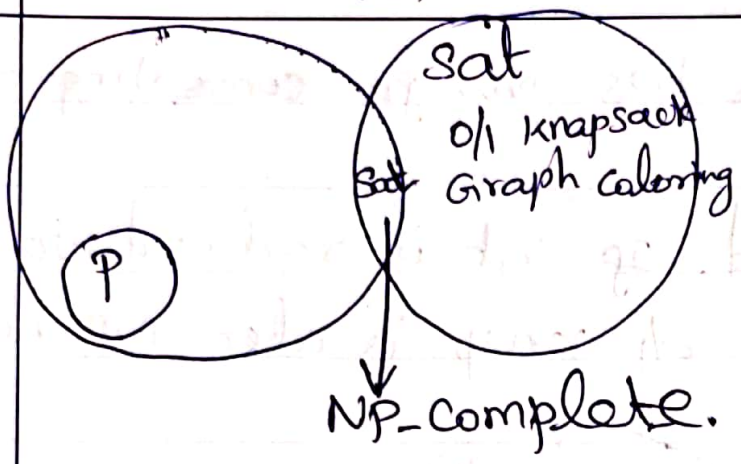


On any problem (L) we need to show that it is directly/indirectly related to sat. & write non-deter poly time alg.

NP (write non-deter alg)

NP - Hard

NP - Non-deterministic alg.



NP-complete.

$P \subseteq NP$

Able to prove $P = NP$

Cook's has proved that, - if satisfiability is lying in P, ~~iff~~ if and only if $P = NP$.