**Algorithm Analysis**

GP to infinity:

AP:

nCr =

Sum of squares

Harmonic Series

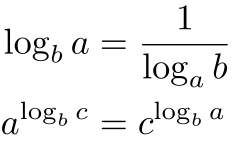
n! = O(nn)

< < n0.25 < < < < < < < < < <

Poly dominates logs:

Exp dominates polys:

,



**Big O: f(n) = O(g(n))** if there exist constants c > 0, n0 > 0 such that 0 <= f(n) <= cg(n) for all n >= n0.

* Define c and n0 such that for all n >= n0, eqn holds

**Big Omega: f(n) = 𝛀(g(n))** if there exists constants c > 0, n0 > 0 such that 0<=cg(n)<=f(n) for all n >= n0.

**Big Theta: f(n) = Θ(g(n))** if there exists constants c1, c2 and n0 such that 0 <= c1g(n) <= f(n) <= c2g(n) for all n >= n0

**Little o: f(n) = o(g(n)))** if there exists a constant n0> 0 such that for *any constant c > 0*, 0 <= f(n) < cg(n) for all n >= n0

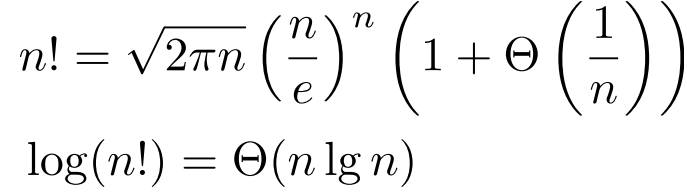
* Define n0 such that for any c > 0 and for all n >= n0, eqn holds

**Little-Omega: f(n) = 𝛚(g(n))** if there exists a constant n0 such that for any constant c > 0, 0<=cg(n)<f(n) for all n >= n0

Assume f(n), g(n) > 0

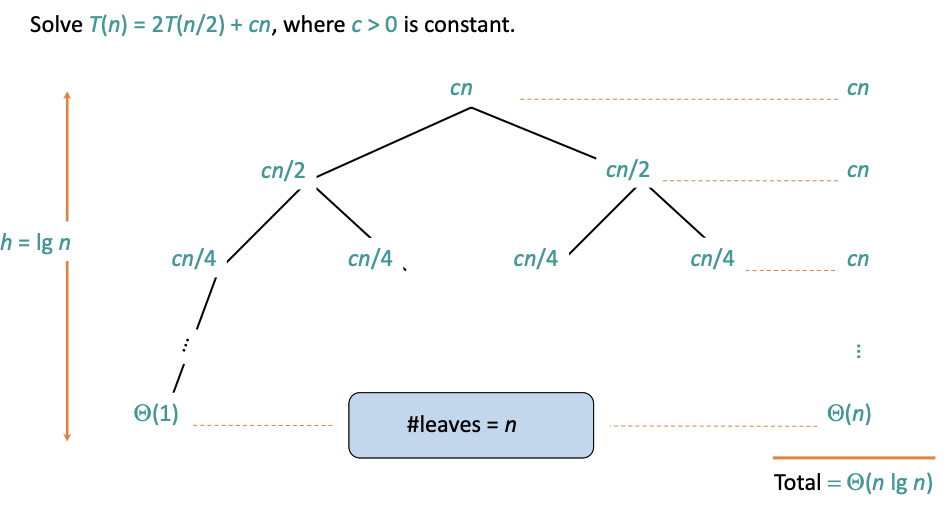
Properties:

* Transitive, reflexive (O, Θ, Ω of itself)
* Symmetry:
* Complementarity:

Stirling’s Approx.: 

**Iteration, Recursion, and Divide-and-Conquer**

**Iterative Algo Proof of correctness:** Loop invariant: Initialization, Maintenance, Termination;

**Recursion Tree Method:**

T(n) = 2T() + 1 = O(log n)

