Randomized QuickSort:

input: array $A = [a_1, \ldots, a_n]$ of n numbers

- Choose a random element of A to be pivot p.
- Partition A into $A_{<p}, A_{=p}, A_{>p}$
- Recursively sort $A_{<p}$ & $A_{>p}$.
- Return $(A_{<p}, A_{=p}, A_{>p})$

Worst case, $\Omega(n^2)$ time.

Expected # of comparisons $\leq 2n \ln n$.

$\Rightarrow O(n \log n)$ expected running time

What is likely running time?

If p was always the median then $O(n \log n)$ time.

Still good if p is "close" to median.

Consider an element a_i

Let $S_1, S_2, \ldots, S_k$ be the subarrays of A containing a_i in the recursive calls.

So $S_1 = A$ & $S_k = \{a_i\}$.

<u>Goal:</u> show $k = O(\log n)$.

For the j^{th} subproblem, it's a "good" pivot if $|S_{j+1}| < \frac{3}{4}|S_j|$.

(The choice of $\frac{3}{4}$ is arbitrary, anything < 1 is ok.)

Let $X_j = \begin{cases} 1 & \text{if the } j^{th} \text{ pivot is good} \\ 0 & \text{if not good} \end{cases}$

<u>Claim:</u> $Pr(X_j = 1) = 1/2$

<u>Why?</u>

Consider sorted $S_j = \boxed{b_1 \le b_2 \le \cdots \le b_m}$

good pivots are $b_{\frac{m}{4}}, \ldots, b_{\frac{3m}{4}}$ (so $\frac{m}{2}$ good choices

Hence, $Pr\left(\begin{matrix}\text{random pivot}\\ \text{is good}\end{matrix}\right) = \dfrac{\frac{m}{2}}{m} = 1/2.$

If we get $\geq \log_{\frac{4}{3}} n$ good pivots then we're done because $|S_k| \le |S_1| \left(\frac{3}{4}\right)^{\log_{\frac{4}{3}} n}$

$$\le n \times \frac{1}{n} = 1.$$

So we need that:

$$X_1 + X_2 + \cdots + X_k \geq \log_{\frac{4}{3}} n$$

for $k = O(\log n)$.

Set $\ell = 40\log_{\frac{4}{3}}(n)$

What is the probability that a_i is involved in $>\ell$ subproblems?

This means $k > \ell$.

$$\Pr(a_i \text{ is involved in } > \ell \text{ subproblems})$$
$$= \Pr(k > \ell)$$
$$\leq \Pr(X_1 + X_2 + \cdots + X_\ell \leq \log_{\frac{4}{3}} n)$$
$$= \Pr(X_1 + \cdots + X_\ell \leq \ell/40)$$
$$\leq \Pr(X_1 + \cdots + X_\ell \leq \ell/4)$$

Claim: $\leq 2 \times (.68)^{\ell/4}$

$$\leq \frac{1}{n^5}$$

Idea of claim:

$$\Pr\left(X_1 + \cdots + X_\ell \leq \frac{\ell}{4}\right) = \sum_{j=1}^{\ell/4} \binom{\ell}{j}\left(\frac{1}{2}\right)^\ell \leq 2\binom{\ell}{\ell/4}\left(\frac{1}{2}\right)^\ell$$

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

$$\leq 2\left(\frac{\ell e}{\ell/4}\right)^{\ell/4}\left(\frac{1}{2}\right)^\ell \leq 2\left(\frac{4e}{2^4}\right)^{\ell/4} \leq 2(\ldots)^{\ell/4}$$

Hence, $\Pr(a_i \text{ is in } > 40\log n \text{ rounds}) \leq \frac{1}{n^5}$

By union bound, (Define events $\mathcal{E}_1, \ldots, \mathcal{E}_n$ look at $\Pr(\mathcal{E}_1 \cup \cdots \cup \mathcal{E}_n)$,

$\Pr(\text{Some } a_i \text{ is in } > 40\log n \text{ rounds}) \leq \frac{1}{n^4}$

Hence, with probability $\geq 1 - \frac{1}{n^4}$

randomized QuickSort takes $O(n\log n)$ time.

(can make this constant smaller

by ↑ this constant in $O(1)$.)

Will prove the claim in this week's HW.

Hashing, but first toy problem

<u>Balls into bins:</u>

n balls & n bins

Each ball is thrown into a random bin

Let load of bin i = # of balls assigned to bin i.

