# Lecture Notes Class 1 and 2

Navketan Verma

Technical University of Munich

## Randomised Algorithm

A randomised algorithm is an algorithm is an algorithm which employs a degree of randomness as part of its logic. The algorithm typically uses random bits as additional input along with the regular inputs with the hope of achieving good performance in the average case over all possible choices of random bits or in simple terms Randomness is another resource that we use along with regular resources to solve any problem.

## Examples of Random Sources

Coin tossing, Radioactivity etc...(more to be added). In common usage, randomised algorithms are approximated using a pseudo random generator in place of true random bits. Such an implementation may deviate from the expected theoritical behaviour .

## Reason

Helps in fooling the adversary.

There are broadly two categories of Randomised Algorithm :- **a)** The algorithms that use random input to reduce the expected running and memory or any other resource but always terminate with correct or expected result in a bounded amount of time **b)** The probabilistic algorithms, which depending on random input have a chance to produce incorrect result (one way or two way error reffered to as false positive or false negative). These algorithms usually fail to terminate in case of incorrect input (e.g Las Vegas Algorithm) or give an incorrect result (Monte Carlo Algorithm). Error part of the results are usually reduced with successive re-runs but only with a polynomal number of re-runs.

In this part of the course, we are deailing with category of the algorithm.

Some of the algorithms discussed in the class :-

### a. Leader Election problem

Its used to designate a single process (or node in a distributed system) as a leader among several nodes, and its made known to all other nodes quickly. The nodes in the Distributed system need some method to break the symmetry and annoint one as the leader ( either highest or lowest UID or any other means).

**b) Fermat's Primality theory**

Fermat theorem is normally used to create a large prime number with high probability (could be used either in crytography or for credit card number). A Fermat's number is typically of the form $F_n = 2^{2^n} + 1$, where n must be non-negative. The first few Fermat numbers are:
3, 5, 17, 257, 65537, 4294967297, 18446744073709551617.....
But like composite numbers of the form $2^p$ - 1, every composite number Fermat number is a strong pseudoprime to base 2. As a result we only presume that we can generate large prime numbers with only high probability but there is a chance that such a number might turn out to be a pseudo prime.

Formally, a composite number n = d.2s+ 1 with d being odd is called a strong pseudoprime to a relatively prime base a when one of the following conditions hold: The definition of a strong pseudoprime depends on the base used; different bases have different strong pseudoprimes. The definition is trivially met if $a \equiv \pm 1$ mod n so these base choices are often excluded.

If we try to find deterministically primality of a number, it will be of the order of $O((logn)^4)$.

## Paradigms of Randomized Algorithm (Abstract models/Notions)

**1.** oiling/Fooling the adversary - Examples :- a) Finger Printing and Hasing b) Random Re-ordering. A Randomised Quick Sort's overall runing time is O(nlogn) which is same as the deterministic Quick Sort's average case (trying to make bad permutations really small).
**2.** Load-Balancing - (to be clarified)
**3.** Markov chains - walking through a collection of objects
**4.** Isolation and Symetry Breaking e.g Leader selection problem (each single node pick one random number look for smallest or largest , should be non-identical for two or more, then elect that as the basis isolating the number basically)

### Probabilistic Methods in existence proofs

Doing Combinatorial counting, we can construct such a proof. E.g Using Probability theory, at each step we throw away atleast half the possible outcomes, then after k-steps as are left with $1/(2^k)$ items (proof of existence).
**Probabilistic analysis of algorithms**
**Randomized Quick Sort :** S be the set of n numbers. Let pivot $\rightarrow$y. Let us partition the set S\{$y$} into two sets S1 and S2, where S1 consists of elements smaller than y and S2 consists of elements larger than y. We recursively sort S1 and S2 such that S1 is in ascending order followed by y followed by S2 (in ascending order as well).

If we find y in c*n steps for some c¿0, we can partition S\{y} into S1 and S2 in (n-1) steps by comparing each remaining element of S\{y} with y. The recurrence relation is as follows :-
$T(n) \leq 2T(n/2) + cn + (n-1)$
$T(n) \leq 2T(n/2) + (c+1)n$
and its solution gives us $T(n) \leq dn \log n$ for some constant d, where T(k) is the time taken to sort k elements in the worst case. This running time holds even if the partition of S\{y} into S1 and S2 doesn't have more than $3n/4$ (or 3-quarters) of the elements. This gives a chance since we know that n/2 candidate element 'y' will possess this property. Big question :- How to find one such element (pivot)?
**RandQS :-** An algorithm with random choices for pivot during its execution (at every step during recursion) can be obtained in unit time. But this will not succeed 100

Input is a set S of n numbers.

Output is the same set in sorted order (increasing/decreasing).
**1)** Pick element y from set S randomly (pivot element Random bits are bernoulli distributed).
**2)** Compare S\{y} to y . Determine $S1 < y < S2$ (1st split of S into S1,S2 and y).
**3)** Recursively sort S1 and S2 and put solutions together by first wirting sorted version of S1 followed by 'y' followed by sorted version of S2.

In any sorting algorithm, normally we measure running time in terms of number of comparisons it performs, since this is the dominant cost in any reasonable implementation. Here our goal is to analyze expected number of compararions in the execution of RandQS.
For $1 \leq i \leq n$, let $S_{(i)}$ denote the element of rank i/the $i^{th}$ smallest element in the set. $S_{(1)}$ denote the smallest and $S_{(n)}$ the largest. $S_{(1)} < S_{(2)} < S_{(3)} <$ ..... $< S_{(n)}$.

Define a random variable $X_{ij}$ such that
$X_{ij} = 1$ if algorithm compares $S_{(i)}$ and $S_{(j)}$
$= 0$ otherwise.
$X_{ij}$ = Count of number of comparasions between $S_{(i)}$ and $S_{(j)}$

Total number of comparasions = $\sum_{i=1}^{n} \sum_{j>1}^{n} X_{ij}$

Expected value of the total number of comparisons =

$$E[\sum_{i=1}^{n} \sum_{j>1}^{n} X_{ij}] = \sum_{i=1}^{n} \sum_{j>1}^{n} E[X_{ij}]$$

[ due to the property of linearity of expectation] .

Let $P_{ij}$ denote the probability tha S(i) and S(j) are compared in an execution. Since $X_{ij}$ only assumes values of 0's and 1's, therefore
$E[X_{ij}] = 1 * P_{ij} + 0 * (1 - P_{ij}) = P_{ij}$.
Now Compute $P_{ij}$.

Now for $i < j, S_{(i)}, S_{i+1}, S_{i+2}, ....S_j$

For execution of *RandQS* , we view it in the form of binary tree T, with each node carrying an element of S. The root of the tree is labeled by 'y' $\rightarrow$ Pivot element. The left subtree contains elements of S1 smaller than 'y' and right subtree contains elements of S2 larger than 'y'. The root element is compared with elements in the two sub-trees, but no comparasion is performed between elements in the two sub-trees (i.e. Element from left sub-tree with element from right sub-tree). Therefore there is a comparasion between S(i) and S(j) only if one is the ancestor of other. The output will will be an in-order traversal if T. For analysis, we are interested in level order traversal of the nodes. The permutation $\pi$ is obtained be visiting nodes of T in the increasing order of the level numbers, from left-to-right and top-to-bottom. The $i^{th}$ level of the tree is the set of all nodes at a distance i from the root.

**Two Observations**

**1)** There is a comparasion between $S_{(i)}$ and $S_{(j)}$ if and only if $S_{(i)}$ and $S_{(j)}$ occurs earlier in the permutation $\pi$ than any element $S_{(l)}$ such that $i < l < j$. i.e for comparasion to happen, $S_{(i)}, S_{(l)}$ and $S_{(j)}$ have to be in ancestor dependent relationship with each other by RandQS algorithm.
**2)** Any element $S_{(i)}$, $S_{(i+1)}$, $S_{(i+2)}$..... $S_{(j)}$ is equally likely to be chosen as the partitioning element (or Pivot), appearing as the $1^{st}$ element in $\in$. Therefore , probability that the element is either $S_{(i)}$ or $S_{(j)}$ is $1^{st}$ element (Mutually exclusive) is exactly $= \frac{2}{(j-i+1)}$.

Expected no of comparisons = E $[\sum_{i=0}^{n}\sum_{j>i}^{n} X_{ij}] = \sum_{i=1}^{n}\sum_{j>i}^{n} E[X_{ij}]$

$$= \sum_{i=1}^{n}\sum_{j>i}^{n} P_{ij}$$

$$= \sum_{i=1}^{n}\sum_{j>i}^{n} \frac{2}{(j-i+1)}$$

$$= \sum_{i=1}^{n}\sum_{k=1}^{n-i+1} \frac{2}{k} = 2\text{nHn} = 2\text{nlogn}$$

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

**Min-Cut Algorithm**

Two events $\varepsilon 1$ and $\varepsilon 2$ are independent if the probability that they both occur is given by $\Pr[\varepsilon 1 \wedge \varepsilon 2] = \Pr[\varepsilon 1]$ * $\Pr[\varepsilon 2]$ In more general case where $\varepsilon 1$ and $\varepsilon 2$ are not necessarily independent.
$\Pr[\varepsilon 1 \wedge \varepsilon 2] = \Pr[\varepsilon 1]$* $\Pr[\varepsilon 2]$

$\Pr[\frac{\varepsilon 1}{\varepsilon 2}]$ = Conditional probability of $\varepsilon 1$ given $\varepsilon 2$ .
$\Pr[\varepsilon 1] = Pr[\varepsilon 1]$ *$\Pr[\frac{\varepsilon 1}{\varepsilon 2}]$* $\Pr[\frac{\varepsilon 3}{\frac{\varepsilon 2}{\varepsilon 1}}]$*..... $\Pr[\frac{\varepsilon i}{\varepsilon 1 \cap \varepsilon 2 \cap \varepsilon 3 ..... \cap \varepsilon i-1}]$

**Min-Cut Algorithm:-**

A cut in a multigraph G is the set of edges whose removal results in G being broken into two or more components. A min-cut is a cut of minimum cardinality.

**Another Definition :-**

Dividing the graph into two non-trivial graphs. Number of edges crossing the cut is the number of that cut. The Cut with minimal links is called the Min-Cut.
**Algorithm :-** Pick an edge uniformly at random and merge the two vertices at its end-points. If as a result there are several edges between some pairs of newly formed vertices, retain them all. Edges between vertices that are merged are removed, so that there are never any self loops into a single vertex as contraction of that edge. The crucial observation is that an edge contraction does not reduce the min-cut size in G.
The algorithm continues till only two vertices remain in the graph, at this point , the set of edges between these two vertices is a cut in G and output is a candidate min-cut.
d(v) = degree of v = number of edges incident on v.
Let k = min-cut size if a particular min-cut 'C'of graph G.

Therefore number of edges of $G \geq \frac{k*n}{2}$ . we execute a sequence of (n-2) edge contractions. i.e pick an edge randomly, then collapse the two ends of the edge. Keep repeating this many times and remember the smallest cut. If the number of edges in the graph is less than $k*(n/2)$, then degree of the last two vertices in the graph (after (n-2) contractions) is less than k, and therefore size of min-cut will be less than k. Let denote the event of not picking an edge of C at the $i^{th}$ step.

$\Pr[\textit{if edge is in the cut}] \leq \frac{(numberofedgesofMin-Cut'\,C')}{totalnumberofedges}$

$\Pr[\textit{if edge is in the cut}] \leq \frac{k}{kn/2} \leq \frac{2}{n}$

$\Pr[\varepsilon 1] \geq$ 1- $\Pr[\textit{if edge is in Min-cut}\geq 1 - \frac{2}{n}$
If $\varepsilon$ occurs, in second step , number of edges to be considered $\geq \frac{k(n-1)}{2}$

$Pr[\frac{\varepsilon 2}{\varepsilon 1}] \geq 1 - \frac{k}{l*((n-1)/2)} \geq 1 - \frac{2}{(n-1)}$

$\Pr[\frac{\varepsilon i}{\varepsilon 1 \cap \varepsilon 2 \cap \varepsilon 3 ..... \cap \varepsilon i-1}] \geq 1 - \frac{2}{n-i+1}$

$\Pr[\varepsilon 1 \cap \varepsilon 2 \cap \varepsilon 3 ..... \cap \varepsilon i - 1] = (1 - \frac{2}{n-i+1}) \geq \frac{2}{n*(n-1)} \geq \frac{2}{n*n}$
The probability of discovering a particular min_cut (which may in fact be the unique min_cut in G) is larger then $2/(n^2)$. If we repeat the algorithm $(n^2)/2$

times, making independent random choices each time, the probability the min_cut is not found in any of the $(n^2)/2$ attempts is at most.

$(1 - \frac{2}{n*n}) \leq 1/e$. By $(n^2)/2$ re-runs, we have managed to reduce the probability of failure from $(1 - \frac{2}{n*n})$ to $1/e$

**Binary planar partition :-**

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

**Ex 1.1**

$E[\sum_{i=1}^{40} X_i] = \sum_{i=1}^{40} E[X_i] = \sum_{i=1}^{40} 1/40$

Since the probability that ith sailor chooses its own cabin = 1/40.
Binary planar partition of a set of n disjoint line segments in the planes BPP (Binary planar partition) consists of a binary tree . Every internal node of tree has two children. For every node, there exists a region r(v) of the plane. Region corresponding to root is the entire plane. Region corresponding r(v) is partitioned by l(v) into two regions, r1(v) and r2(v) . Any region 'r' of the partition is bounded by the partition lines on the path from root to the node corresponding to r in the tree. Given the set S = $S_1, S_2...S_n$ of non intersecting line segments in the plane, we wish to find binary planar partition, such that every region in the partition contains at most one line segment (or one portion of line segment). As a result an input line segment $S_i$ will be divided into separate line segments $S_{i1}, S_{i2}$.... where each one of them will be lying in different region.

**Painters Algorithm :-**

First draw the objects that are farthest behind and then progressively draw the objects that are in front.
**Recursive implementation of binary partition tree with painters algorithm :-**
At the root of the BPP tree, determine which side of the partitioning line 'L1' is behind from the viewpoint and render all objects in that sub-tree (recursively). Once done with the back portion of 'L1', do it again for the portion in front of 'L1', painting over objects already drawn.

Time to render is directly proportional on the size of the binary planar partition tree, therefore its good to construct as small BP partition as possible. The tree needs to be traversed completely, but its depth is not exactly that is meaningful, because the construction of the new partition will break a segment Si into smaller pieces, the size of the partition need not be n or O(n).

For a segment S, let L(s) be the line segment obtained by extending (if necessary) S on both sides to infinity. For $S = S_1, S_2....S_n$ a simple class of partition is the set of autopartition, formed by only using lines $L(S_1), L(S_2)...L(S_n)$ in constructing the partition. Here we will consider randomized auto partition.

**Partition Tree :-** Start with whole thing and divide into two parts and then recursion. Partition tree is of the order of $O(n \log n)$. Line segment of infinite length L(s) in both direction.

Input: Set S = $S_1, S_2...S_n$ of non intersecting line segments. Output: A binary antipartition P $\pi$ of S.

1. Pick a permutation $\pi$ of 1,2,...n uniformly at random from n! Possible permutations.
2. While a region contains more than one segment (sub-segment), cut it with $l(S_i)$, where i is first in the ordering permutation $\pi$ such that $S_i$ cuts that region. [while there exists a region with $> 1$ sub $-$ segment, pick the first segment in the region as a new partitioning line.]

