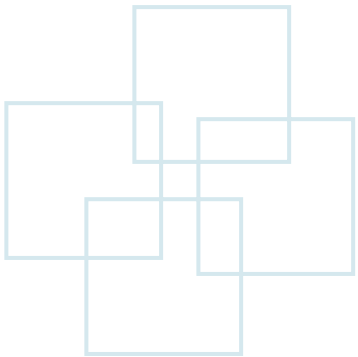


Topic 5: Probabilistic Analysis and Randomized Algorithms





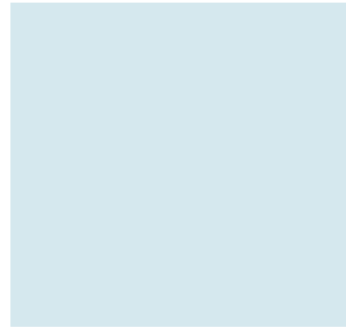
The Primary Goal of This Topic

- Explain the difference between
 - *Probabilistic analysis* and
 - *Randomized algorithms*.
- Present the technique of *indicator random variable*.
- Give an example of the analysis of a randomized algorithm → *Permuting an array in place*.

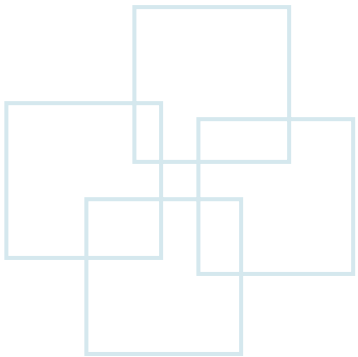


Outline

- The hiring problem
- Indicator random variables
- Randomized algorithms



The Hiring Problem





Scenario

- You are using an employment agency to hire a new office assistant.
- The agency sends you one candidate each day.
- You interview the candidate and must immediately decide whether or not to hire that person.
 - But if you hire, you must also fire your current office assistant even if it's someone you have recently hired.
- Cost to interview is c_i per candidate (interview fee paid to agency).
- Cost to hire is c_h per candidate includes cost to
 - **Fire current office assistant** + **Hiring fee paid to agency.**
- Assume that $c_h > c_i$.
- You are committed to having hired, at all times, the best candidate seen so far.
 - Whenever you interview a candidate who is better than your current office assistant, you must fire the current office assistant and hire the candidate.
 - Since you must have someone hired at all times, you will always hire the first candidate that you interview.



Pseudocode to Model This Scenario

- Assumes that the candidates are numbered 1 to n and that after interviewing each candidate, we can determine if it's better than the current office assistant.
- Uses a *dummy candidate* 0 that is worse than all others, so that the first candidate is always hired.

```
HIRE-ASSISTANT( $n$ )
```

```
 $best = 0$  // candidate 0 is a least-qualified dummy candidate
```

```
for  $i = 1$  to  $n$ 
```

```
    interview candidate  $i$ 
```

```
    if candidate  $i$  is better than candidate  $best$ 
```

```
         $best = i$ 
```

```
    hire candidate  $i$ 
```



Cost

- If n candidates, and we hire m of them, the cost is $O(nc_i + mc_h)$.
 - Have to pay nc_i to interview, no matter how many we hire.
 - So we focus on analyzing the hiring cost mc_h .
 - mc_h varies with each run - it depends on the order in which we interview the candidates.
 - This is a model of a common paradigm:
 - We need to find the maximum or minimum in a sequence by examining each element and maintaining a current “winner.”
 - The variable m denotes how many times we change our notion of which element is currently winning.



Worst-Case Analysis

- In the worst case, we hire all n candidates.
- This happens if each one is better than all who came before.
 - In other words, if the candidates appear in **increasing order** of quality.
 - If we hire all n , then the cost is $O(nc_i + nc_h) = O(nc_h)$ (since $c_h > c_i$).



Probabilistic Analysis

- In general, we have no control over the order in which candidates appear.
- We could assume that they come in a random order:
 - Assign a rank to each candidate: $rank(i)$ is a unique integer in the range 1 to n .
 - The ordered list $\langle rank(1), rank(2), \dots, rank(n) \rangle$ is a permutation of the candidate numbers $\langle 1, 2, \dots, n \rangle$.
 - The list of ranks is equally likely to be any one of the $n!$ permutations.
 - Equivalently, the ranks form a **uniform random permutation**
 - Each of the possible $n!$ permutations appears with equal probability.
- **Essential idea of probabilistic analysis:**
 - We must use knowledge of (or make assumptions about) the distribution of inputs.
 - The **expectation** is over this distribution.
 - This technique requires that we can make a reasonable characterization of the
 - input distribution.



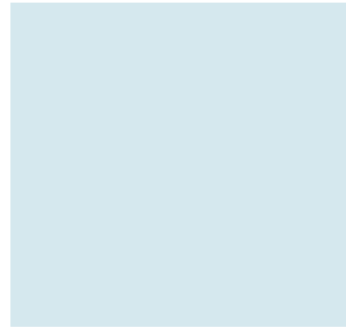
Randomized Algorithms

- We might not know the distribution of inputs, or we might not be able to model it computationally.
 - Instead, we use **randomization** within the algorithm in order to impose a distribution on the inputs.
- ***For the hiring problem***
 - Change the scenario:
 - The employment agency sends us **a list of all n candidates** in advance.
 - On each day, we randomly choose a candidate from the list to interview (but considering only those we have not yet interviewed).
 - Instead of relying on the candidates being presented to us in a random order, we take control of the process and **enforce a random order**.