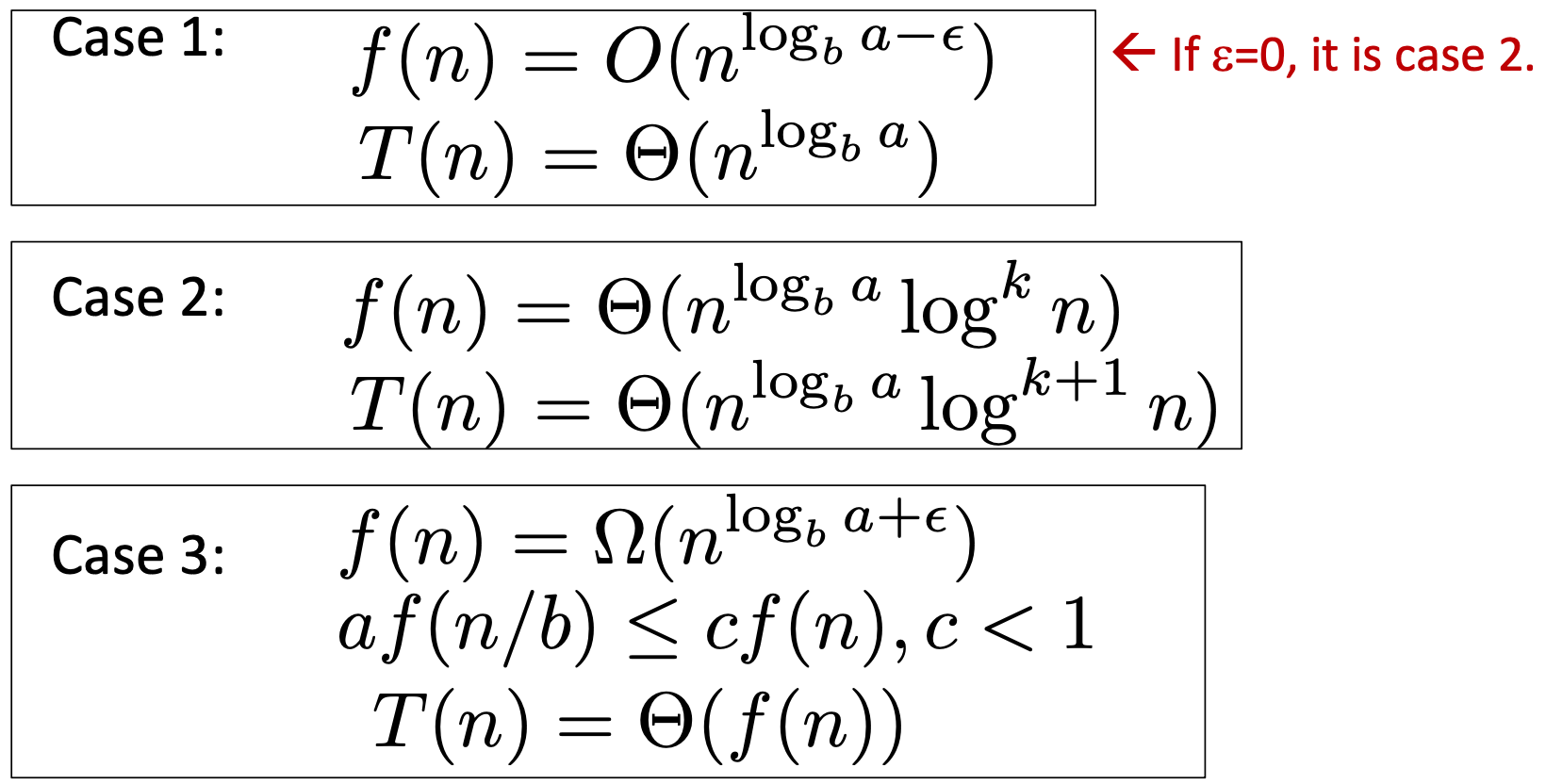
T(n) = 2T() + log n = O(log n log log n)