**Theorem**

The expected size of autopartition output by the algorithm is $O(n \log n)$.

**Proof:** u,v be two segments.

index(u,v) = if u intersected v at ith position i.e. Before v, u has already intersected (i-1) segments.

index(u,v) = 3(or k say) = index(u,w), where u is not equal to w i.e we can have only two intersections with same index for one segment (say u $\rightarrow$ on two sides of its infinte length). (u $\rightarrow$ on two sides of its infinte length)

u intersects $v \Rightarrow l(u)$ intersects v, index (u,v) = $i \Rightarrow$
u $\rightarrow$ u1, u$\rightarrow$ u2.....u $\rightarrow$ ui-1, u$\rightarrow$ v
index(u,v)=i if u is picked by randomiyed algorithm before any of $u_1, u_2...u_{i-1}, v$
Probability of this happening = $\frac{1}{i+1}$ in current region.
Random variable :

$$C_{uv} = \begin{cases} 1 & \text{if u intersects v} \\ 0 & \text{if u intersects v} \end{cases}$$

$E[C_{uv}] = Pr[u \cap v] \leq (\frac{1}{index(u,v)+1})$
E[*size of partition tree*] $= n + E[\sum_{u=1}^{n} \sum_{v \neq u}^{n} C_{uv}]$

$$E[\sum_{u=1}^{n} \sum_{v \neq u}^{n} C_{uv}] = \sum_{u=1}^{n} \sum_{v \neq u}^{n} E[C_{uv}] = \sum_{u=1}^{n} \sum_{v \neq u}^{n-1} \frac{2}{i+1} = 2nHn$$

this is due to linearity of expectation.

E[ *size of patition tree*] $\leq (n+2nHn) \leq n(1+2Hn) \leq n(1+2\log n) \Rightarrow O(n \log n)$

Our randomized algorithm actually provides a proof of existence of the intersection of line segments by other segments. Here again RA analysis is around same as the average case anlysis of Deterministic algorithms. [Andy Yao's principle]

**Another way of interpreting the above theorem:-**

Since the expected size of the binary planar partition constructed by the algorithm is $O(n \log n)$ on any input, there must exist a binary auto partition of size $O(n \log n)$ for every input.

**Hiring a new secretary/employee:-**

Proof: M needs a new secretary. n candidates Ci, for i=1....n are in the fray for interview.
Decision right after the interview. Cost of relinquishing the contract = b. If number of candidates selected = k, then total cost of relinquishing = (k-1)b, as (k-1) removed. Worst case (deterministic) = (n-1)b $\leftarrow$ every one hired and then (n-1) removed.

**Randomized algorithm:-**

randomized order of input. Random variable Xi

$$X_i = \begin{cases} 1 & \text{if the candidate is picked or selected} \\ 0 & \text{i otherwise} \end{cases}$$

E[ *Total Cost*] = $[\sum E[X_i] - 1] \subset b = [\sum E[X_i] - 1] \subset b$
$\Pr(X_i = 1)$ = Probability of the firing of $i^{it}$ candidate chosen randomly at a time = 1/i.
E[ *Total Cost* ] = $[\sum E[X_i] - 1] \subset b \leq [H_{n+1} - 1] \subset b \Rightarrow O(\log n)$
Here randomized algorithm is used to escape adversary/worst case probability.

**Matrix Multiplication or finger printing:**

Let 3 matrices be A,B,C $0,1^{nxn}$, we need to find A.B = C ?
Matrix multiplication MM: O($n^w$ ) w = 2.376 is the fastest result possible.
Randomized algorithm: choose r $0,1^n$ at random. Compute Br = x
Compute Ax = y
Compute Cr = z
accept if y =z $O(n^2)$
Observation if $A.B = C$, we get accepted with $O(n^2)$ otherwise $O(n^2)$ with probability $\geq 1/2$
Proof:- $D = A.B - C! = 0$, repeat the alogorithm k-times (independent random choices).
*Errorprobability* $\leq 2^{-k}$

**Game Tree Evaluation :-**

Motivation → Two person zero sum game. For every configuration, finite number of moves are possible. Configuration tree formation has to be done, leaves of the configuration tree as final or non-final state. My Move's layers → 0,2,4,6....(also called as min layers as i will try to minimize the cost layers which are at even distance from the root).
Opponent's Move's layers → 1,3,5,7....(also called as max layers layers which are odd distance from the root). Associated with each leaf is a real number which we call its value. The evaluation of game tree is as follows: each leaf returns the value associated with it. Each max node returns the largest value returned by its children whereas each min node return the smallest value returned by its children. Given a tree with value at its leaves, we have to determine the value at its root. The children of a node can be directly associated with the options available to any player at any point in the game. The value returned by the leaves represent the value of the game for either player. One seeks to minimize the cost whereas the other to maximize. In our discussion, value if leaves is limited to the bits 0 and 1. Thus each min node can be thought of as the boolean AND operation and each max node can be thought of as the boolean OR operation.
Let $T_{d,k}$ denote a uniform tree in which root and every internal node has 'd' children and every leaf is at a distance 2k from the root.Thus any root to leaf path passes through exactly k-AND nodes (including the root) and k OR nodes and there are $d^{2k}$ leaves. Number of layers in such a tree = 2k+1, numbered 0,1,2,3...2k.Values at the nodes are 0,1.
$T_{2,k}$ → each step has two possible moves (two states of non-determinism). And every leaf is at a distance 2k from the root. There are a total of $2^{2k}$ leaves $=4^k$ leaves. Number of layers in such a tree = 2k+1, numbered 0,1,2,3...2k. Values at the nodes are 0,1.
To evaluate an AND internal node v, the algorithm chooses one of its children (a sub-tree rooted at an OR node) at random and evaluates it recursively invoking the algorithm. If 1 is returned by the sub-tree, the algorithm proceeds to the other child (again recursively). If 0 is returned, the algorithm returns 0 for v. To evaluate an OR node, the same procedure applies with roles of 0 and 1 interchanged. We now argue by induction on k that expected cost of evaluating any instance of $T_{2,k}$ is at most $3^k$. The above problem of the Game tree evaluation can be viewed as a Quantified Boolean Formula or QBF where the quantifiers in the expression are alternated as Universals and existential, depending on the scenario.
$\forall X_1 \exists X_2 \forall X_3 \exists X_4 ....P(X_1, X_2, X_3, X_{2k}) \subseteq_2 \pi$
The only difference is that QBF starts from existential and then universal, here its the opposite, 1st universal then existential. Both PSPACE complete problems and are contained in polynomial heirarchy-PH.

**Randomized Algorithm for evaluating $T_{2,k}$ :-**

AND node :- Choose one of its two child randomly and evaluate recursively if its 0, then output 0(no need to evaluate the other child), else if the output is

1 yes then also evaluate (recursively) the other child. If output 1, then give the output 1 else output 0.
OR node :- Same procedure but exchange the role of 0 and 1. Here if $1^{st}$ output is 1, output 1 else recursively check for the other one. If one of them is 1, output 1 else output 0.
Theorem:- Expected number of leaves to be evaluated in $T_{2,k}$ tree is leaves $\leq 3^k$ instead of $leaves \leq 4^k$ as in deterministic algorithm.
Proof by Induction :- Induction over k. If the whole layer is 2k then OR layer is $T_{2,k-1}$. If value(OR) = 0 : both children needs to be evaluated $cost \leq 2.3^{k-1}$
If value(OR) = 1 : with probability atleast 1/2 algorithm picks a child of OR with value 1.
: with probability 1/2, algorithm picks a child of OR with value 0, so both children needs to be evaluated.
$cost \leq (1/2).3^{(k-1)} + (1/2).2.3^{k-1} \leq (3/2).3^{k-1}$ Now consider the next root of the $T_{2,k}$ tree which is an AND node. Since its an AND case so it will return a 1 if both its sub-trees return 1 (which are infact OR rooted as shown above).

$$cost \leq (1/2).2.3^{k-1}+(1/2).[(3/2).3^{k-1}+2.3^{k-1}] \leq 3^{k-1}.[1+1+(3/4)] \leq 3^{k-1}.(11/4) \leq 3^k$$

**LAS VEGAS Vs MONTE CARLO:-**

LAS VEGAS :- Here algorithm always gives the correct solution. Only thing that varies from one execution to the other is the running time. e.g. for this type of algorithm is Randomized Quick Sort.

**MONTE CARLO:-** In this algorithm errors can happen. Error's may be of one sided or both sided. Also they can be of the type false positive or false negative. e.g. of such an algorithm is min-cut algorithm. Here results sometime may be in correct. With a number of re-runs we try to bound the probability of the error. What we do basically is to minimize the error probability by a substantial number of re-runs with gaining accuracy at the expense of time. Here in this case we get basically a geometric distribution = $P(1-P)^y$ , where r is the number of re-runs. P = probability of success and 1-P = probability of failure of the algorithm. As you can see since 1-p is less than one, as we increase the number of re-runs, the probability of failure will go down significantly. As a result the probability of failure becomes substantially smaller.

$$P[Success\ Probability\ ] = 1 - P[Failure\ Probability] = 1 - (1-P)^y$$

**Probabilistic Recurrence:-**

This section basically deals with finding of the $k^{th}$ smallest element from a set S of n elements. We pick a random element y from the set S divide the remaining $S^y$ into two sets $S_1$ smaller than and $S_2$ larger than y as in RandQS. Suppose if $(S_1) = k-1$, then y is the desired element. Otherwise if we recursively find the $k^{th}$ smallest element in $S_1$ else we find $k - S_1 - 1$ smallest element in $S_2$. Now

suppose we have to find the expected number of times we pick a random variable or expected number of times we make recursive calls. While this question may not be important for the find algorithm but its important for the analysis of parallel or geometric algorithms. Intuitively we expect that the size of the residual problem is divided by a constant factor at each recursive level, so that the expected number of recursive invocations is $O(\log n)$. Lets generalize this Situation:-
Consider a particle whose position is a positive integer from 1 to n and whose position changes at discreet time steps randomly independently of the past. The process stops only when the particle reaches position 1. If the particle is currently at position $m > 1$, then it jumps to m-X where X is the random variable ranging over 1 to (m-1). All that is known about X is that
$E[X] \geq g(m) \in R$ , $g : R^+ \to R^+$ is a monotone positive non-decreasing function . If the particle starts at n, what is the expected number of jumps before it reaches position 1 and stop ?

**Theorem :-** Let T(n) be the random variable denoting the number of steps(jumps) in which the particle reaches position 1. Then, $E[T] \leq \int_1^n \frac{dx}{g(x)}$

Proof:- Induction on n: n = 1.
Let $f(n) = \int_1^n \frac{dx}{g(x)}$ , for $n \geq 1$.
First Jump: position changes from n to n-X (X taken random variable from 1 to (n-1)).
$E[T(n)] \leq 1 + E[f(n - X)]$
$E[f(n - X)] = E[\int_1^n \frac{dx}{g(x)} - \int_{n-X}^n \frac{dx}{g(x)} = f(n) - E[\int_{n-X}^n \frac{dx}{g(x)}]$
$E[T(n)] \leq (1 + f(n) - E[\int_{n-X}^n \frac{dx}{g(x)}]) \leq (1 + f(n) - \frac{[X]}{g(n)}) \leq f(n)$

**Machine Model :-** Turing Machine Model will be used to discuss complexity theory issues. For describing and analysing algorithms, however we will use RAM (Random Access Machine Models)

**Turing Machine Model :-** A Deterministic Turing Machine is a quadruple $M = (Q, \Sigma, \delta, q)$ . Here Q is the finite set of states and $q \in Q$ is the initial state of the machine. The machine uses finite set if symbols denoted by $\Sigma$ , this includes special symbols of START and BLANK. Symbol $\delta$ in the quadruple M defines the transition function also called as the partial function

$\delta : (QX\Sigma) \to (Q \cup (HALT, YES, NO)X\Sigma X(\leftarrow, \to, STAY))$.
The Machine has three halting states namely HALT(halting state),YES (accepting state) and NO (rejecting state). The input to the machine is written on a tape, unless otherwise specified, machine may read from and write to this tape. If its a single tape machine, then we presume that the read head cannot move to the left. Left end of the tape is fixed, but right end is not (infinite length).

$\delta$(Partial function): Deterministic Turing machine $\rightarrow$ One state possible for one transaction.
$\delta$(Multivariate partial function):Nondeterministic Turing machine $\rightarrow$ More than one state possible for one transaction .
A Turing Machine (shortly abriviated as TM) M accepts $x \in \Sigma^{star}$ iff(if and only if) on input on x ,after finitely many steps M stops and accepts the input (M ends in one of the accepting state/states) . A Turing Machine M rejects $x \in \Sigma^{star}$ iff on input x, after finite number of steps M stops in a non final state/non accepting state.

**Variations of Turing Machine:-** Usually there are many variations of turing machine that can be found, but all of them can be proved to be equivalent of the one we discussed above with a max of quadratic overhead. Some of them have been discussed below.

**Off line K-tape Turing Machine:-** Writes output once computation is over. Usually have seperate tapes for computation (also know as work tapes), perhaps the most common dipiction of Turing Machines. K-tape indicates K work tapes. This type of TM models are a must if we are talking about logarithmic languages or logarithmic complexity classes (L and NL).

**Online Turing Machine:-** When turing machine has to write its output immediately after reading every character on the input tape.
Later we are going to read about another type of turing machines specifically for Randomized algorithms, they are called as **Probabilistic Turing Machines**. But in practise its very hard to program and work with Turing Machine Models. Whatever is possible to do via Turing Machine can be done via RAM and vice-versa (with just polynomial time overhead).

**RAM Model:-** In RAM Model we have a machine that can perform following operations involving registers and main-memory: input-output operations, memory-register transfers, direct and indirect addressing of memory,branching and arithematic operations. In RAM Models, the input tape is read just once but it has an infinite memory to hold the information.

**Language and Decision Problems :-**
if $\Sigma$ denotes the set of alphabets of finite length, with
$1 \leq (\Sigma) < \infty$ , L is a langauge, $L \subseteq \Sigma^{star}$ then A Deterministic Turing Machine M (or RAM) , recognizes Langauge L, iff
*i)* $\forall x \in L$ : M accepts x and
*ii)* $\forall x \notin L$ : M rejects x

**Deterministic Turing Machine :-**

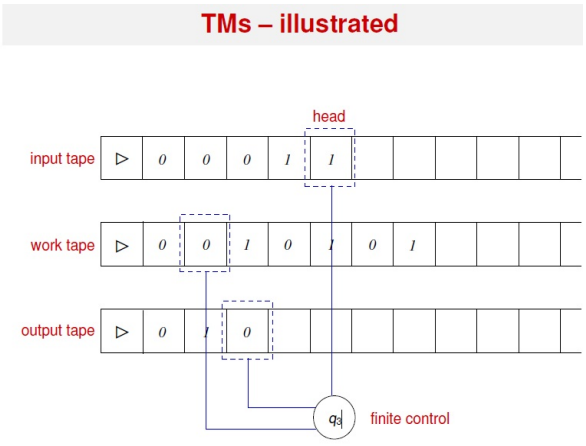



Fig. 1: TMs_illustrated Image

**k-tape Turing Machines**

- **K scratchpad tapes, infinitely long, contain cells**
- One input tape, read-only
- One output tape, write-only
- Working tapes : k heads positioned on individual cells for reading and writing
- **Finite control (finite set of rules)**
- **Vocabulary, alphabet to write in cells**
- **Actions:** depending on symbols under heads, control state. One can move heads (right, left, stay) write symbols into current cells.

