CS7960 L1 : Review of Sequential Model

Turing Machines (Alan Turing 1936)
- single tape: moveL moveR, read, write
  each constant time
  constant pointer memory
  tape infinite (extra memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)
- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

Scanning (max)
- TM : O(n)
- VNA: O(n)

Sorting
- TM : O(n^2)
- VNA: O(n log n)

Searching
- TM : O(n)
- VNA: O(log n)

how big is log n, n, n log n, n^2 :

10^x	1	2	3	4	5	6	7	8	9
searc	0.000001	0.000001	0.000001	0.000002	0.000001	0.000002	0.000002	0.000007	0.001871
MAX	0.000003	0.000005	0.000006	0.000048	0.000387	0.003988	0.040698	9.193987	>15 min
QuiS	0.000005	0.000030	0.000200	0.002698	0.029566	0.484016	7.833908	137.9388	
BubS	0.000003	0.000105	0.007848	0.812912	83.12960	~2 hour	~9 days	???	

Gradations:

LOG	poly log (n)	:	log^c (n)
P	poly (n)	:	n^c
-- NP --			
EXP	exp (n)	:	c^n

Theory:

 - LOG not studied much since count loading of data
 - P is poly (n). Lots of neat algorithms.
     Sometimes constant c (in n^c) important, sometimes not.
 - EXP usually hopeless, but 1.000001^n is ok.
 - NP : verify solution in P, find solution conjectured EXP.
     if EXP number of (parallel) machines -> in P. (bits of solution argument)

Probability:

Let A, B be random variables.

Pr[A] * Pr[B] = Pr[A and B] off A and B are independent.

Pr[A and B] < Pr[A] + Pr[B]   "Union Bound"

Expected value A = E[A] = sum_{a \in U} a * Pr[a = A]

E[A] + E[B] = E[A + B]   "Linearity of Expectation"

Hash Functions:

h : U -> [n]

U := set of possible inputs, maybe [m], maybe [a-z,A-Z]^28
[n] := output universe

H = family of hash functions.

If H *universal* for x != y then Pr_{h \in H}[h(x) = h(y)] <= 1/n

Simple example
  h_{a,b}(x) = ((a x + b) mod p) mod n
where a in [1,p] and b in [0,p], both at random, and p > m and prime.

Mulitply-Shift hashing (Dietzfelbinger 97)
  high-order-bits(h_a(x) = (a x mod 2^w), N)   // top M bits of first arg
where a < 2^w (odd, at random), w := number of bits in machine word, n = 2^N