T(n) = 2T() + n = O(n) + O( log log n) = O(n)

T(n) = T() + 1 = O(log log n) ⇒ log log n levels

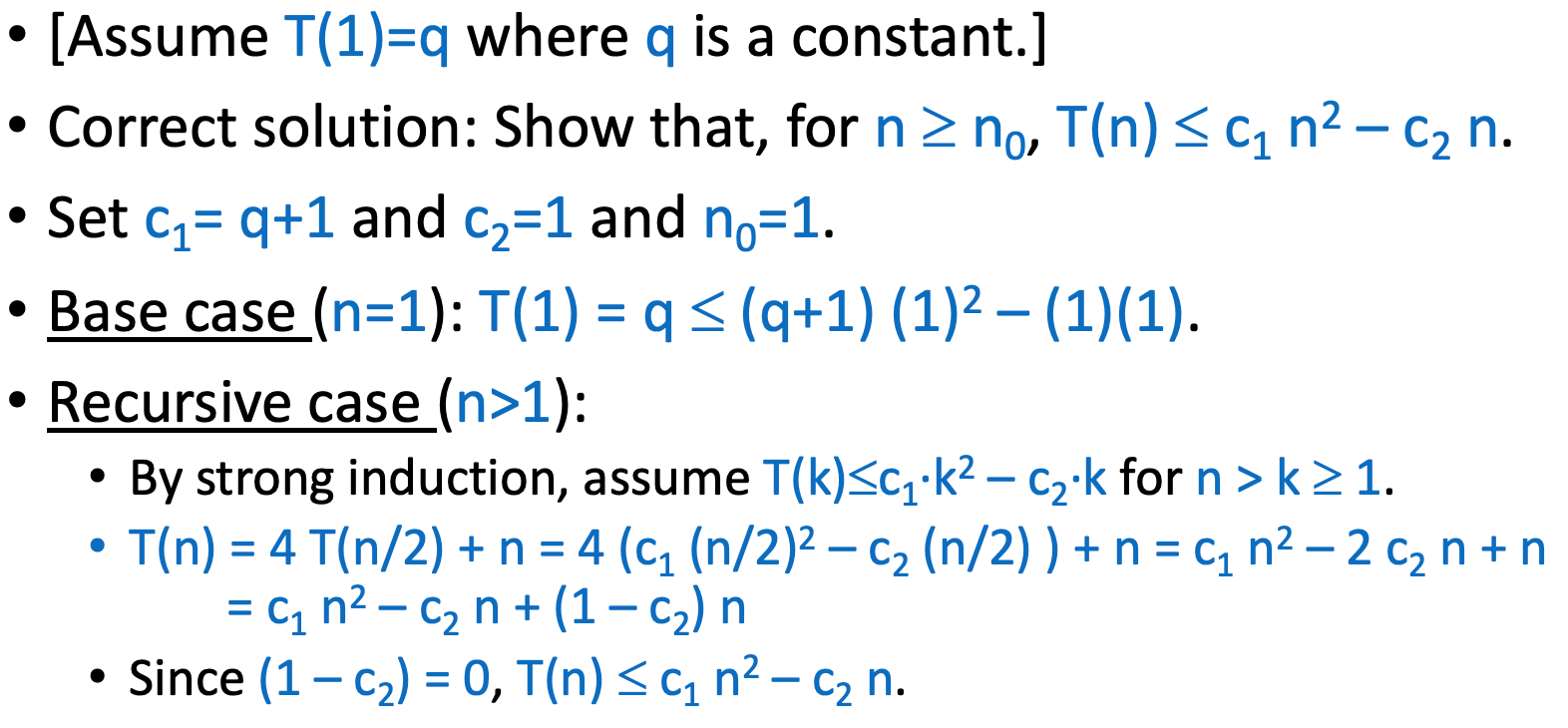
**Master method:** T(n) = a T(n/b) + f(n) where a >= 1, b > 1 and f is asymptotically positive.