**Non-deterministic Turing Machines (NDTM):-**

**Definition (NDTM)**
A non-deterministic TM (NDTM) is a quadruple $M = (Q, \Sigma, \delta, q)$ like a deterministic TM except Q contains a distinguished accepting state $q_{accept}$ .$\delta$ is a pair $(\delta_1, \delta_2)$ of transition functions.
At each step, NDTM non-deterministically chooses to apply either $\delta_1$ or $\delta_2$ . A NDTM M accepts x, M(x) = 1 if there exists a sequence of choices s.t. M reaches $q_{accept}$ and M(x) = 0 if every sequence of choices makes M halt without reaching $q_{accept}$.

## Probabilistic Turing Machine (PTM):-

**Definition (PTM)**
We obtain from an NDTM M = $(\Gamma; Q; \delta_1; \delta_2)$ a probabilistic TM (PTM) by

choosing in every computation step the transition function uniformly at random, i.e., any given run of M on x of length exactly 'l' occurs with probability $2^{-l}$ . A PTM runs in time T(n) if the underlying NDTM runs in time T(n), i.e., if M halts on x within at most T($|x|$) steps regardless of the random choices it makes.

**Definition (DTIME):-**

Let T : N $\rightarrow$ N be a function.$L \subseteq [0,1]^{star}$ is in **DTIME(T)** if there exists a Turing Machine M deciding L in time T' for $T' \subseteq O(T)$ . Here D refers to deterministic. Constants are ignored since TM can be sped up by arbitrary constants.

**Definition (SPACE):-**

Let S : N $\rightarrow$ N and $L \subseteq [0,1]^{star}$. Define $L \subseteq SPACE(S)$ iff there exists a TM M deciding L in no more than S'(n) locations on Ms work tapes ever visited during computations on every input of length n for $S' \subseteq O(S)$.

## Complexity Classes :-

**P (Polynomial Time):-**

Set of class of polynomial time Problems/Algorithms that can be solved in deterministic polynomial time , polynomial being a function of the input length. For every $L \subseteq [0,1]^{star}$ holds: $L \in P$ if and only if there exists a polynomial P : N$\rightarrow$N and a polynomial-time TM M such that for every
$\forall x \in [0,1]^{star}, x \in L$ , M(x) = 1 and $x \in L$, M(x) = 0.

**NP (Non-Deterministic Polynomial Time) :**

Non-Deterministic polynomial time algorithm NP computable with Non-Deterministic Turing Machine in polynomial time. For every L $\forall L \subseteq [0,1]^{star}$ holds: $L \in NP$ if and only if there exists a polynomial p : N $\rightarrow$ N and a polynomial-time Turing Machine M such that for every $\forall x \in [0,1]^{star}, x \in L$ iff $\exists u \in [0,1]^{p(x)}$ : M(x; u) = 1 where M is called verifier and u is called certificate.

**EXP (Deterministic Exponential time) :-**

EXPTIME (also called EXP) is the set of decision problems solvable be deterministic turing machine in $O(2^{p|n|})$ time and denoted as EXP.

**NEXP (Non Deterministic Exponential time) :-**

The complexity class **NEXPTIME** (sometimes called **NEXP**) is the set of decision problems that can be solved by a non-deterministic Turing machine using time $O(2^{p|n|})$ for some polynomial p(n), and unlimited space.In terms of NTIME.

**coNP :-**

A language $L \subset [0,1]^{star}$ is in **coNP** iff there exists a polynomial p and a polynomial time TM M such that $\forall x \in [0,1]^{star}, x \in L$ implies $\forall u \in [0,1]^{p|n|}$ , M(x; u) = 1

**PSPACE(Polynomial Space):-**

In computational complexity theory,**PSPACE** is the set of all decision problems which can be solved by a Turing machine using a polynomial amount of space. Its denoted as ????????????

**NSPACE(Non Deterministic Polynomial Space) :-**

In computational complexity theory, the complexity class **NSPACE**(**f**(**n**)) is the set of decision problems that can be solved by a non-deterministic Turing machine using space $O(f(n))$, and unlimited time. It is the non-deterministic counterpart of DSPACE. Its denoted as **NPSPACE** = ?????????????

**Space Hierarchy Theorem:-**

**Theorem (Space Hierarchy)**
Let f ; g : N → N be space-constructible such that $f \in o(g)$ . Then $SPACE(f(n)) \subset SPACE(g(n))$ inclusion is strict, stronger theorem than corresponding time theorem only constant space overhead. f ; g can be logarithmic too.

**Time vs. Space :-**

Let s : N → N be space-constructible. Then
$DTIME(S(n)) \subseteq SPACE(S(n)) \subseteq NSPACE(S(n)) \subseteq DTIME(2O(S(n)))$

**Theorem (Savitch) :-**

For every space-constructible S : N → N with $S(n)$ . Savitch: non-deterministic space can be simulated by deterministic space with quadratic overhead (by path enumeration in configuration graph).

**Read-once Certificates (Special case for NL):-** Similar to NP, also NL has a characterization using certificates.

**Theorem (read-once certificates)**
$L \subset [0,1]^{star}$ is in NL iff there exists a deterministic logspace TM M (verifier) and a polynomial p : N → N such that for every
$\forall x \in [0,1]^{star}$ iff $\exists u \in [0,1]^{p(|n|)}$ :M(x; u) = 1.
Certificate u is written on an additional read-once input tape of M.

example: path in a graph is a read-once certificate.

**Configuration Graphs :-**

Let M be a deterministic or non-deterministic TM using s(n) space. Let x be some input. This induces a configuration graph G(M; x) with

- nodes are configurations:
- state
- content of work tapes
- edges are transitions (steps) that M can take.

**Properties of configuration graph**

Outdegree of G(M;x) is 1 if M is deterministic; 2 if M is non-deterministic. G(M;x) has at most $((Q)) * I^{c*S(n)}$ nodes (c some constant) which is in $2^{s(n)}$ . G(M; x) can be made to have unique source and sink with existence of acceptance of path from source to sink which can be checked in time O(G(M; x)).
$NSPACE(S(n)) \subseteq DTIME(2^{S(n)})(usingBFS)$.
**Footnote :-** This argument holds good as long as $SPACE(n) = \omega(\log n)$ , if we are below logn, then the situation becomes more complicated.
if $NSPACE \leq 3.5n^{7.3}$, then PSPACE is at most $(NSPACE)^2$.
Result NSPACE = PSPACE
Logarithmic Space :- Deterministic logarithmic space normally denoted as L and non-deterministic logarithmic space denoted as NL. Its an open question weather L = NL ? $(\log)^2$ Or $(\log)^n$ are called polylog space.
NSPACE and NL are closed under complement. For deterministic classes its easy, since we can flip the output. i.e. NL = coNL and NSPACE = coNSPACE .

**Theorem (Immerman-Szelepcsnyi):**
For reasonable $s(n) \geq \log n, NSPACE(s(n)) = co - NSPACE(s(n))$.

**Proof:-** Let M be a nondeterministic machine using s(n) space. We will create a nondeterministic machine N such that for all inputs x, N(x) accepts if and only if M(x) rejects.
Fix an input x and let s=s($|x|$). The total number of configurations of M(x) can be at most cs for some constant c. Let t=$c^s$. We can also bound the running time of M(x) by t because any computation path of length more than t must repeat a configuration and thus could be shortened.
Let I be the initial configuration of M(x). Let m be the number of possible configurations reachable from I on some nondeterministic path. Suppose we knew the value of m. We now show how N(x) can correctly determine that M(x) does not accept.
Let r=0, For all nonaccepting configurations C of M(x). Try to guess a computation path from I to C.
If found let r=r+1.
If r=m then accept o.w. reject. If M(x) accepts then there is some accepting configuration reachable from I so there must be less than m non-accepting configurations reachable from I so N(x) cannot accept. If M(x) rejects then there

is no accepting configurations reachable from I so N(x) on some nondeterministic path will find all m nonaccepting paths and accept. The total space is at most O(s) since we are looking only at one configuration at a time. Of course we cannot assume that we know m. To get m we use an idea called inductive counting. Let mi be the number of configurations reachable from I in at most i steps. We have m0=1 and mt=m. We show how to compute mi+1 from mi. Then starting at m0 we compute m1 then m2 all the way up to mt=m and then run the algorithm above. Here is the algorithm to nondeterministically compute mi+1 from mi. Let mi+1=0
For all configurations C
Let b=0, r=0
For all configurations D
Guess a path from I to D in at most i steps
If found
Let r=r+1
If D=C or D goes to C in 1 step
Let b=1
If $r < mi$ halt and reject
Let mi+1=mi+1+b
The test that $r < mi$ guarantees that we have looked at all of the configurations D reachable from I in i steps. If we pass the test each time then we will have correctly computed b to be equal to 1 if C is reachable from I in at most i+1 steps and b equals 0 otherwise.
We are only remembering a constant number of configurations and variables so again the space is bounded by O(s). Since we only need to remember mi to get mi+1 we can run the whole algorithm in space O(s).

## Open and known problems

**OPEN/UNKNOWN :-**

**P = NP?**
**NP = coNP?**

**KNOWN**

- if an NP-complete problem is in P, then P = NP $P \subseteq coNP \cap NP$
- if $L \in coNP$ and L is NP-complete then NP = coNP
- if P = NP then P=NP=coNP
- if $NP \neq coNP$ then $P \neq NP$
- if $exp \neq NEXP$ then $P \neq NP$ (equalities scale up, inequalities scale down)

**Reductions :-**

Instead of tight bounds say which problem is harder we use reductions . IF there is an efficient procedure for problem A and an efficient procedure for B using the procedure for A
**THEN** B cannot be radically harder than A
**notation:** $B \leq A$ , the important point to note is that, reductions cannot be more powerful than the problem class itself. Therefore for NP complete or PSPACE complete problems, only poly time reductions should be applied. For L and NL problems logrithmic reductions should be applied as polynomial reductions are too powerful for that class of problems.

**Logspace reductions:-**

**Definition (logspace reduction)** Let $L', L \subseteq [0,1]^{star}$ be languages. We say that L is logspace-reducible to L', written $L \leq \log L'$ if there is a function f : $[0,1]^{star} \rightarrow [0,1]^{star}$ computed by a deterministic TM using logarithmic space such that $x \in L$ implies $f(x) \in L'$ and $((f(x))) \in O((((x)))^c)$ for some constant $c > 1$ for every $x \in [0,1]^{star}$.
L logspace reduction is transitive. If C is logspace reducible to B $C \in L$ this imples $B \in L$ NL-hardness and NL-completeness can be defined in terms of logspace reductions.

**Many-One Reductions:-**

In computability theory or computational complexity theory, a many-one reduction is a reduction which converts instances of one decision problem into instances of another decision problem. Reductions are thus used to measure the relative computational complexity/difficulty of the two problems. We apply translation function to a problem A to take it to another problem B assuming we know a procedure to solve problem B.Many-One reductions are also called as karp-reduction.
Many-one reductions are special case and a stronger form of Turing Reductions. With many-one reductions the oracle can be invoked only once at the end and the answer cannot be modified.

**Oracle:-** Questions can be asked in polytime and all answers can be obtained in polytime. All membership questions can be asked in a single step. This is also called as Turing reduction where many questions can be asked(all in single step). Its like a sub-routine call. Turing reductions are also called cook-reduction (Inventor of NP).

Ex 1:- CVP or Circuit Value Problem $\rightarrow$ Given the description of the circuit with $(x_1, x_2, ...x_n) \in [0,1]^n$ ,determine the output of the problem. Its a P-Complete problem in logspace.
Ex 2:- 3-SAT or SAT is a NP-Complete problem (3-CNF).

Ex 3:- QBF is a PSAPCE complete problem.

**Proof:-**
**Definition (QBF)**
A quantified Boolean formula is a formula of the form $Q_1x_1Q_2x_2....Q_nx_n\psi(x_1, x_2, ...x_n)$

- where each $Q_i \in [\forall, \exists]$
- each $x_i$ ranges over [0,1]
- $\psi$ is quantifier-free.
- wlog we can assume prenex form.Formulas are closed, ie. each QBF is true or false.
- if all $Q_i = \exists$ , we obtain SAT as a special case
- if all $Q_i = \forall$ , we obtain Tautology as a special case

**QBF is in PSPACE**

Polynomial space algorithm to decide QBF with n variables and size m.
$qbfsolve(\Psi)$ if $\Psi$ is quantifier-free
return evaluation of $\Psi$
if $\Psi = Qx.\Psi'$
if $Q = \exists$
if $qbfsolve(\Psi'[X \to 0])$ return true
if $qbfsolve(\Psi'[X \to 1])$ return true
if $Q = \forall$
$b1 = qbfsolve(\Psi'[X \to 0])$
$b2 = qbfsolve(\Psi'[X \to 1])$
return $b1^b2$
return false Each recursive call can re-use same space,therefore qbsolve uses at most O(n + m) space This proves $QBF \in PSPACE$
Now we need to show that every problem $L \in PSPACE$ is polynomial-time reducible to QBF.
**Proof:-**
Let $L \in PSPACE$ arbitrary. $L \in SPACE(S(n))$ for polynomial s. $m \in O(S(n))$ bits needed to encode configuration C.
There exists Boolean formula $\psi M, x$ with size O(m) such that $\psi M, x(C, C') = 1$ iff $C, C' \in [0, 1]^m$ encode adjacent configurations; see Cook-Levin Define QBF $\Psi$ such that $\Psi(C, C')$ is true iff there is a path in G(M, x) from C to C'. $\Psi(Cstart, Caccept)$ is true iff M accepts x.
Define $\Psi$ inductively !.
$\Psi i(C, C')$ : there is a path of length at most $2^i$ from C to C'.
$\Psi = \Psi m$ and $\Psi 0 = \psi M, x$
$\Psi i(C, C') = \exists C''.\Psi i - 1(C, C'') \wedge \Psi i - 1(C'', C')$
might be exponential size, therefore use equivalent
$\Psi(C, C'') = \exists C''.\forall D1.\forall D2.((D1 = C \wedge D2 = C'') \vee (D1 = C'' \wedge D2 = C'))$

$\Rightarrow \Psi i - 1(D1, D2)$

C" stands for m variables $\Rightarrow (\Psi i) = (\Psi i - 1) + O(m) \Rightarrow (\Psi) \in (m^2)$

**Property of reduction :-**

If a problem is complete for lower notion, then it will always be complete for higher notion. There are also languages between two classes(not always). e.g. GI(Graph Isomorphism). its not NP-Complete but it is also not in P.

We define NP normally in terms of non-deterministic computation tree, which is a perfect ninary tree and whose height is $P(|X|)$ which is polynomial in the length of the input .
Details of computation tree (Configuration graph) has already been provided above. If atleast one computation path gives an accepting path, then its accepting else rejecting.

**SATISFIABILITY(SAT):-**

Input is a prepositional formula (CNF). Can we get a satisfying assignment? Its a mother problem for NP-Complete class. We get the certificate from oracle and plugging in the values to check if its a one '1', we have satisfying assignment.