How large is the max load?
= Maximum # of balls in a bin?

It is $\Theta(\log n)$ with high probability.

$$\Pr(\text{bin } i \text{ has load} > 2\log n)$$

$$\leq \binom{n}{2\log n}\left(\frac{1}{n}\right)^{2\log n}$$

$$= \frac{(n)(n-1)\times\cdots\times(n-2\log n+1)}{(2\log n)!}\left(\frac{1}{n}\right)^{2\log n}$$

$$\leq \frac{n^{2\log n}}{(2\log n)!}\left(\frac{1}{n}\right)^{2\log n}$$

$$\leq \left(\frac{ne}{2n\log n}\right)^{2\log n} \quad \text{since } k! \geq \left(\frac{k}{e}\right)^k$$

$$\leq \left(\frac{1}{2}\right)^{2\log n} \quad \text{for } n \text{ sufficiently large so that } \log n > e.$$

< 1

Let $\mathcal{E}_i$ = event that bin i has load $> 2\log n$

$$\Pr(\underset{\cancel{\text{some}}}{\text{max}} \text{ load} > 2\log n)$$
$$= \Pr(\text{some bin has load} > 2\log n)$$
$$= \Pr(\mathcal{E}_1 \cup \mathcal{E}_2 \cup \cdots \cup \mathcal{E}_n)$$
$$\leq \sum_{i=1}^{n} \Pr(\mathcal{E}_i) \quad (\text{union bound})$$
$$\leq n \times \frac{1}{n^2}$$
$$= \frac{1}{n}$$

Hence, with probability $\geq 1 - \frac{1}{n}$, max load $\leq 2\log n$.

Can improve to show max load is $\leq \dfrac{\log n}{\log\log n}$ with high probability.

Better idea:

Assign balls sequentially to bins.

For $i = 1 \to n$

Choose 2 random bins j & k.

Let $L(j)$ & $L(k)$ denote their current load

If $L(j) < L(k)$ assign ball i to bin j

else assign it to bin k.

(So assign to the least loaded of 2 random bins)

Max load is $O(\log\log n)$ with high probab.

(with d choices, it's $O\left(\frac{\log\log n}{\log d}\right)$).

Chain hashing:

Running example: Want to maintain a list of unacceptable passwords. When someone enters a password, we want to quickly check if its allowed or not.

HUGE set U = universe of possible passwords

hash table of size n: $H = [0, 1, \ldots, n-1]$

Map elements of U into bins $\{0, 1, \ldots, n-1\}$

Using hash function $h: U \to \{0, 1, \ldots, n-1\}$

In chain hashing $H[i]$ is a linked list of elements stored there.

Start with each list as empty

To store $S \subset U$,

for each $x \in S$,

add x into S by

- computing $h(x)$, then
- adding x onto linked list at $H[h(x)]$.

To query is $y \in S$?

- Compute $h(y)$
- check linked list at $H[h(y)]$ to see if it contains y.

Say $m = |S|$, $n = |H|$, then we're throwing m balls into n bins.

The maximum query time is the max load.

When $m=n$ then the max load is $O\left(\frac{\log n}{\log \log n}\right)$.

To get max load $O(1)$ we need ~~m~~ $n=\Omega(m^2)$.

Bloom filters:

also faster & less space

Simpler approach but allow false positives with small probability.

Maintain set $S \subset U$.

- Can insert x into S.
- Can check: is $x \in S$?

  if $x \in S$, we always output YES

  if $x \notin S$, we usually output NO

  but we have a false positive (output YES) with small probability

- Cannot delete x from S

H is a 0-1 array of size n

Start with H as all 0's.

hash function $h: U \to \{0, 1, \ldots, n-1\}$

To add x into S

mark $h(x) = 1$

To check if x is in S

if $h(x) = 1$ then output YES

if $h(x) = 0$ then output NO

Problem:

if $x \notin S$, but we added y into S where $h(x) = h(y)$

Then we get a false positive when we query: is $x \in S$?

Better approach:

k hash functions: $h_1, h_2, \ldots, h_k$

each $h_i : U \to \{0, 1, \ldots, n-1\}$

To add x into S,

for $i = 1 \to k$,

set $H[h_i(x)] = 1$

To check if $x \in S$?

if for all i, $H[h_i(x)] = 1$

then output YES

else output NO