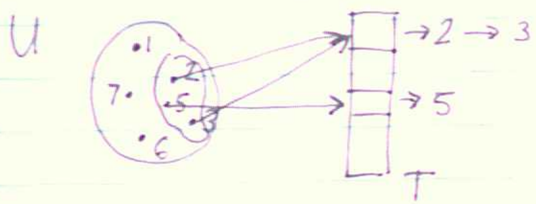
Math 340 - Feb. 24

Randomness & Algorithms

1. Hashing: Balls & Bins

Hash Table is a data structure, generalizes arrays.
Each piece of data that we want to store is assigned a key (ie. an id #).
Let U be the universe of keys $\{1, 2, \ldots, |U|\}$.
A hash function maps every key to a slot in the hastable $h: U \to T$
Typically $|T| < c|U|$ and $|T|$ is comparable to the number of keys you want to store.



Typically if many pieces of data are stored in the same slot they are accessed as a chain.

We'd like to design a hash function h such that the length of the longest chain is minimized.

Let h be a random function.
We'll analyze the behaviour in this case.
(In reality we need to be random-like but easy to recompute. eg. if x is a key, p is a prime such that $p = |T|$ then $(ax+b) \bmod p$ where $1 \le a, b \le p$ random)

n pieces of data are stored in n slots.
Slots are chosen independently, uniformly at random.
Equivalently, n bins and n balls, put in bins one by one at random.

We want to estimate $\mathbb{E}[M^*]$, where M^* - maximum # of balls in all the bins.

Theorem:
$\mathbb{E}[M^*] \le 2\log n + 1$ for n large enough $\quad (\log n = \ln n)$

Proof:
Let $X_1, X_2, \ldots, X_n$ be the random variables counting # of balls in each bin

$$p(X_i = r) = \binom{n}{r}\left(\frac{1}{n}\right)^r\left(1-\frac{1}{n}\right)^{n-r} \le \frac{n^r}{r!}\left(\frac{1}{n}\right)^r = \frac{1}{r!}$$

Binomial Distribution $p = \frac{1}{n}$

Note: $\binom{n}{r} = \frac{n(n-1)\ldots(n-r+1)}{r!} \le \frac{n^r}{r!}$, $\quad k! \ge \left(\frac{k}{2}\right)^{\frac{k}{2}}$

Let $r^* = 2\log n$ and $\log n \ge e^3$

Step 1: $p(X_i = r^*) = \frac{1}{(2\log n)!} = \frac{1}{\log n^{\log n}} \le \frac{1}{e^{3\log n}} = \frac{1}{n^3}$

Also $p(X_i = k) \le \frac{1}{n^3} \quad \forall k \ge r^*$

So, $p(X_i \ge r^*) \le n \cdot \frac{1}{n^3} = \frac{1}{n^2}$

Step 2: (union bound for maximum)
$$p(M^* \ge r^*) = p\left(\bigcup_i X_i \ge r^*\right) \le \sum_i^n p(X_i \ge r^*) \le n \cdot \frac{1}{n^2} = \frac{1}{n}$$

Step 3: (estimate expectation by separating cases)
$$\mathbb{E}[M^*] = \sum_{k=0}^{n} k\, p(M^* = k) = \sum_{k \le r^*}^{\le r^*} k\, p(M^* = k) + \sum_{r^* \ge k}^{\le n} k\, p(M^* = k)$$
$$\le r^* \underbrace{\sum_{k \le r^*} p(M^* = k)}_{\le 1} + n \cdot \sum_{r^* \ge k} p(M^* = k) \le r^* + n \cdot \frac{1}{n} = 2\log n + 1$$

□

If we select 2 bins and put a ball into the less crowded bin, then $E[M^*] \sim \log(\log n)$