$SAT = \{\exists C_1 \exists C_2 .....\exists C_m \in 0, 1^m : F(C_1, C_2, ...C_m) = 1$ where F is the prepositional formula .
$CVP = \{\exists X_1 \exists X_2 .....\exists X_m \in 0, 1^m : P(X_1, X_2, ...X_m)$ where the values of inputs $X_1, X_2..X_m$ have been provided as well as the description of the circuit, find the ouput value of P .

$\rightarrow$ For Non-deterministic problem paradigm is guess and proof.
$\rightarrow$ For Deterministic problem paradigm is create and proof.

**Probabilistic Complexity Classes:-**

Probabilistic complexity classes includes problems which uses random bits for its execution along with regular problem description and the input. Randomness can make possible computation tasks that are probably impossible without it. Cryptography is a good example: if the adversary can predict the way you generate your keys, you cannot encrypt messages. Randomness is also good for breaking symmetry and making arbitrary choices. For solving or analysing or even sometimes putting the definition of Probabilistic Complexty classes we use the notion Probabilistic Turing Machine or PTM (defined above).

**Recap NP:-**

Lets have one more look at NP and then we move to towards Probabilistic classes taking Randomized NP as the basis.
For every$L \forall L \subseteq 0,1^{star}$ holds: $L \in NP$ if and only if there exists a polynomial p : N $\rightarrow$ N and a polynomial-time Turing Machine M such that for every $\forall x \subseteq 0,1^{star}, c \in L$ iff $\exists u \in 0,1^{p((x))}$ : M(x; u) = 1 where M is called verifier and u is called certificate.

**NP captures the class of possibly (not) tractable computations:**

- Certificate 'u' is a satisfying assignment
- Dont know how to compute u in poly-time, but
- If there is a u, then is polynomial in , and
- We can check in poly-time if a u is a certificate/solution.
- Deemed untractable. Conjecture: $P \neq NP$.

**Randomizing NP :-** Obtain from NP a more realistic class by randomization $\rightarrow$ Choose u uniformly at random from $0,1^{P(X)}$ .

**Definition (Accept/Reject certificates and probabilities) :-**

Fix some $L \in NP$ decided by M using certificates u of length
$P(|x|) : A_{M,x} = u \in 0,1^{P((x))} : M(X,u) = 1$ and $R_M, x = u \in 0,1^{P((X))} : M(X,u) = 0$ :
If we choose uniformly at random:

- $A_{M,x}$ is the event that u says accept x.
- $R_{M,x}$ is the event that u says reject x.

**Definition (Accept/Reject certificates and probabilities (contd)):-**

$Pr[A_{M,x}] = \frac{A_{M,x}}{2^{P(|x|)}} 2p(jxj)$ and $Pr[R_{M,x}] = Pr[R_{M,x}] = \frac{R_{M,x}}{2^{P(|x|)}} = 1 - Pr[A_{M,x}]$
$L \in NP_{iff} \forall x \in 0,1^{star}$
$x \in L \Rightarrow \S Pr[A_{M,x}] \geq 2^{-P(|X|)}$ and $x * LPr[A_{M,x}] = 0$

**Examples of randomized algorithms :-**

- Randomized poly-time with one-sided error: RP; coRP; ZPP
- Power of randomization with two-sided error: PP; BPP

**Definition (Randomized P (RP)) :-**

$L \in RP$ if there exists a polynomial p : N→ N and a polynomial-time TM M(x,u) using certificates u of length $(u) = p(|x|)$ such that for every $x \in [0,1]^{star}$
$x \in L \Rightarrow Pr[A_{M,x}] \geq 1/2$ and $x * LPr[A_{M,x}] = 0$
→ $P \subseteq RP \subseteq NP$
→ $coRP = L : L \in RP$
→ RP remains unchanged if we replace $(1/2)byn^{-k}$ or §(3/4)

**One-sided error probabiliy for RP:**

→ False negatives: if $x \in L$, then $Pr[R_{M,x}] \leq (1/2)$ or $(1/4)$ (depending on acceptance probability is $(1/2)$ or $(3/4)$.
→ If M(x, u) = 1, output $x \in L$; else output probably, x*L
→ Error reduction by rerunning a polynomial number of times.
Given input x , Choose $u \in [0,1]^{p(|x|)}$ uniformly at random.
→ Run M(x, u).
→ If M(x, u) = 1, output: yes, $x \in L$ .
→ If M(x, u) = 0, output: probably, x*L , Called Monte Carlo algorithm.

**Definition(coRP) :-**

$L \in coRP$ if and only if there exists a polynomial P: N→N and a polynomial-time TM M(x,u) using certificates u of length
$(u) = p(|c|)$ such that for every $x \in [0,1]^{star}, x \in L$
$x \in L \Rightarrow Pr[A_{M,x}]= 1$ and $x * L \Rightarrow Pr[A_{M,x}] \leq (1/2)$ or $(1/4)$ ( depending on which value we choose for RP acceptance and rejection):

**One-sided error probability for coRP:**

→ False postives: if $x * L$, then $Pr[A_{M,x}] \leq (1/4)$ → If M(x, u) = 1, output probably,$x \in L$ ; else output $x * L$ The class RP (for Randomized Polynomial time) consists of all languages L that have a randomized algorithm A with worst case polynomial running time such that for any input $x \in \sum^{star}$,
→ $x \in L \Rightarrow Pr[A(x)accepts] \geq (1/2)$
→ $x \in L \Rightarrow Pr[A(x)accepts] = 0$
An RP algorithm is Monte Carlo, but the mistake can only be if $x \in L$ . coRP is all the languages that have a Monte Carlo algorithm that make a mistake only if $x.L$ . A problem which is in RP\coRP has an algorithm that does not make a mistake, namely a Las Vegas algorithm.

**Definition (Zero Probability of Error-P (ZPP)) :-**

→ $ZPP = RP \wedge coRP$. If $L \in ZPP$, then we have both an RP-TM and a coRP-TM for L.

The class ZPP ( for Zero-error Probabilistic Polynomial time) is the class of languages that have Las Vegas algorithms in expected polynomial time.
**Assume** $L \in ZPP$
Then we have Monte Carlo algorithms for both $x \in L$ and $x \in \bar{L}$ .
Given x: -
→ Run both algorithms once.
→ If both reply probably, then output dont know.
→ Otherwise forward the (unique) yes-reply.
→ Called Las Vegas algorithm.
Till Now we have seen algorithms (or class of problems) where we have allowed only one sided error. But now we will consider algorithms which has two sided error i.e Probability of error for both $x \in L$ and $x * L$

**RP obtained from NP by**
→ choosing certificate u uniformly at random
→ requiring a fixed fraction of accept-certificates if $x \in L$
$x \in L \Rightarrow Pr[A_{M,x}] \geq (1/2)$ and $x * L \Rightarrow Pr[A_{M,x}] = 0$
RP-algorithms can only make errors for $x \in L$
By allowing both errors for both cases, can we obtain a class that is larger than RP,
but still more realistic than NP?
Assume we change the definition of RP to:

$x \in L \Rightarrow Pr[A_{M,x}](1/2)$ :

**Two-sided error probabilities:**
→ False negatives: If $x \in L : Pr[R_{M,x}] \leq (1/2)$
→ False negatives: If $x * L : Pr[A_{M,x}] \leq (1/2)$
Outputs: probably, $x \in L$ and $x * L$ probably,

**Probabilistic Polynomial Time (PP):-**

$L \in PP$ if there exists a polynomial P:N→N and a polynomial-time TM M(x,u) using certificates u of length u = $P(|x|)$ such that for every $x \in 0, 1^{star}$
$x \in L \Leftrightarrow Pr[A_{M,x}] \geq (1/2)$ :
$RP \subseteq PP \subseteq exp$
→ One May replace $\perp$ "greater than or equal to" by "strictly greater than" and vice-versa.
→ One May replace probability (1/2) by (3/4) .
→ **PP = coPP**
→ PP: $x \in L$ iff x is accepted by a majority
→ $x * L$, then x is not accepted by a majority (which is not same as majority rejects x!)

## Randomness and Non Uniformity

A basic issue in the study of randomized algorithms is the extent to which randomized is necessary for solving a problem. When is it possible to remove the randomization in a randomized algorithm? The answer depens on aspects of problem being solved. The goal of this section is to show that the question is more subtle then it appears at first, and touches on the issue of uniformity algorithms. We now study the notion of a randomized circuit, and a general technique by which randomization can be removed in polynomial sized randomized circuits.

A boolean circuit with n inputs is a directed acyclic graph with the following properties:

- There are n vertices of in degree 0; These are called the inputs to the circuit and labeled $x_1, x_2, ...x_n$. There is one vertex with out degree 0; this is called output of the circuit.

- Every vertex v that is not an input or the output is labeled with one boolean funtion b(v) from the set AND,OR,NOT. A vertx labeled NOT has in degree 1.

- Every input to the circuit is assigned b boolean value. Under such an assignment of input values, each vertex v computes the Boolean function b(v) of the values on incoming edges, and assign the value to its outgoing edges. The value of the outpu is thus a Boolean function of $x_1, x_2, ...x_n$; the circuit is said to compute this function.

- The size of circuit is the number of vertised in it.

Except that there may be more than n vertices of in degree() and these are partitioned into two classes;
(1) Randomised inputs, an independent rnadom value 0,1 and
(2) The n circuit inputs, $x_1, x_2, ...x_n$.

$f_n$ the function f restricted to inputs from $0, 1^n$; A sequence C = $C_1, C_2$.. of circuit family. If $C_n$ has inputs and computes $f(x_1, x_2, ...x_n)$ at its output for all inputs $x_1, x_2, ...x_n$. The family C is said to be Plolynomial sized if the size of $C_n$ is bounded above by p(n) for every n, where p(.) is a polynomial. A randomized circuit family for f is circuit family for f that, in addtion to the n inputs $x_1, x_2, ...x_n$, takes m randon bits $r_1, .....r_m$ reach of the equiprobability 0 or 1. In addition for every n, circuit $C_n$ must satisfy two properties:

- If $f_n(x_1, x_2, ...x_n) = 0$, then the output of the circuit is 0 regardless of the values of random inputs $r_1...r_m$.

– If $f_n(x_1, ..., x_n) = 1$, then the outpur of the circuit is 1 with the probability at least 1/2. In other words, at least one of the $2^m$ choices of the bits $r_1, ...r_m$ will result in circuit evaluation to 1. We will refer to such -tuplets $r_1, ....r_m$ as witness for $(x_1, x_2, ...x_n)$ in that they testify to the correct value of $f_n(x_1, x_2, ...x_n)$, when it is 1.

**Adleman's Theorem**

From a matrix M with rows $2^n$ rows, one for each possible input from $0, 1^n$. The matrix has $2^m$ columns, one for each of the possible m-tuples from $0, 1^m$ that tha $r_i$ can assume. The entry $M_{ij}$ is 1 if the settiong of the $r_1, ....r_m$ corresponding to column k is a witness for the $x_1, x_2, ...x_n$ corresponding to the row j; otherwise the entry is 0. Eliminate all rows of M corresponding to inputs for which $f_n$ evaluates 0.

By definition, at least half the entries of every surviving row of the M equal 1. Therefore, there must be a column with at least half its entries 1. Therefore, there must be a column with at least half its entries 1; in other words 1; in other words. there is an assignement of 0s and 1s to the $r_i$ that serves as witness to at least half of the possible inputs. Let this witness be $r_1(1), .....r_m(1)$. Delete the column in M corresponding to $r_1(1), .....r_m(1)$, and all rows 1s in this column. Thus $T_1$ computs the correct value of $f_n(x_1, x_2, ...x_n)$ whenevr the input corresponds to one od the rows we have just eliminated. W have delete all the rows of M while building at most n circuits $T_1, .....T_n$.

Above mentioned is the first example we have seen of derandomization, where we take a Randomized algorith or compuatition, or computation and diminish or entierly remove the randomness in it. This is often a useful technique for the design of deterministic algorithms. Does theorem mean that randomization is dispensible in all polynomial-time computation? The answer is no, and has to do with the issue of non-uniformity in computation. The deterministic circuit-generated by the above procees is one that works for a particular value of n. Indeed, the circout it produces for n inputs may have very little rsemblance to the circuit it produces for n+1 inputs, even if the original randomized circuit were similar. Any "practical" algorithm of circuit will in face exhibit this property of similarity, which is formalized in the literature under the name uniformity.

**Adviced class**

Complexity theory formalizes this intuition by classifying algorithms as being uniform or non-uniform as follows. Let a(n) be a function from the positive integres to string in $\sum^*$. An algorithm A is said to use advice a if on an input of length n it is given the string a(n) on a read only tape. We say that A decides a language L with advice a if on an input x it uses the read onle string $a(|x|)$ to decide the membership of x in L. In other words, a single advise string a(n)

enable the algorithm A to decide the membership of x in L for all inputs x having length n. Uniform algorithms are those that use no advice strings at all, whereas non-uniform algorithms are those that use advice. For the complexity class P, we define the class P/ poly to consist of all languages L that have non uniform polynomial time algorithm A such that the length of the advice string a(n) is bounded by a polynomial in n. Likewise, we may define the RP/ poly.

## 1 Moments and Derivation

In analysing the performance of a randomized algorithm, we often like to show that the behavious of the algorithm is good almost all the time. e.g. is more importnant (desirable) to show that the running time is small with high probability , not just that it has a small expectation. In the study of occupancy problems, the motivation is to study general bounds on probability that a Random variable diviates fas from its expectation, enabling it to avoid such custom node analysis. The probability that a random variable deviates by a margine n amount from its expectation is suffered to as a fail probability for derivation.

### Occupancy Problems

Family of stochastic processes that is fundamental to the analysis of many randomized algorithms are called occupancy problems. It provides for the general bounds on the probability that a random variable deviates far from its expectation, enabling us to avoid such a custom mode analysis.
The probability that a random variable deviates by a given amount from its expectation is suffered to a fail probability for the derivation.

### Occupancy Problems*****

We anvision each of m is distinguishable objects ("balls") is being randomly assigned to one of n distinct classes ("bins"), i.e each ball is placed in a bin chosen independently and uniformly at random. Our discussions of the occupancy problem will illustrate ***** in the analysis of randomized algorithms. "The probability of the union of ***** is no more than the sum of their probabilities." For arbitary elements $k_1, k_2, ....k_n$ not necessarily independent.

$$P_r[\bigcup_{i=1}^{n} h_i] \leq \sum_{i=1}^{n} P_r[h_i]$$

Note that this principle is extremely useful because it assumes nothing about dependencies between the units. Thus, it enables us to analyze phenomenon involving ends with very complicated interactions, without having to ***** the variables.