Compare f(n) with n:



**Substitution method:**

E.g. T(n) = 4T(n/2) + n

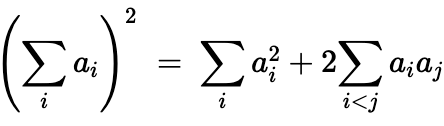


**Randomization**

,

Bernoulli trials before the first success: expected no. = 1/p

Pr[X = k] =



Average case analysis: Expected running t1ime of algo when the input is chosen uniformly at random from the set of all n! permutations

Algorithms like Quicksort are **Distribution Sensitive**: runtime depends on the initial permutation (input)

| Las Vegas | Output is always correct; Runtime is a r.v. |
| --- | --- |
| Monte Carlo | Output may be incorrect with some small probability; Runtime is deterministic |

**Hashing**

Desired properties: (N is number of stored items)

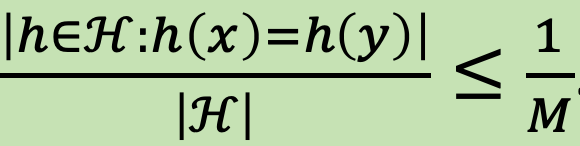
* Minimise collisions: Worst case = θ(N)
* Intuition: h(x) should be a “random” value
* Minimise storage: M = O(N)
* h(x) should be easy to compute

Randomization:

* Select a random hash function from hash family
* Use the same hash function for all elements

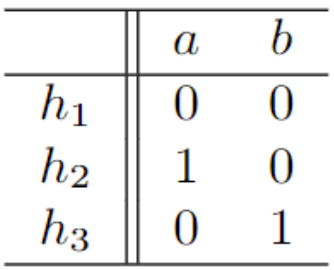
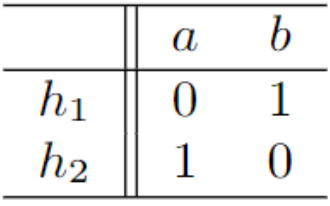
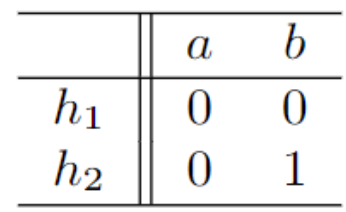
Universal Hashing:

Suppose H is a set of hash functions mapping 𝑈 to [𝑀]. We say H is universal if for all 𝑥 ≠ 𝑦: (Fraction of hash functions for which x and y collide <= 1/M)

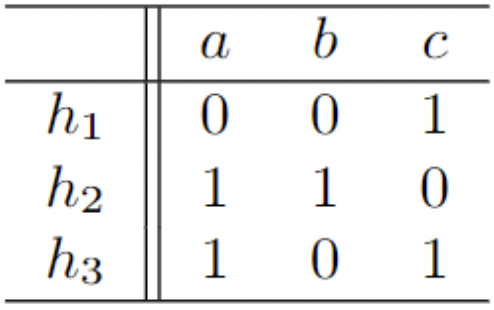
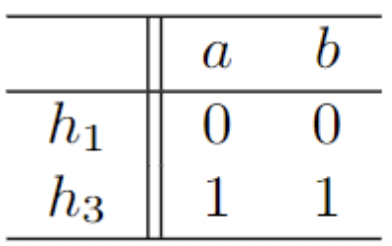


* For any 𝑥 ≠ 𝑦, if **h is chosen uniformly at random** from a universal H, there’s at most 1/M probability that h(𝑥) = h(𝑦).
* If 𝑥, 𝑦 are chosen uniformly from the universe 𝑈, the probability that h(𝑥)=h(𝑦) <= 1/𝑀 ⇒ FALSE: h can be all zero function

Universal hashing e.g.