2. Randomized Quicksort

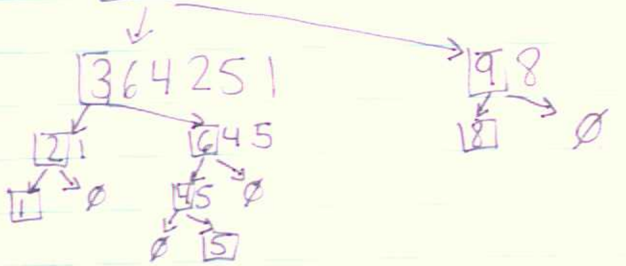
Given an array of numbers $x_1, x_2, \dots, x_n$ quicksort is an algorithm sorting it which works as follows: We compare x_1 to all other numbers and divide them into groups

$$G^- = \{x_i \mid x_i < x_1\} \text{ and } G^+ = \{x_i \mid x_i > x_1\}$$

and recursively sort G^{-1} and G^+.

$$\text{Quicksort}(x_1, \dots x_n) = (\text{QuickSort}(G^-), x_1, \text{QuickSort}(G^+)).$$

Ex. $\boxed{7}$ 3 9 6 4 8 2 5 1



Result: 1 2 3 4 5 6 7 8 9

We'd like to perform few comparisons. In bad cases, when the list is sorted, we need $\sim n^2$ comparisons.

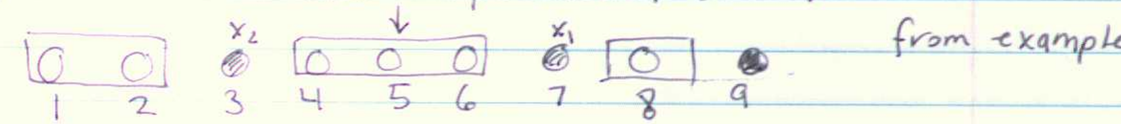
If we randomly reorder the list $\sim n \log n$ comparisons suffice on average.

Theorem:
If the list is reordered randomly then the expected # of comparisons needed is $\leq 2n\log n$

Proof:
Assume that our list is just a permutation of $\{1, 2, \dots, n\}$.
Let T_i be the number of comparisons performed when i^{th} number is processed.

$$T_1 = n-1, \quad T_2 = \begin{cases} |G^-| - 1 & \text{if } x_2 < x_1 \\ |G^+| - 1 & \text{if } x_2 > x_1 \end{cases}$$

We will estimate expectation of T_i.



i numbers have been processed and I can assume that the i^{th} number was selected among the processed one at random.

Out of i choices of i^{th} number each unselected # (empty circle) will be compared to at most 2 of the i possible numbers.

Total number of comparisons over our i choices of i^{th} number $\leq 2(n-i)$

The average number of comparisons $\leq \dfrac{2(n-i)}{i}$

Thus, $E[T_i] \leq \dfrac{2(n-i)}{i}$.

$\mathbb{E}$[total # of comparisons]

$$= \sum_{i=1}^{n} \mathbb{E}[T_i] \leq \sum_{i=1}^{n} \frac{2(n-i)}{i}$$

$$= \sum_{i=1}^{n} \left(\frac{2n}{i} - 2\right)$$

$$= 2n\left(\sum_{i=1}^{n} \frac{1}{i}\right) - 2n$$

$$\leq 2n(\log n + 1) - 2n$$

$$= 2n \log n$$

Note:

$$\sum_{i=1}^{n} \frac{1}{i} \leq \log n + 1$$

□

Any algorithm which only performs comparisons can not be much faster.

Suppose an algorithm always succeeds in k comparisons. So the result of an algorithm and final order can be ordered in k ones and zeros.

But there are $n!$ possible inputs ($n!$ initial orders)

Different initial order should yield different outputs

So $2^k \geq n!$

$$2^k \geq n! \geq \left(\frac{n}{e}\right)^n$$

$$k \log_2 2 \geq n \log_2\left(\frac{n}{e}\right) = n(\log_2 n - \log_2 e)$$

$$k \geq n \log_2 n - n \log_2 e$$

$$\approx (\log_2 e) n \log_2 n \approx 1.44 n \log n$$