Consider the first case m=n for $1 \leq i \leq n$, let $X_i$ be the number of balls in the $i^{th}$ bin. We have $E[X_i] = 1$ for all i, yet we do not expect that during a typical experiemnt every bin receives exactly one ball. Rather, we expect some bins to have many balls than one and some bins to have no balls at all.
Let us now try to make a statement of the form, "with very high probability, no of bins receives more than k balls" for a suitably chosen k.let $\varepsilon_j(k)$ denote the **** the bin j has k or more balls in it. We concentrate on $\varepsilon_j(k)$. The probability that bin 1 has exactly i balls

$$\binom{n}{i}(1/n)^i(1-1/n)^{n-i} \leq \binom{n}{i}(1/n)^i \leq (ne/i)^i(1/n)^i \leq (e/i)^i$$

Probability that bin 1, has k or more balls

$$\begin{aligned} P_r[\varepsilon_1(k)] &\leq \sum_{i=k}^{n}(e/i)^i \\ &\leq (e/k)^k[1 + (e/k) + (e/k)^2 + .....] \\ &\leq (e/k)^k[\frac{1}{1-(e/k)}] \end{aligned}$$

let k' be $k' = \lceil \frac{3ln(n)}{lnln(n)} \rceil$
let the swiftly chosen L ( defined by k*) is K = (k*) = $[\frac{3lnn}{lnln(n)}]$
$P_r[\varepsilon_1(k*)] \leq (e/k*)^{k*}[\frac{1}{1-(e/k*)}]$
$e/k* = \frac{e}{\frac{3ln(n)}{lnln(n)}}$ ,since $e = 2.73 \leq 3, ln(n) \leq lnln(n)$ [grows much slower]

$0 < (e/k*) < 1 \Rightarrow 0 < 1 - r/k* < 1$
$1/(1 - e/k*) > 1$, Lets consider worst case value to be 2.

$$\begin{aligned} P_r[\varepsilon_1(k*)] &\leq 2.(e/k*)^{k*} \\ &\leq 2.[\frac{e}{\frac{3ln(n)}{lnln(n)}}]^{k*} \\ &\leq 2.[\frac{e.e^{lnlnln(n)}}{e^{ln(3lnln)}}]^{k*} \\ &\leq 2.[e^{1-ln(3lnln)+lnlnln(n)}]^{k*} \\ &\leq 2.[e^{1-ln(3)+lnln(n)+lnlnln(n)}]^{k*} \\ & ln(3) > 1(= lne) \\ &\leq 2.[e^{-lnln(n)+lnlnln(n)}]^{k*}, \text{ for sufficiently large n} lnlnln(n)/lnln(n) < 1/3 \\ &\leq 2.exp[-lnln(n) + lnlnln(n)].\lceil\frac{3ln(n)}{lnln(n)}\rceil \\ &\leq 2.exp[3ln(n) + \frac{3ln(n)}{lnln(n)}.ln(n)] \\ &\leq 2.exp[-2ln(n)] \leq 2.(n-2) \leq 2/n^2 \end{aligned}$$

This is the probability that $\varepsilon_1(k)$ = bin 1 for k or more balls in it $P_r[\varepsilon_1(k)] \leq 2/n^2 \leq n^{-2}$
The same equation tells us that the upper bound applies to $P_r[\varepsilon_i(k*)]$ for all i

$$P_r[\bigcup_{i=1}^{n} \varepsilon_i(k*)] \leq \sum_{i=1}^{n} P_r[\varepsilon_i(k*)] \leq \sum_{i=1}^{n} n^{-2} \leq (1/n)$$

The probability that no bin has more than$k* = \lceil \frac{3ln(n)}{lnln(n)} \rceil$ balls
$P_r[$ no bin contains more than $k* = \lceil \frac{3ln(n)}{lnln(n)} \rceil] = P_r[\bigcup_{i=1}^{n} \varepsilon_i(k*)]$
$\geq 1 - n/n^2 = 1 - 1/n$ [can also be given as 1 -2/n].

**Additional info**

Suppose that m balls one randomly assigned to n bins. We study the probability of the event that all th balls have been assigned to distinct bins. The special case n = 365 is called as birthday problem, with 365 days of the year (no leap year) corresponds to 365 bins. How long must m be before two people in the group are likely to share their birthdays?

Consider the assignement of the balls to the bins as a sequential process we throw first ball into a random bin, then second and so on. for $2 \leq i \leq m$, ler $\varepsilon_1$ be the event that ith ball gets in a bin, not containing any of first i ball.

$P_r[\varepsilon_1 \mid \bigcap_{j=2}^{i-1} = [n - (i-1)/n]$.[left over balls]/ [total no of balls] $= [1 - (i-1)/n]$

$$P_r[\bigcap_{i=2}^{m} \varepsilon_i] = P_r[\varepsilon_2 \prod_{i=3}^{m} P_r[\varepsilon_i \mid \bigcap_{j=3}^{i-1} \varepsilon_j]$$

$\Rightarrow P_r[\bigcap_{i=2}^{m} \varepsilon_i] \leq \prod_{i=2}^{m}(1 - (i-1)/n)$
It is easy to compute $P_r[\varepsilon_i \mid \bigcap_{j=2}^{i-1} \varepsilon_j]$, since this is simply the probability that the ith ball lands in an empty bin given that the first (i-1) all fell into distinct bins and it is easy to see,
$P_r[\bigcap_{i=2}^{m} \varepsilon_i] = [\frac{n-(i-1)}{n}] = [1 - (i-1)/n]$
$P_r[\bigcap_{i=2}^{m} \varepsilon_i] \leq \prod_{i=2}^{m}[1 - (i-1)/n]$ ,use identity 1- x $\leq e^{-x}$.

$$P_r[\bigcap_{i=2}^{m} \varepsilon_i] \leq \prod_{i=2}^{m} e^{-(i-1)/n}$$
$$\leq e^{-\sum_{i=2}^{m}(i-1)/n} \leq e^{-1/n \sum_{i=1} m-1i}$$
$$\leq e^{-1/n.m(m-1)/2} \leq e^{-m(m-1)/2n},$$

Let the value of $m = \lceil \sqrt{2n} + 1 \rceil$, the probability that all the m balls fall in n different bins

$$\leq e^{(-m^2+2m)/2n}$$
$$\leq exp[(-(2n + 1 + 2\sqrt{2n}) + 2(\sqrt{2n} + 1))/2n]$$
$$\leq exp[(-2n - 1 + 2)/2n] = exp[(-2n + 1)/2n]$$
$$\leq exp(-1) \leq 1/e$$

As m increases, the probability drops drastically.

**Markov and Chebyshev inequalities:**

Expectation: Let X be a discrete random variable and f(x) be any valued function. Thus the explanantion of f(x) is given by
$E[f(x)] = \sum_x f(x).P_r[X = x]$
$E[X] = \sum_x x.P_r[X = x]$
$\sigma_x$ = standard derivation
Variance = $E[X^2] - (E[X])^2$
Variance of $X = {\sigma_X}^2 = E[(X - E[X])^2]$
$= E[X^2 + E(X)^2 - 2XE(X)]$
$= E[X^2] + E[X]^2 - 2E[X]E[X] = E[X^2] - E[(X)]^2$
Theorem:- let X be a random variable assuming only non-negative values ($\geq 0$), Therefore all $t \in IR^+, P_r[X \geq t] \leq E[X]/t$
Proof:-

$$f(X) = \begin{cases} 0 & \text{for } x < t \\ 1 & \text{otherwise } (i.e. x \geq t) \end{cases}$$

$E[f(x)] = \sum_x f(x).P_r[X = x]$
$\sum_{x \geq t} f(x).P_r[X = x] + \sum_{x<t} .P_r[X = x]$ [Definition off(x)]
$E[f(x)] = \sum_{x \geq t} P_r[X = x] = P_r[X \geq t]$
Since $f(x) \geq x/t$
$P_r[X \geq t] \leq E[E/t] \leq E[X]/t$
This is the highest possible bound when we know only that X is non negative and has a given expectation. Unfortunately, the Markov inequallity by itself is often too weak to yield good results.
$k \in N$
$P_r[X \geq k.E[X]] \Rightarrow k.E[X] = 1 \Rightarrow E[X] = 1/k$
$P_r[X \geq 1] = P_r[X = 1] = E[X] = 1/k$
$\Rightarrow t \in IR^+, P_r[h(y) \geq t] \leq E[R(y)]/t.....X = h(y)$

$P_r[X \geq t] \leq E[X]/t$
$\Rightarrow P_r[h(y) \geq t] \leq E[h(y)]/t$. We now show that the Markov inequality can be used to drive better bounds on the said probability by using more information about the distribution of the random variable.

**Chebyshev Bound**

Based on the knowledge of the variance of distribution for a random variable X,
$\mu_X = E[X]$ ( expectation) . variance = ${\sigma_X}^2 = E[(x - \mu_X)^2]$
$\sigma_X$ = Std. deviation of X *Chebyshev inequality* :- let X be the random variable with
$\mu_X = E[X], {\sigma_X}^2 = E[(x - \mu_X)^2], \sigma_X = S.D$
$P_r[|X - \mu_X| \geq t\sigma_X] \leq 1/t^2$
Proof :-$P_r[|X - \mu_X| \geq t\sigma_X] = P_r[|X - \mu_X|^2 \geq t^2{\sigma_X}^2]$
$Y$(random variable) $= (|X - \mu_X|)^2$
$E[Y] = E[(|X - \mu_X|)^2] = (\sigma_X)^2$
from Markov's inequality
$P_r[Y \geq t^2{\sigma_X}^2] \leq E[Y]/(t^2{\sigma_X}^2) \leq {\sigma_X}^2/(t^2{\sigma_X}^2) \leq 1/t^2$
$\Rightarrow E[|X - \mu_X| \geq t] \leq {\sigma_X}^2/(t^2)$
$E[X^k] = k^{th}$ normal of X.
$\mu = E[X] =$ 1st moment of X.
$E[(|X - \mu|)^k] = k^{th}$ central moment.

**pg 6**

Let X and Y be the random variable and f(x,y) be a function of two variables x and y. Then
$E[f(X,Y)] = \sum_{x,y} f(x,y).P(x,y)$
if x and y are val independent random variable. $P(x,y) = P(x \cap y) = P(x).P(y)$
$f(x,y) = f(x \cap y) = f(x).f(y)$
$E[f(x,y)] = \sum_{x,y} f(x,y).P(x,y) = \sum_x \sum_y f(x).f(y).P(x).P(y)$
$= \sum_x f(x).P(x) \sum_y f(y).P(y) = E[X].E[Y]$
$X = \sum_{i=1}^m X_i$then ${\sigma_X}^2 = \sum_{i=1}^m {\sigma_{x_i}}^2$

${\sigma_X}^2 = E[X - {\mu_X}^2] = E[X^2 + {\mu_X}^2 - 2\mu_X X]$
$= E[(X^2)] + E[{\mu_x}^2] - 2E[X]\mu_X$
$= E[(X^2)] - {\mu_X}^2 = E[(X^2)] - E[(X)]^2$

$\mu_X = \sum_{i=1}^m \mu_{x_i}$[Linearity of expectation]
$X = \sum_{i=1}^m X_i \Rightarrow E[X] = \sum_{i=1}^m E[X_i]$
$\mu_X = E[X], \mu_{x_i} = E[X_i]$

$X^2 = (\sum_{i=1}^m X_i)^2$
$\Rightarrow {\sigma_X}^2 = E[X - {\mu_X}^2] = E[(\sum_{i=1}^m X_i - \sum_{i=1}^m \mu_{x_i})^2]$

$= E[(\sum_{i=1}^{m}(X_i - \mu_i))^2]$
$E[\sum_{i=1}^{m}(X_i - \mu_i)^2] + 2\sum_{i=1}^{m}\sum_{j>i}(X_i - \mu_j)(X_i - \mu_j)$
$= E[\sum_{i=1}^{m}(X_i - \mu_i)^2] + 2\sum_{i=1}^{m}\sum_{j>i}^{n} E[(X_i - \mu_j)(X_i - \mu_j)]$

Since $\forall X_i, Y_i$ are independent variables, so is $(X_i, \mu_i)$ and $(X_j, \mu_j)$
${\sigma_{x_i}}^2 = E[X - {\mu_X}^2] = \sum_{i=1}^{m} E[X_i - {\mu_i}^2] + 2E[X_i - \mu_i]E[X_j - \mu_j]$
$= \sum_{i=1}^{m} E[X_i - {\mu_i}^2]....E[X_i - \mu_i] = 0, E[X_j - \mu_j] = 0.$
$= \sum_{i=1}^{m} E[X_i - {\mu_i}^2] = \sum_{i=1}^{m} {\sigma_{x_i}}^2$
$\Rightarrow \sum_{i=1}^{m} {\sigma_{x_i}}^2 = Var[\sum_{i=1}^{m} X_i]$

Note:- it is possible in the case of $X_i$ for i= 1,2,3,....n, Whereas
$E[\sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} E[X_i]$ Always holds
$\Rightarrow E[aX + b] = a.E[X] + E[b]$
$\Rightarrow Var[aX] = a^2 Var[X]$

**Two point Sampling:-**

Variances of the sum of the independent random variable equals the sum of the variances (prooved above) ${\sigma_X}^2 = \sum_{i=1}^{m} {\sigma_{x_i}}^2$
[Given $X = \sum_{i=1}^{m} X_i$ and $X_i$ for $i = 1....n$ independent random variable]
Let X and Y be discrete random variable defined on the same probability space. The point closing function of X and Y is the function.
$P(x, y) = P_r[X = x \cap Y = y]$
$P_r[Y = y] = \sum_x P(x, y), P_r[X = x] = \sum_x P(x, y)$
$P_r[X = x \mid Y = y] = P_r(x, y)/P_r[Y = y] = P_r[X = x \cap Y = y]/P_r[Y = y]$
$= P_r[X = x \cap Y = y]/\sum_x P(x, y)$

These definitions extend to a set $X_1, X_2.....X_m$ of none than two random variables, such a set of random variables is said to be pairwise independent if $\forall i \neq j$ and $x, y \in IR$
$P_r[X_i = x \mid X_j = y] = P_r[X_i = x]$

**Two point sampling and probability amplification:-**

suppose n be a prime number (not very small) and $\mathbb{Z}_n$
denote the ring of integers modulo n. For a and b chosen independently and uniformly at random with a,b $\in \mathbb{Z}_n$. Let $Y_i = ai + bmodn$ for $i = o, 1, ...n - 1$.
Observation : For $i \not\equiv j$ (mod n) , $Y_i$ and $Y_j$ are uniformly distributed on $\mathbb{Z}_n$and pairwise independent.
Given fix values for $Y_i, Y_j$, we can solve $Y_i = ai + b(modn)$ and $Y_j = aj + b(modn)$ uniquely for a and b as

$$\begin{bmatrix} X_i \\ Y_i \end{bmatrix} = \begin{bmatrix} ai + b \\ aj + b \end{bmatrix} = \begin{bmatrix} i & 1 \\ j & 1 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix}$$

We now consider an application of these concepts to the reduction of the number of random bits used by RP Algorithms. Consider an RP algorithm A for deciding whether input strings x belong to a language L.
Given $x$, A picks a random number r from the range $\mathbb{Z}_n = 0, 1, ....n-1$
For a suitable choice of a prime number n and computes a binary value $A(x,r)$ with the following properties :-
$\rightarrow$ if $x \in L$, when $A(x,r) = 1$ for at least half possible numbers of r
$\rightarrow$ if $x \notin L$, then $A(x,r) = 0\ \forall x$.
Therefore, for a randomly chosen r, A(x,r) = 1 is conclusive proof that $x \in L$, when $A(x,r) = 0$ is evidence that $x \in L$
Two point sampling refers to two sample as above:-

$$A(x,r) = \begin{cases} 1 & \text{with prob} \geq 1/2 \text{ if } x \in L \\ 0 & \text{otherwise } x \notin L \end{cases}$$