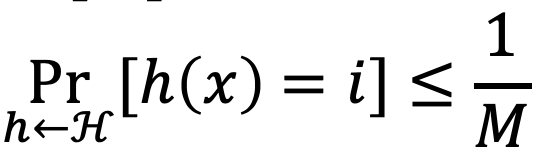


Not Universal



Uniform family of hash functions:

H is said to be a uniform family of hash functions if for every key 𝑥 ∈ 𝑈 and every hash value 𝑖 ∈ [𝑀], it holds that:



* for all 𝑀, there exists a family H that is uniform but not universal.
  + E.g. For every 𝑖 = 1, ... , 𝑀, let h1 be the constant function mapping all of 𝑈 to 𝑖.
  + {h1, ... , hM} is uniform but not universal.

Collision Analysis

Suppose H is a universal family of hash functions mapping 𝑈 to [𝑀]. For any 𝑁 elements 𝑥1,..., 𝑥N , the expected number of collisions between 𝑥1 and the other elements is < .

Expected Cost

Suppose H is a universal family of hash functions mapping 𝑈 to [𝑀]. For any sequence of 𝑁 insertions, deletions and queries, if 𝑀 ≥ 𝑁, then the expected total cost for a random h ∈ H is 𝑂(𝑁).  
Construction of universal family

Suppose 𝑈 is indexed by 𝑢-bit strings, and 𝑀 = 2m. For any binary matrix 𝐴 with 𝑚 rows and 𝑢 columns: hA(𝑥) =𝐴𝑥(mod2)

* {h(x): 𝐴 ∈ {0,1}m✕u} is universal.

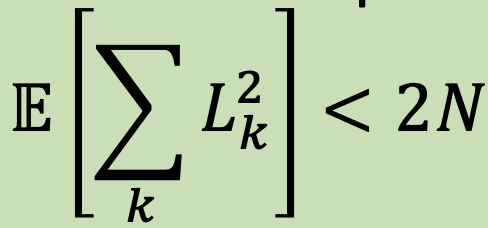
Perfect Hashing: Quadratic Space

If H is universal and 𝑀 = 𝑁2, then if h is sampled uniformly from H, the expected number of collisions is < 1

Perfect Hashing: 2-Level Scheme

* h: 𝑈 → [𝑁] from a universal hash family.
* Let 𝐿k be the number of 𝑥i’s for which h(𝑥i) = 𝑘
* Choose h1,...,hN second-level hash functions hk: [𝑁] → [𝐿k2] such that there are no collisions among the 𝐿k elements mapped to 𝑘 by h.

If H is universal, then if h is sampled uniformly from H:



Pairwise Independent

if, for any 2 distinct universe elements x,y, and for any 2 hash values i1, i2: (probability of a random hash function in H)

Pr[h(x) = i1,h(y)=i2] = 1/M2

**Amortised Analysis**

Analysing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive.

Aggregate Method:

Counting each operation's complexity, then find a pattern and come up with an upper bound.

