The Hiring Problem

- An agency gives you a list of \( n \) persons
- You interview them one-by-one
- After each interview, you must immediately decide if this candidate should be hired
- You can change your mind if a better one comes up later, but that will cost
The Hiring Problem: straightforward algorithm

Always change your mind if a better one shows up

Given n candidates, how many times will we change our mind?

Worst case: n times
Expected case: ?
**Worst case:** \( n \) times
(better and better candidates)
Expected case: ?

We need **probabilistic analysis**
Probabilistic analysis

Use an indicator variable

\[ X_i = \begin{cases} 
1 & \text{if candidate } i \text{ is hired} \\
0 & \text{otherwise} 
\end{cases} \]

\( E(X_i) \) denotes the expected value of \( X_i \)

Expected number of re-hiring: \( E \left( \sum_{i=1}^{n} X_i \right) = \sum_{i=1}^{n} E(X_i) \)
Use an indicator variable

\[ X_i = \begin{cases} 
1 \text{ if candidate } i \text{ is hired} \\
0 \text{ otherwise} 
\end{cases} \]

\[ E(X_i) \text{ denotes the expected value of } X_i \]

Expected number of re-hiring: 

\[ E\left( \sum_{i=1}^{n} X_i \right) = \sum_{i=1}^{n} E(X_i) \]

\[ E(X_i) = \Pr(\text{candidate } i \text{ is hired}) \]

\[ \Pr(\text{candidate } i \text{ is hired}) = \Pr(\text{candidate } i \text{ is better than all before}) = \frac{1}{i} \]

\[ \sum_{i=1}^{n} E(X_i) = \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1) \]
Probabilistic analysis: problem of "trust"

- We assume that the agency’s list is randomly ordered
- What if not? (If we pay the agency each time we re-hire.....)
Solution: Use a *randomized algorithm*

- Shuffle the list randomly before the interviews
- Now, the probabilistic analysis holds *independent* of the agency’s behaviour
The Hiring Problem: randomized algorithm

1. Permute the list randomly
2. Always change your mind if a better one shows up

Given n candidates, how many times will we change our mind?

(Worst case: n times)

Expected case: $\ln n + O(1)$ times
To think about

1. How do we permute an array randomly?

2. On-line Hiring Problem: We can not change our mind, we can only hire one. How do we do this?
Balls and Bins

Balls are thrown in bins, each ball is thrown randomly and independently

n balls, b bins
Balls and Bins

Balls are thrown in bins, each ball is thrown randomly and independently.

- **n** balls, **b** bins

**Expected number of balls in a given bin:** \( \frac{n}{b} \)

**Expected number of balls until a given bin gets one:** \( b \)

**Expected number of balls until each bin has a ball:** \( b \ln b + O(1) \)

If \( b = n \) (i.e., we throw \( n \) balls):

- **Expected maximum number of balls in a bin:** \( \Theta(\ln n) \)
- **Expected largest number of consecutive empty bins:** \( \Theta(\ln n) \)
- **Expected value of** \( \sum \text{(number of balls)}^2 = \Theta(n) \)
General Idea of Hashing

- Store elements (key, value) in an array
- Use the hash function to determine where each key is stored
- If the hash function is good, the keys are nicely spread
- If two keys have the same hash function, we have a collision, which must be handled
Handling collisions

Chaining

Open addressing

Double hashing:
Start at $h_1$
Jump $h_2$
How do we select a good hash function?
Universal Hash function

- A randomly selected hash function
- Works well with high probability for any set of keys
- Good example of randomized algorithm
Universal Hash function

- Given a set $H$ of hash functions that maps keys into $0...m$.
- If for each pair $(x,y)$ the number of hash functions for which $h(x) = h(y)$ is at most $|H| / m$, then $H$ is universal.
Theorem 11.3

- If we store $n$ keys into a table of size $m$ using chaining, the expected length of the chain containing key $k$ is $n / m$ ($= \alpha$)

- Proof sketch: for each other key, the probability of collision with $k$ is $1 / m$
Finding universal hash functions is easy!

- Class $H_{p,m}$ consists of all hash functions $h(k) = ((ak+b) \mod p) \mod m$

  where
  - $m$ is table size,
  - $p$ is a prime, $p > m$
  - $a$ and $b$ are random numbers

**Theorem 11.5:** $H_{p,m}$ is universal
With universal hashing, \textit{expected} cost per operation is low.

But what if we want the \textit{max} cost per operation to be low?

(Let’s say we wish to construct a static hash table to be stored on a CD-ROM and we want each search to be fast)