$r \rightarrow$ vector of random bits, in the range $\mathbb{Z}_n = 0, 1, ....n-1$ for a suitably chosen n. if $A(X,r) = 1 \Rightarrow r$ is witness for $x \in L$, Otherwise no witness $|r|$ = Length of r = $\log n$
Repeat the computation of $A(x,r)$ for $t > 1$ independently chosen $r_i$ where, i= 1,2,...t. This is done to reduce the probability of finding $A(x,r) = 0$ for some randomly chosen r through $x \in L$. In this case if for any i we obtain $A(x,r_i) = 1$ we declare that x is in L, else we declare that x is not in L. By the independence of the trials , we are guaranteed that the probability of incorrectly classifying an input $x \in L$ (by disclosing that it is not in L) is at most $2^{-t}$.
$\Rightarrow$ errorr probability $\leq 2^{-t}$ , but this is at the cost of $\Omega(t \log n)$ random bits. Suppose instead that we are only willing to use $O(\log n)$ random bits. Take two independent sample from $\mathbb{Z}_n$ called a and b.
if we directly use a and b as witness, i.e. computing A(x,a) and A(x,b), yields an upper bound of 1/4 on error probability.
A better scheme is : let $r_i = ai + b \bmod$ n compute $A(x,r_i)$ for $1 \leq i \leq t$. If for any i $A(x,r_i) = 1$, we declare x is in L. else otherwise.
We can show that the error probability is much smaller than 1/4. Our analysis this way will be insensitive to the actual value of r in $\mathbb{Z}_n$ which are witness for x. We will only rely on the fact that at least half the value of r are witnesses. Clearly $A(x,r_i)$ is a random variable on the probability space of pairs a and b chosen independently from $\mathbb{Z}_n$.
we have seen earlier that $r_i$'s are pairwise independent and so are $A(x,r_i)$ for $1 \leq i \leq t$. let $Y = \sum_{i=1}^{t} A(x,r_i)$ Assuming $x \in L, E[Y] \geq \sum_{i=1}^{t} 1/2 \geq t/2$
$Var[Y] = {\sigma_Y}^2 \leq t/4 \Rightarrow \sigma_Y \leq \sqrt{t}/2$
The probability that the pairwise independent iterations produce an incorrect classification corresponds to the event Y = 0 and
$P_r[Y = 0] \leq P_r[|Y - E[Y]| \geq t/2]$
$\leq {\sigma_Y}^2/(t/2)^2$
$\leq (t/4)/(t^2/4)$
$\leq 1/t$
By Chebyshev inequality, the latter is at most 1/t. Therefore the error proba-

bility is at most 1/t, which is cosiderably important over the error bound of 1/4 achieved by the naive use of a and b. This improvement is sometimes referred as probability amplification.

## The Coupon Collectors Problem

There are n types of coupons, and at each trial one coupon is picked in random. How many trials one has to perform before picking all coupons? Let m be the number of trials performed. We would like to bound the probability that m exceeds a certain number, and we still did not pick all coupons.
Let $C_i \in$ 1, . . . ,n be the coupon picked in the i-th trial. The j-th trial is a success, if $C_j$ was not picked before in the first j- 1 trials. Let $X_i$ denote the number of trials from the i-th success, till after the (i + 1)-th success. Clearly, the number of trials performed is

$$X = \sum_{i=0}^{n-1} X_i$$

Clearly, the probability of $X_i$ to succeed in a trial is $p_i = n - i/n$ , and $X_i$ has geometric distribution with probability $p_i$ .
As such $E[X_i] = 1/p_i$, and $var[X_i] = q/p^2 = (1 - p_i)/{p_i}^2$. Thus,

$$E[X] = \sum_{i=0}^{n-1} E[X_i] = \sum_{i=0}^{n-1} n/(n-i) = nH_n = n(lnn + \Theta(1)) = nlnn + O(n).$$

Where, $H_n = \sum_{i=0}^{n} 1/i$ the n-th Harmonic number.
As for variance, using the independence of $X_0, X_1, ....., X_{n-1}$, we have
$V[X] = \sum_{i=0}^{n-1} V[X_i] = \sum_{i=0}^{n-1} 1 - p_i/{p_i}^2 = \sum_{i=0}^{n-1} \frac{1-(n-1)/n}{((n-i)/n)^2}$

$= \sum_{i=0}^{n-1} \frac{1/n}{((n-i)/n)^2} = \sum_{i=0}^{n-1} (i/n).(n/(n-i))^2$

$= n \sum_{i=0}^{n} i/(n-i)^2 = n \sum_{i=0}^{n} (n-i)/i^2 = n(\sum_{i=0}^{n} n/i^2 - \sum_{i=0}^{n} 1/i)$

$= n^2 . \sum_{i=0}^{n} 1/i^2 - nH_n.$

Since,$\lim_{n\to\infty} 1/i^2 = \pi^2/6$, we have $\lim_{n\to\infty} V[X]/n^2 = \pi^2/6$.
This implies a weak bound on the concentration of X, using Chebyshev inequality, but this is going to be quiet weaker than what we implied we can do, Indeed, we have

$$P_r[X \geq n \log n + n + t.n\pi/\sqrt{6}] \leq P_r[|X - E[X]| \geq t.V[X]] \leq 1/t^2$$

,
for any t.

# The Occupancy and Coupon Collector problems

## The Coupon Collectors Problem Revisited

There are n types of coupons, and at each trial one coupon is picked in random. How many trials one has to perform before picking all coupons? Let m be the number of trials performed. We would like to bound the probability that m exceeds a certain number, and we still did not pick all coupons.
In the previous lecture, we showed that

$$P_r[\#\text{of trials} \geq nlogn + n + t.n.\pi/\sqrt{6}] \leq 1/t^2$$

for any t. A stronger bound, follows from the following observation. Let ${Z_i}^r$ denote the event that the i-th coupon was not picked in the first r trials. Clearly, $P_r[{Z_i}^r] = (1-1/n)^r \leq e^{-r/n}$.
Thus, for r = $\beta nlogn$, we have $P_r[{Z_i}^r] \leq e^{-(\beta nlogn)/n} = n^{-\beta}$. Thus,
$P_r[X > \beta n \log n] \leq P_r[\bigcup_i Z_i^{\beta nlogn}] \leq n.P_r[Z_1] \leq n^{-\beta+1}$
This is quite strong, but still not as strong as we can do.
Let $c > 0$ be a constant, $m = nlnn + cn$ for a positive integer n. Then for any constant k, we have

$$\lim_{n\to\infty} \binom{n}{k}(1-k/n)^m = exp(-ck)/k!.$$

Proof: $(1-k^2m/n^2)exp(-km/n) \leq (1-k/n)^m \leq exp(-km/n)$.
Observe also that $\lim_{n\to\infty}(1-k^2m/n) = 1$, and $exp(-km/n) = n^{-k}exp(-ck)$.
Also,
$\lim_{n\to\infty} \binom{n}{k}k!/n^k = lim_{n\to\infty}(n.(n-1).....(n-k+1))/n^k = 1$.
Thus,
$\lim_{n\to\infty} \binom{n}{k}(1-k/n)^m = lim_{n\to\infty}(n^k/k!)exp(-km/n)$
$= lim_{n\to\infty}n^k/k!.n{-}k.exp(-ck) = exp(-ck)/k!$.
Let the random variable X denote the number of trials for collecting each of the n types of coupons. Then, for any constant $c \in IR$, and $m = nlnn + cn$, we have

$\lim_{n\to\infty} P_r[X > m] = 1 - exp(-e^{-c})$.

Before dwelling into the proof, observe that
$1 - exp(-e^{-c}) \approx 1 - exp(-e^{-c}) = e^{-c}$, as such the bound in the above theorem is indeed a considerable improvement over the previous bounds.
We have $P_r[X > m] = P_r[\bigcup_i {Z_i}^m]$. By inclusion-exclusion, we have
$P_r[\bigcup_i {Z_i}^m] = \sum_{i=1}^n (-1)^{i+1}{P_i}^n$, where

${P_j}^n = \sum_{1\leq i_1\leq i_2\leq ....\leq i_j\leq n} P_r[\bigcap_{v=1}^j {Z_{i_v}}^m]$
Let ${S_k}^n = \sum_{i=1}^k (-1)^{i+1}{P_i}^n$. We know that ${S_{2k}}^n \leq P_r[\bigcup_i {Z_i}^m] \leq {S_{2k+1}}^n$.
By symmetry, ${P_k}^n = \binom{n}{k}P_r[\bigcap_{v=1}^k {Z_v}^m] = \binom{n}{k}(1-k/n)^m$,

Thus, $P_k = \lim_{n\to\infty} {P_k}^n = exp(-ck)/k!$.
Let $S_k = \sum_{j=1}^{k}(-1)^{j+1}P_j = \sum_{j=1}^{k}(-1)^{j+1}.exp(-cj)/j!$.
Clearly, $\lim_{k\to\infty} S_k = 1 - exp(-e^{-c})$ by the Taylor expansion of exp(x) for $x = -e^{-c}$. Indeed,
$exp(x) = \sum_{j=0}^{\infty} x^j/j! = \sum_{j=0}^{\infty}(-e^{-cj}/j! = 1 + \sum_{j=0} \infty(-1)^j e^{-cj}/j!$

Clearly, $\lim_{k\to\infty} {S_k}^n = S_k$ and $\lim_{k\to\infty} S_k = 1 - exp(-e^{-c})$. Thus, (using fluffy math), we have
$\lim_{n\to\infty} P_r[X > m] = \lim_{n\to\infty} P_r[\bigcup_{i=1}^{n} {Z_i}^m]$
$= \lim_{n\to\infty} \lim_{k\to\infty} {S_k}^n = \lim_{n\to\infty} S_k = 1 - exp(-e^{-c})$.

**A Technical Lemma**

For any $y \geq 1$, and $|x| \leq 1$, we have

$$(1 - x^2y)e^{xy} \leq (1 + x)^y \leq e^{xy}$$

The right side of the inequality is standard by now. As for the left side, we prove it for $x \geq 0$Let us first prove that

$$(1 - x^2)e^x < 1 + x$$

Dividing by (1 + x), we get (1 - x) $e^x \leq 1$, Which obviously holds by the Taylor expansion of $e^x$, Indeed,
$(1 - x)e^x = e^x - xe^x = 1 + x/1! + x^2/2! + x^3/3!.... - x - x^2/1! - x^3/2!.... \leq 1$.
Next, observe that $(1 - x^2)^y \geq 1 - yx^2, for y \geq 1$. As such,
$(1 - x^2y)e^{xy} \leq (1 - x^2)^y e^{xy} = ((1 - x^2)e^x)^y \leq (1 + x)^y \leq e^{xy}$.
A similar argument works for $x \leq 0$.

## Cherno. Inequality

### Tail Inequalities

**The Cherno. Bound  General Case**
Here we present the Cherno. bound in a more general settings.
Let $X_1, ....X_n$ ba an n independent Bernouli trials, where

$$P_r[X_i = 1] = p_i, \text{and} P_r[X_i = 0] = q_i = 1 - p_i$$

( Each $X_i$ is known as a Poisson trials.) And let X = $\sum_{i=1} bX_i$ . $\mu = E[X] = \sum_i p_i$.
We are interested in the question of what is the probability that $X > (1 + \delta)\mu$?
Theorem : For any $\delta > 0$, we have $P_r[X > (1 + \delta)\mu] < (e^\delta/(1 + \delta)^{(1+\delta)})^\mu$
Or in a more simplified form, for any $\delta \leq 2e - 1$ ,

$$P_r[X > (1 + \delta)\mu] < exp(-\mu\delta^2/4,$$

, and

$$P_r[X > (1 + \delta)\mu] < 2^{-\mu(1+\delta)},$$

for $\delta \geq 2e - 1$. Proof: We have $[P_r[X > (1 + \delta)\mu] = P_r[e^{tX} > e^{t(1+\delta)\mu}]$. By the Markov inequality, we have:

$$P_r[X > (1 + \delta)\mu] < E[e^{tX}]/e^{t(1+\delta)\mu}$$

On the other hand,

$$E[e^{tX}] = E[e^{t(X_1+X_2+...+X_n)}] = E[e^{tX_1}]...E[e^{tX_n}]$$

Namely,

$$[P_r[X > (1 + \delta)\mu] < \frac{\prod_{i=1}^n E[e^{tX_i}]}{e^{t(1+\delta)\mu}} = \frac{\prod_{i=1}^n (1 + p_i(e^t - 1))}{e^{t(1+\delta)\mu}}$$

Let y = $p_i(e^t - 1)$ We know that $1 + y < e^y$ (since y ¿ 0). Thus,

$$P_r[X > (1 + \delta)\mu] < \frac{\prod_{i=1}^n exp(p_i(e^t - 1))}{e^{t(1+\delta)\mu}} = \frac{exp(\sum_{i=1}^n p_i(e^t - 1))}{e^{t(1+\delta)\mu}}$$

$$= \frac{exp((e^t - 1)(\sum_{i=1}^n p_i}{e^{t(1+\delta)\mu}} = \frac{exp((e^t - 1)\mu)}{e^{t(1+\delta)\mu}}$$

$$= \left(\frac{\exp(\delta)}{(1 + \delta)^{1+\delta}}\right)^\mu$$

if we set $t = \log(1+\delta)$.

**Definition:** $F^+(\mu,\delta) = \left[\frac{e^\delta}{(1-\delta)^{1+\delta}}\right]$

Arkansas Aardvarks win a game with probability 1/3. What is their probability to have a winning season with n games. By Cherno. inequality, this probability is smaller than

$$F^+(n/3,1/2) = \left[\frac{e^{1/2}}{1.5^{1.5}}\right]^3 = (0.89745)^{n/3} = 0.964577^n$$

For n = 40, this probability is smaller than 0.236307. For n = 100 this is less than 0.027145. For n = 1000, this is smaller than $2.17221.10^{-16}$ (which is pretty slim and shady). Namely, as the number of experiments is increases, the distribution converges to its expectation, and this converge is exponential.