e.g. k-bit Binary counter: No. of bit flips

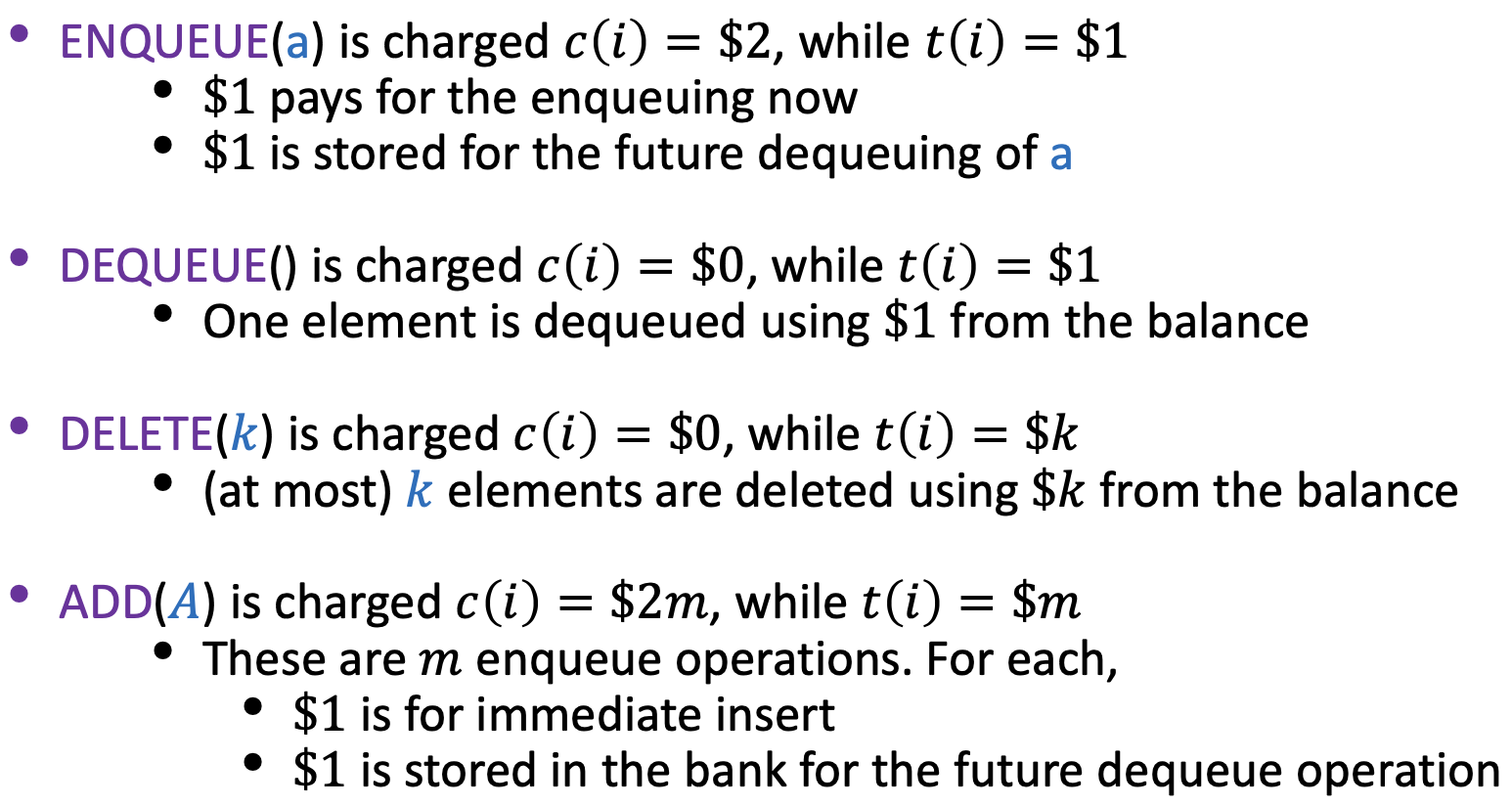
Let f(i) = no. of times ith bit flips

f(0) = n; f(1) = n/2; f(2) = n/4; f(i)=n/2i

Accounting Method:

1. Pay more (“amortised costs”) for inexpensive operations
2. Use balance to pay for the more exp operations later
3. Show that balance never < 0

E.g. n1 ENQUEUE, n2 DEQUEUE, n3 DELETE(k), n4 ADD(A) where |A|=m



Potential Method:

𝜙(𝑖): Potential at the end of the 𝑖-th operation

Conditions to be fulfilled by 𝜙:

* 𝜙(0)=0
* 𝜙(𝑖) ≥ 0, for all 𝑖

c(i) ≝ Actual cost of 𝑖-th operation + (𝜙(𝑖) − 𝜙(𝑖 − 1))

c(i) = Actual cost of 𝑛 operations + (𝜙(𝑛) − 𝜙(0))

* observe the costly operation and see if there is some quantity that is “decreasing” during the operation

E.g. cost of deleting all of the elements in dynamic tables

* Insert: 𝜙(𝑖) = 2i - size(𝑇)

→ table size halved when num elements after del = ½ original

𝜙(𝑖) = size(𝑇) − 𝑛