**Theorem:** $P_r[X < (1-\delta)\mu] < e^{-\mu.\delta^2/2}$

**Definition** $F^-(\mu,\delta) = e^{-\mu.\delta^2/2}$

Let $\Delta^-(\mu,\varepsilon)$ denote the quantity, which is what should be the value of $\delta$, so that the probability is smaller than $\varepsilon$. We have that

$$\Delta^-(\mu,\varepsilon) = \sqrt{\frac{2\log 1/\varepsilon}{\mu}}$$

.

$$\Delta^+(\mu,\varepsilon) < \frac{\log(1/\varepsilon)}{\mu} - 1$$

**A more convenient form** : Proof

$$\left[\frac{e}{1+\delta}\right]^{(1+\delta)\mu} \leq \left[\frac{e}{1+2e-1}\right]^{(1+\delta)\mu} \leq 2^{-(1+\delta)\mu}$$

since $\delta > 2e-1$,

We proove this only for $\delta \leq 1/2$. For details about the case $1/2 \leq \delta \leq 2e-1$,

| Values | Probabilities | Inequality |
|---|---|---|
| $-1, +1$ | $\mathbf{Pr}[X_i = -1] =$ $\mathbf{Pr}[X_i = 1] = \frac{1}{2}$ | $\mathbf{Pr}[Y \geq \Delta] \leq e^{-\Delta^2/2n}$ $\mathbf{Pr}[Y \leq -\Delta] \leq e^{-\Delta^2/2n}$ $\mathbf{Pr}[\lvert Y\rvert \geq \Delta] \leq 2e^{-\Delta^2/2n}$ |
| $0, 1$ | $\mathbf{Pr}[X_i = 0] =$ $\mathbf{Pr}[X_i = 1] = \frac{1}{2}$ | $\mathbf{Pr}\big[\big\lvert Y - \frac{n}{2}\big\rvert \geq \Delta\big] \leq 2e^{-2\Delta^2/n}$ |
| 0,1 | $\mathbf{Pr}[X_i = 0] = 1 - p_i$ $\mathbf{Pr}[X_i = 1] = p_i$ | $\mathbf{Pr}[Y > (1+\delta)\mu] < \Big(\frac{e^\delta}{(1+\delta)^{1+\delta}}\Big)^\mu$ |
| | For $\delta \leq 2e - 1$ $\delta \geq 2e - 1$ For $\delta \geq 0$ | $\mathbf{Pr}[Y > (1+\delta)\mu] < \exp\big(-\mu\delta^2/4\big)$ $\mathbf{Pr}[Y > (1+\delta)\mu] < 2^{-\mu(1+\delta)}$ $\mathbf{Pr}[Y < (1-\delta)\mu] < \exp\big(-\mu\delta^2/2\big)$ |

Fig. 2: Summary of Chernoff type inequalities

$$P_r[X > (1+\delta)\mu] < \frac{e^\delta}{(1+\delta)^(1+\delta)}^\mu = exp(\mu\delta - \mu(1\delta)ln(1+\delta))$$

The Taylor expansion of $ln(1+\delta)$ is

$$\delta - \delta^2/2 + \delta^3/3 - \delta^4/4 \geq \delta - \delta^2/2$$

,
for $\delta \leq 1$, Thus,

$$P_r[X > (1+\delta)\mu] < exp(\mu(\delta - (1+\delta)(\delta - \delta^2/2))) = exp(\mu(\delta - \delta + \delta^2/2 - \delta^2 + \delta^3/2))$$

$$\leq exp(\mu(-\delta^2/2 + \delta^3/2)) \leq exp(-\mu\delta^2/4)$$

, for $\delta \leq 1/2$

**Application of the Cherno. Inequality Routing in a Parallel Computer**

Let G be a graph of a network, where every node is a processor. The processor communicate by sending packets on the edges. Let [1, . . . , N] denote be vertices (i.e., processors) of G, where $N = 2^n$, and G is the hypercube. As such, each processes is a binary string $b_1, b_2...b_n$ .
We want to investigate the best routing strategy for this topology of network. We assume that every processor need to send a message to a single other processor. This is representation by a permutation $\pi$, and we would like to figure out how to send the permutation and create minimum delay?

In our model, every edge has a FIFO queue of the packets it has to transmit. At every clock tick, one message get sent. All the processors start sending the packets in their permutation in the same time.

- Pick a random intermediate destination $\sigma(i)$ from [1, . . . , N]. Packet $v_i$ travels to $\sigma(i)$.
- Wait till all the packets arrive to their intermediate destination.
- Packet $v_i$ travels from $\sigma(i)$ to its destination d(i).

Theorem : For any deterministic oblivious permutation routing algorithm on a network of N nodes each of out-degree n, there is a permutation for which the routing of the permutation takes $\Omega(\sqrt{N/n})$ time.
Oblivious here refers to the fact that the routing of packet is determined only by inspecting only the packet, and without referring to other things in the network.
*How do we sent a packet?* We use bit fixing. Namely, the packet from the i node, always go to the current adjacent node that have the first di.erent bit as we scan the destination string d(i). For example, packet from (0000) going to (1101), would pass through (1000), (1100), (1101). We assume each edge have a FIFO queue. The routing algorithm is depicted above.
We analyze only (i) as (iii) follows from the same analysis. In the following, let $p_i$ denote the route taken by $v_i$ in (i).

Once a packet $v_j$ that travel along a path $p_j$ can not leave a path $p_i$ , and then join it again later. Namely, $p_i \cap p_j$ is (maybe an empty) path.
*Lemma:* Let the route of a message c follow the sequence of edges $.\pi = (e_1, e_2, ..., e_k)$. Let S be the set of packets whose routes pass through at least one of $(e_1, ..., e_k)$. Then, the delay incurred by c is at most $|S|$.

Proof : A packet in S is said to leave $\pi$ at that time step at which it traverses an edge of $\pi$ for the last time. If a packet if ready to follow edge $e_j$ at time t, we define its lag at time t to be t-j. The lag of c is initially zero, and the delay incurred by c is its lag when it traverse $e_k$ . We will show that each step at which the lag of c increases by one can be charged to a distinct member of S.
We argue that if the lag of c reaches l+ 1, some packet in S leaves $\pi$ with lag l When the lag c increases from l to l + 1, there must be at least one packet (from S) that wishes to traverse same edge as c at that time step, since otherwise c would be permitted to traverse this edge and its lag would not increase. Thus, S contains at least one packet whose lag reach the value l, Let $\tau$ be the last time step at which any packet in S has lag l. Thus there is a packet d ready follow edge $e_\mu$ at $\tau$ such that $\tau - \mu = l$. We argue that some packet of S leaves $\pi$ at $\tau$; this establishes the lemma since once a packet leaves $\pi$, it would never join it again and as such will never again delay c.
Since d is ready to follow $e_\mu$ at $\tau$, some packet $\omega$. (which may be d itself) in S follows $e_\mu$ at time $\tau$. Now $\omega$ leaves $\pi$ at time $\tau$; if not, some packet will follow $e_{\mu+1}$ at step $\mu + 1$ with lag still at l, violating the maximality of $\tau$ We charge to $\omega$ the increase in the lag of c from l to l + 1; since $\omega$ leaves $\pi$, it will never be

charged again. Thus, each member of S whose route intersects $\pi$ . is charge for at most one delay, establishing the lemma.
Let $H_{ij}$ be an indicator variable that is 1 if $p_i$ and $p_j$ share an edge, and 0 otherwise. The total delay for $v_i$ is at most $\leq H_{ij}$ . Note, that for a fixed i, the variables $H_{i1}, ..., H_{iN}$ are independent (note however, that $H_{11}, ..., H_{NN}$ are not independent!). For $P_i = (e_1, ..., e_k)$, let T(e) be the number of packets (i.e., paths) that pass through e.

$$\sum_{j=1}^{N} H_{ij} \leq \sum_{j=1}^{N} T(e_j)$$

and thus

$$E[\sum_{j=1}^{N} H_{ij}] \leq E[\sum_{j=1}^{k} T(e_j)]$$

Because of symmetry, the variables T (e) have the same distribution for all the edges of G. On the other hand, the expected length of a path is n/2, there are N packets, and there are Nn/2 edges. We conclude E[T (e)] = 1. Thus

$$\mu = E[\sum_{j=1}^{N} H_{ij}] \leq E[\sum_{j=1}^{k} T(e_j)] = E[|p_i|] \leq n/2$$

By the cheroff inequality, we have

$$P_r[\sum_{j} H_{ij} > 7n] \leq P_r[\sum_{j} H_{ij} > (1+13)\mu] < 2^{-13\mu} \leq 2^{-6n}$$

Since there are $n = 2^n$ packets, we know that with the probability $\leq 2^{-5n}$ all packets arrive to their temporary destination in a delay of most 7n.
*Theorem*: Each packet arrives to its destination $in \leq 14n$ stages, in probability at least 1 - 1/N (note that this is very conservative).

**Application of the Cherno. Inequality  Faraway Strings**

Consider the Hamming distance between binary strings. It is natural to ask how many strings length n can one have, such that any pair of them, is of Hamming distance at least t from each other. Consider two random strings, generated by picking at each bit randomly and independently. Thus, $E[d_H(x, y)] = n/2$, where $d_H(x, y)$ denote the hamming distance between x and y. In particular, using the Cherno. inequality, we have that

$$P_r[d_H(x, y) \leq n/2 - \Delta] \leq (-2\delta^2/n).$$

Next, consider generating M such string, where the value of M would be determined shortly Clearly, the probability that any pair of strings are at distance at

most $n/2 - \delta$, is

$$\alpha < M^2 exp(-2n^2/16n) = M^2 exp(-n/8).$$

Thus, for M = exp(n/16), we have that $\alpha < 1$. We conclude:
There exists a set of exp(n/16) binary strings of length n, such that any pair of them is at Hamming distance at least n/4 from each other.
This is our first introduction to the beautiful technique known as the probabilistic method we will hear more about it later in the course.
This result has also interesting interpretation in the Euclidean setting. Indeed, consider the sphere $\mathbb{S}$ of radius $\sqrt{n/2}$ centered at (1/2, 1/2, . . . , 1/2)$\in \mathbb{R}^2$. Clearly, all the vertices of the binary hypercube $\{0,1\}^n$ lie on this sphere. As such, let P be the set of points on $\mathbb{S}$ that exists according to Lemma, A pair p, q of points of P have Euclidean distance at least $\sqrt{d_H(p,q)} = \sqrt{n4} = \sqrt{n/2}$ from each other. We conclude:
Consider the unit hypersphere $\mathbb{S}$ in $\mathbb{R}^n$. The sphere $\mathbb{S}$ contains a set Q of points, such that each pair of points is at (Euclidean) distance at least one from each other, and $|Q| > exp(n/16)$.

## Chernoff Bounds III

$F^+(\mu,\delta) < \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}^\mu\right) < \left(\frac{e}{(1+\delta)}\right)^{(1+\delta)\mu}$

if $\delta < 2e-1$, then $F^+(\mu,\delta) < 2^{-(1+\delta)\mu}$

$\Delta^+(\mu,\delta) = (\log_2 1/\varepsilon)/\mu - 1$

**Theorem:-** For $0 < \delta \leq \mu$

$F^+[\mu,\delta) \leq e^{-C(u)\mu\delta^2} = \varepsilon$

C(u) = $\frac{(1+u)ln(i+u)-u}{u^2}$

Then : if $u = 2e-1$, this gives
$F^+(\mu,\delta) < e^{-\mu\delta^2/4}$

and thus $\Delta^+(\mu,\varepsilon) < \sqrt{(4ln(1/\varepsilon))/\mu}$

**Proof :-** Lemma 1 is a variant of this theorem with proof
$0 < \delta \leq 1, F^+(\mu,\delta) < e^{-\mu\delta^{2/3}}$

example : n Balls into n bins, X = # balls in last bin
$P_r(X_1 > m) \leq 1/n^2$
Chernoff bound : $m = 1 + \Delta^+(1, 1/n^2)$,

using the above estimate (for big $\delta$)
$\Rightarrow \Delta^+(1, 1/n^2) < 2\log_2 n - 1$
Better value would be $\delta = \frac{1.5lnn}{lnlnn}$ in the original $F^+$.
Hint :- simple formula for $\Delta^+, \Delta^-$ ok if $\mu = \Omega(\log n)$
example :- [ set boundry] $A \in 0, 1^{nxn}$
Probability :- Find $b \in -1, 1^n$ s.t $\mod Ab_\infty$ is minimised

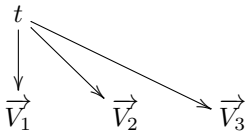
Detemine each $b \in -1, 1^n$ randomly and independently with $P_i(b, 1) = 1/2$
X = $a_j \Rightarrow E[x] = 0$
$P_r(|x - 0| \geq 4\sqrt{nlnn}) \leq (2/n^2)$

**Routing on a parellel computer** :- Network of N-processor connected by channels, change of each process is (# outgoing/ is ports) = d.

3- dimentional binning Hyper cubes.
n- dimentional N = $2^n$ (no. of modes) $\Rightarrow n = \log_2 N$
$\#edges = n.2^{n-1}$ undirected
$= 2.n2^{n-1}$ (directed) $= n.2^n$
nodes $V_1, V_2.....V_N$
$\pi \in S_n$ routing request : emergency processor is a source and distorton of exactly
$i \xrightarrow{Packet} \pi(i)$ [route only depends on origin and destination]
$i \xrightarrow{occupy} \pi(i)$
**Theorem:** For any deterministic routing algorithm of a network of N-mode of in/out d, these is one insource a permutation requiring $\Omega(\sqrt{N}/d)$ steps (Multi-port mode).

$\Omega(\sqrt{N}/d)$ in single port module

**Proof:**




t destination graph for $b_t$ :

$$S_k = \begin{cases} \text{set of edges} & \\ b \geq k & \text{route to k} \end{cases}$$

$V(S_k) = \{setof * * * incidenttoedgesins(k)$
$t \geq V(S_n)$ if $k \leq N/d, Assumek \leq N/d$

$|V - v(S_k)| < (K-1)(d-1)/(|V(S_k)|)$
Proof : $\varepsilon \in V - V(S_k)$
Path $\varepsilon, V_1, V_2.....V_e$ to t in b.

$V_i$ frost mode on this path in $V(S_k)$.

$$W\bullet \longrightarrow \bullet V_1 \longrightarrow \bullet V_2 \longrightarrow \bullet t \in V(S_k)$$

$edge(V_{i-1}, V_1)$ is at most (K-1) ****
$\Rightarrow$ at most (k-1)(d-1) starting points w hit $v_i$ as first node.

$$|V - V(S_k)| \leq (k-1)d|V(S_k)|$$

$$N = V(S_k) + |V - V(S_k)|$$

$$\leq |V(S_k)| + (k-1)(d-1)|V(S_k)|$$

$$\leq 2[1 + (k-1)(d-1)|S_k|]$$

$$S_k \geq \frac{N}{2[1 + (k-1)(d-1)]}$$

## Parellel routing (real width to be mentioned)

routing ********
in/out degree $\leq$ d.
$\exists$ at most (k-1)d nodes with charged.
$|S_k| \geq \frac{N}{2[1+(k-1)(d)]} \geq \frac{N}{2.k.(d)} \leq |S_k|$

Per destination graph, we have at least $\geq N/2kd$ k-congested edges together we have $\sum_{n=1}^{N} N/2kd = N^2/2kd$

E = Min no of edges $\leq Nd$

Average no of marks per edges $= N^2/2kd/Nd = N/2kd^2$ $\exists$ edge with $\geq N/2kd^2$ marks (k-congested)
Pick k such that $K = N/2kd^2, K = \sqrt{N/2d^2}$

********** $S_i \mapsto t_i complementedittopermutationoutingrequest$ *****

### Randomized approach for parellel routing :-

n-dimentional hypercube **** route depends only on diffrention $\Rightarrow$ graph is a trees

### Bit finding routing protocol:-

******* for each edge, no of nodes $= 2^{d-1}$

**Routing Algorithm**

Phase 1 : Pick for each i, a random intermediate destination $\sigma \in 0, 1, 2, ...N-1$ for N-Hypercube
*** $V_i$ goes from $\sigma_i$ to the original desination $d_i$. $i \leftrightarrow d_i$ one to one (injective) into (it is bijective) use bit forming protocolformate sure all routes and shortest routes
warning :- $i \Rightarrow \sigma_i$ need not be 1-to-1 (n- balls, m-bins)
for each outing edge we have a better, though we can send out 1 *** at a time (more *** conjunction)
Use FIFO, two **+ to maintain there mutula order.
Lemma : Let the routes of path $V_i$ be $(e_1, e_2, e_3.....e_n)$
S:= Set of paths other than $V_i$ that use at least of the $e_j, i = 1, 2...k$
The the delay inccoured by $V_i is \leq |S|$
Proof :- *****
V have $e_1$ in step t when step t is the least step in which V used one of the $e_j$.
V uses one of the $e_j$
if paths V want to take $e_j$ at time t, we say lag of V at time t is (t - j) lag.
If for our path, lag from l to l+1 increases from l to l+1. Let t' be the last time lags whole a.
Let t' be the last time lags where lag l path moves. such a *** must *** $e_i$, it is uniquely clustered.
eachpacket is charged one unit.

### 1.1 Single port module(to be added)

$$H_{ij} = \begin{cases} 1 & \text{if } e_i \text{ and } e_j \text{ share at least one edge} \\ 0 & \text{otherwise} \end{cases}$$

$\Rightarrow$ change of $V_i \leq \sum_{j \neq i} H_{ij}$

T(e) = RV that gives no of routes in passing sequences $\sigma$ using edge

$\Rightarrow \sum_{j \neq 1}^{e} H_{ij} \leq \sum_{j \neq 1}^{k} T(e_j)$
$\Rightarrow E[\sum_{j \neq 1} H_{ij}] \leq \sum_{j=1}^{k} E[T(e_j)] \leq n/2$
avg. no of edge $***$ path in Hypercube $= n/2$

**Chernoff bound:-**

$$P_r[\sum_i J_{ij} \geq \sigma_n] \leq 2^{-\sigma n}$$

$$P_r[\text{Any packet delay} \geq \sigma_n] \leq 2^{-5n}$$

$\Rightarrow$ Phase one endes in $\leq n + \sigma_n$ steps with prob $\geq 1 - (1/32)^n$
(trace **** ) This is for multiport model.

**A wiring problem**

Hypercube: having more than two matrices wins can only run vertically or horizontally.
Problem: Find a layout of the Nets **** which minimizes the width of man bounding.
For net i, variable $X_{i0}$ and $X_{i1}$
IMAGE***
b bounding : $T_{b0} =$ { i; i passes through b if $X_{i0} = 1$
$T_{b1}$ { i ; 1 passes through b if $X_{i1} = 1$
$\Rightarrow$ Integer program $\Rightarrow$ integer linear problem, minimize w , the width of the boudries; where

$$X_{i0}, X_{i1} \in 0, 1$$

$$X_{i0} + X_{i1} = 1$$

$$\sum_{i \in T_{b0}} X_{i0} + \sum_{i \in T_{b1}} X_{i1} \leq w$$

**Relaxation:** $0 \leq X_{i0}, X_{i1} < 1$ (Randomized rounding)
$\hat{X}_{i0}, \hat{X}_{i1}$ are functional solutions.
$\hat{W}$ solution for Lp.
$W_{opt}$ solution for$I_{LP}$
**Wayout: randomized rounding:**
Set $\hat{X}_{i0} = 1$ with prob $\hat{X}_{i0}$
Theorem : Let $0 < \varepsilon < 1$, this with probability $\geq (1 - \varepsilon)$,
the global wiring 'S' produced by a randomized rouding of the relaxed solution algorithm, statistics

$$W_s \leq \hat{W}.(1 + \Delta^{+}(\hat{W}, \varepsilon/2n))$$

$$\hat{W} \leq W$$

$$\hat{W} =$$

optimal fractional solution(reduced)

## A wiring problem :-

Proof : Consider boundry b
$\sum_{i\in T_{i0}} \hat{X}_{i0} + \sum_{i\in T_{i0}} X_{i1} \leq \hat{W}$
$W_s(b) = \#$ wires crossing through boundry b

$$\sum_{i\in T_{i0}} \bar{X}_{i0} + \sum_{i\in T_{i1}} \bar{X_{i1}}$$

$$E[W_s(b)] = E[\sum_{i\in T_{i0}} \bar{X}_{i0}] + E[\sum_{i\in T_{i1}} \bar{X}_{i1}]$$

$$E[W_s(b)] = \sum_{i\in T_{i0}} E[\bar{X}_{i0}] + \sum_{i\in T_{i1}} E[\bar{X}_{i1}]$$

$$= \sum_{i\in T_{i0}} E[\hat{X}_{i0}] + \sum_{i\in T_{i1}} E[\hat{X}_{i1}] \leq \hat{W}$$

.

$$Prob[W_s(b) > \hat{w}(1 + \Delta^+(\hat{W}, \varepsilon/2n))] \leq \varepsilon/2n$$

$$Prob[\text{any boundry} > .....] \leq 2n.\varepsilon/2n \leq \varepsilon$$

If $\hat{W}$ is small then estimate is not very good.
if $\hat{W}$ is big ( at least $\log n$ ) then estimate is very good.

**Martin Gales**

$\sum_{i=0}^{n} X_i$
In book, general treatment using sequences of probability spaces.
adding additional probability (a called filters)
$X_i$ lines in a probability space which is a ***** of probability space of $X_{i-1}$.

$$P_r[X = x \mid A]$$

$$P_r[X = x \mid Y = y] = \frac{P_r[X = x \bigcap Y = y]}{P_r[Y = y]}$$

$$= P(x, y) / \sum_x P(x, y)$$

$$E[X \mid Y = y] = \frac{\sum_x X.P(x, y)}{\sum_x P(x, y)}$$

**lemma** :$E[E[X \mid Y]] = E[X]$
**lemma** :$E[Y.E[X \mid Y]] = E[X.Y]$

**Definiton:-** A sequence $X_0, X_1, ....$ of R.V is called a Marting gale sequence if $V_i > 0$ $E[X_i \mid X_0, X_1, ...X_{i-1}] = X_{i+1}$
e.g. $a_i$ bet in ith step $X_0$ starting capital then $X_i$ is capital after bet i.

**Lemma 1.** :- Let $X_0, X_1, X_2.....$ be a martin gale, Then $V_i \geq 0, E[X_i] = E[X_0]$
$[E[X_i] = X_i + ...]$
**definition:** Let $X_0, X_1 ... be Martingale$
$Y_i = X_i - X_{i-1} (Then X_i = X_0 + \sum_{j=1}^{i} Y_j)$
A sequence $Y_1, Y_2...$ of Rvs is called a Martin gale difference sequence if, $\forall i \geq 1$
$E[Y_i, \mid Y_1....Y_{i-1}] = 0$
Super and sub Martingale sub-optimal for ******

## Martin Que

**Martin Gales :-** RV's $X_0, X_1, .....X_{i-1}, X_i$
$E[X_i \mid X_0, .....X_{i-1} = X_{i-1}]$
$= .....$
$= X_0$

Theorem: ( Kolomogorov - dooll's inequality) :- let $X_0, X_1.....$
be a martingal $\lambda > 0$
Thus $P_r[min_{0 \leq i \leq n} X_i \geq \lambda] \leq E[(X_n)]/\lambda$
Proove meds to be added.
Theorem ( Azuma's Inequality ):- let $X_0, X_1...$ be a Martingals such that
$\forall x |X_k - X_{k-1}| \leq C_k (C_k maydependonk)$

Then $\forall t > 0 and \forall \lambda > 0 : - \ P_r[|X_t - X_0 \geq \lambda|] \leq 2 - e^{-\frac{\lambda^2}{2 \sum_{k=1}^{t} C^2 k}}$

analogus to chebyshev inequality

Corollary : C does not depend on k i.e. $|X_k - X_{k-1}| \leq C.$
$P_r[|X_t - X_0| \geq \lambda C \sqrt{t}] \leq 2.e^{-\lambda^2/2}$
This is also called as the methods of small differences.

Definition : $f : D_1 x D_2 x .....D_n \rightarrow IRn - aryfunction f[n - aryfunction]$

satisfies the Lips duitz condition if for all
$(X_1, X_2, .....X_n) \in \pi D_i, any i \in [n] and any$

$y_i \in D_i | f(X_1, X_2....X_{i-1}, y_i, X_{i+1}.....X_n) - f(X_1, X_2.....X_{i-1}, X_i, Xi + 1.....X_n)| \leq$
1

RVs $X_1, X_2.....X_n, f$ Lips duitz . Define
$Y_0 = E[f(X_1, X_2....X_n)]$

for $i = 1, 2, .....n$
$Y_i = E[f(x_1..X_n) \mid X_1....X_i]$
$\Rightarrow |Y_i - Y_{i-1}| \leq 1$, so can apply azuma's inequlaity
Application : Occupance problem - balls into bins.
(1) Throw m balls into n bins :- Z = number of empty bins in the end.
$X_i$ = bin for balls i Z = $f(X_1, X_2.....X_n)$, f is Lipz duitz
$\Rightarrow P_r[|Z - E(2)| \leq \lambda] \leq 2.e^{-\lambda^2/m}$ above corollary.
(2) Theorem : Let r = m/n , Z as above then,
$\mu = E[Z] = n(1 - 1/n)^m \approx n.e^- m/n$
$\approx n.e^{-r}$
and for $\lambda > 0, P_r[|Z - N| \geq \lambda] \leq 2.e^{-\frac{\lambda^2(n-1/2)}{n^2-\mu^2}}$
r = 1 ,$e^{-r} = 36.79$r = 2 ,$e^{-r} = 13.5$ r = 3 ,$e^{-r} = 4.98$ Proof :- (2nd part ) Z as above $Z_t = E[Z \mid t - 1ballshasbeenthrown]$
The $Z_t's$ use matin gale.

Z( Y ,t ) =$E[Z \mid Ybinsareemptyattimet]$
$= Y(1 - 1/n)^{m-t}$
1/t = # empty bins at time $t_0$.
$Z_{t-1} = Z(Y_{t-1}, t - 1) = Y_{t-1}(1 - 1/n)^{m-t+1}$

In step t, two possibilities
(1) $t^{th}$ ball enters currently all the non empty bin, this with probability = $1 - Y_{t-1}/n$
Then $Y_E = Y_{t-1}$ and
$Z_t = Z(Y_t, t) = Z(Y_{t-1}, t) = Y_{t-1}(1 - 1/n)$
(2) $t^{th}$ ball $\rightarrow$ currently empty bin, Prob for this = $Y_{t-1}/n$ Then $Y_t = Y_{t-1}$ and
$Z_t = Z(Y_t, t) = Z(Y_{t-1}, t) = (Y_{t-1} - 1)^{t-1}(1 - 1/n)^{m-t}$
RV $\delta = Z_t - Z_{t-1}$
To show that $\delta_t$ s are small - 2 cases
(1) with prob : $1 - Y_{t-1}/n, \delta_t$ is
$\delta_0 = Y_{t-1}(1 - 1/n)1m - t - Y_{t-1}(1 - Y_n)^{m-t+1}$
$(2)withprobY_{t-1}/n, \delta_t is$
$\delta_1 = (Y_{t-1} - 1)(1 - Y_n)^{m-t} - Y_{t-1}(1 - Y_n)^{m-t+1}$
$= -(1 - Y_{t-1}/n)(1 - Y_n)^{m-t}$
$-(1 - 1/n)^{m-t} \leq \delta_t \leq (1 - Y_n)^{m-t}$
Thus for t = 1,2,.... m
$C_t = (1 - 1/n)^{m-t}$
$\Rightarrow |Z_t - Z_{t-1}| \leq C_t$
$\sum_{t=1}^{m} C^2{}_t = 1 - (1 - Y_n)^{2m}/1 - (1 - Y_n)^2 = \frac{n^2-\mu^2}{2n-1}$
$P_r[|2 - \mu| \geq \lambda \leq 2.e^{-\lambda^2/2(n^2-\mu^2)/(2n-1)}]$
$\leq 2.exp(-\lambda^2 n - 1/2/(n^2 - \mu^2))$
For r large, this tail bound converges to $2.exp(-\lambda^2/n(1 - e1{-}2r))$

estimate from normal distribution

$$2.exp(-\frac{\lambda^2 e^r}{2n(1-e^{-r})}) = 2.exp(-\frac{\lambda^2 e^2}{n.2(1-e^{-r})})$$

r = 1 1.1565 2.1501
r = 2 1.0186 4.2728
r = 3 1.0025 38.1711

**Probability Method**

Existence proof, some has efficient construction
(a ) set balancing :- $V_i \in 0, 1^n \rightarrow$ where $V_1$ are the charachteristic vector
$W \in -1, 1^n$
$\forall : \|WV_i\|_\infty \leq 4\sqrt{nlm}$
Better argument : $t\sqrt{n}$ (b) Gave two Evaluation:-
# have $= 2^{2k} = 4^k$.
certificate of winning of the size $\leq 2^k$.
Probablistically $n^{2.0793}$ (roughly)
(c) Max - cat : G ( V , E) unidirectional simple graph
$V = V_1 \oplus V_2 | C_{V1V2}| \geq m/2$
$\frac{Min-cut}{minstcut}$ = Man. Flow

**Stoer Wagner Algorithm**

Randomized Flag :
$\forall v \in V$ : Put v into $V_1, V_2$ with prob 1/2.
$ev, w$ Prob (e is in cut) $= Y_2$
$E[|C_{V1V2}|] = m.1/2 = m/2$

**K-MAX SAT**

x,y,z ... boolean variable
$\bar{x}, \bar{y}, \bar{z}...Negationofvariable$
CNF $= F(X_1, X_2, ....X_n) = C_1 \wedge C_2, C_3 \wedge C_5$
$C_i = (\bar{X_{i1}} \vee X_{i2} \vee ..... \vee x^2{}_t)$
$F(X) = X \wedge \hat{X}$

**K-MAT-SAT (K literals)**

$\forall i$ set $X_i$ to v with prob 1/2
Prob(clause $C_i$) is unsatisfied.
$E$[#satisfied clauses] $\geq \sum_{i=1}^{t} (1-2)^{-k_i}$

$\geq t/2, t = \#$ classes
How to do deterministically??

**Approximation Algorithm**

A(I) = Approximate on Instance I
opt( I) =
$0 \leq A(I)/opt(I)_2 \leq 1$
Pitch the lowest bound for all
Minimisation Problem $\rightarrow$ denominator at least as big as Numerator
Minimisation Problem$\rightarrow$ 1 over Numerator

**Other Solution**

$Z_j \in 0, 1$ for classes $c_j$
$y_i \in 0, 1$ for variable $X_i$
Maximise $\sum_{j-1}^{t} Z_j, \forall i, j, Z_j, y_i \in 0, 1$
$\forall j, \sum_{\bar{x_l} \in c_j} Y_l + \sum_{\bar{x_l} \in c_j} (1 - y_l) \geq Z_j$
ILP $\Rightarrow \hat{X}_i, \hat{X}_j \in [0, 1]$.
$\sum_{j=1}^{t} \hat{Z}_j$ opt integral solution
Show : With randomised rounding $\geq (1 - 1/e) \sum_{j-1}^{t} \hat{Z}_j$ clauses are satisfied in
Proof : Let $B_k = 1 - (1 - 1/k)^k$
Claim: Prob that clause $C_j$ with k literals is satisfied by randomised **** is $\geq \beta_k \hat{Z}_j$
Since $\beta \geq (1 - 1/e)$ this proves the claim.
w.log $C_j = X_i \vee X_2 .... \vee X_k$
It is unsatisfied with prob $\prod_{i=1}(1 - y_i)$
Its satisfied with probing $\geq 1 - \prod_{i=k}^{k}(1 - y_i)$
This is smallest if $y_i$'s are as small as possible .
$\sum_{i=1}^{k} = \hat{Z}_j$ and all are equal.
$i.e \hat{Y}_i = \hat{Z}_j/k$
Thus it satisfies to show $1 - \prod_{i=1}^{k}(1 - 2/k) \geq \beta_k Z$.
$Z \in [0, 1]$
This function is concave .