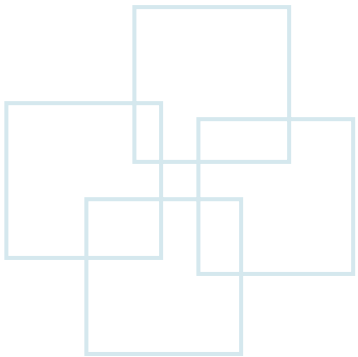


Randomized Algorithms (Cont.)

- An algorithm is *randomized* if its behavior is determined in part by values produced by a *random-number generator*.
 - **RANDOM**(a, b) returns an integer r , where $a \leq r \leq b$ and each of the $b - a + 1$ possible values of r is equally likely.
 - In practice, **RANDOM** is implemented by a *pseudorandom-number generator*, which is a **deterministic method** returning numbers that “*look*” random and pass **statistical tests**.



Indicator Random Variables





Indicator Random Variables

- A simple yet powerful technique for computing *the expected value of a random variable*.
- Helpful in situations in which there may be *dependence*.
- Given a sample space and an event **A**, we define the **indicator random variable**:

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs,} \\ 0 & \text{if } A \text{ does not occur.} \end{cases}$$

- **Lemma**

For an event A , let $X_A = I\{A\}$. Then $E[X_A] = \Pr\{A\}$.

Proof Letting \bar{A} be the complement of A , we have

$$\begin{aligned} E[X_A] &= E[I\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\} \quad (\text{definition of expected value}) \\ &= \Pr\{A\}. \end{aligned}$$



Simple Example

- Determine the expected number of heads when we flip a fair coin one time.
- Sample space is $\{H, T\}$.
- $\Pr\{H\} = \Pr\{T\} = \frac{1}{2}$.
- Define *indicator random variable* $X_H = I\{H\}$.
 - X_H counts the number of heads in one flip.
- Since $\Pr\{H\} = \frac{1}{2}$, **lemma** says that $E[Hx] = \frac{1}{2}$.



Slightly More Complicated Example

- Determine the expected number of heads in n coin flips:
 - Let X be a **random variable** for *the number of heads in n flips*.
 - Compute the expected value: $E[X] = \sum_{k=0}^n k \cdot \Pr\{X = k\}$
(**This calculation is too cumbersome.**)
- Use indicator random variables instead:

For $i = 1, 2, \dots, n$, define $X_i = I\{\text{the } i\text{th flip results in event } H\}$.

Then $X = \sum_{i=1}^n X_i$.

Lemma says that $E[X_i] = \Pr\{H\} = 1/2$ for $i = 1, 2, \dots, n$.

Expected number of heads is $E[X] = E[\sum_{i=1}^n X_i]$.



Slightly More Complicated Example (Cont.)

Problem: We want $E[\sum_{i=1}^n X_i]$. We have only the individual expectations $E[X_1], E[X_2], \dots, E[X_n]$.

Solution: Linearity of expectation says that the expectation of the sum equals the sum of the expectations. Thus,

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^n X_i\right] \\ &= \sum_{i=1}^n E[X_i] \\ &= \sum_{i=1}^n 1/2 \\ &= n/2. \end{aligned}$$

$E[X+Y] = E[X] + E[Y]$
Linearity of expectation applies even when there is dependence among the random variables.



Analysis of the Hiring Problem

- Assume that the candidates arrive in a random order.
- Let X be a random variable that equals the number of times we hire a new office assistant.
 - Define indicator random variables X_1, X_2, \dots, X_n , where $X_i = \mathbb{I} \{ \text{candidate } i \text{ is hired} \}$.

Useful properties:

- $X = X_1 + X_2 + \dots + X_n$.
- Lemma $\Rightarrow E[X_i] = \Pr \{ \text{candidate } i \text{ is hired} \}$.



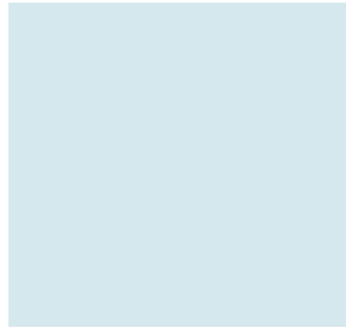
Analysis of the Hiring Problem (Cont.)

- We need to compute $\Pr \{\text{candidate } i \text{ is hired}\}$.
 - Candidate i is hired if and only if candidate i is better than each of candidates $1, 2, \dots, i - 1$.
 - Assumption that the candidates arrive in random order \Rightarrow candidates $1, 2, \dots, i$ arrive in random order \Rightarrow any one of these first i candidates is equally likely to be the best one so far.
 - Thus, $\Pr \{\text{candidate } i \text{ is the best so far}\} = 1/i$.
 - Which implies $E[X_i] = 1/i$.
- The expected hiring cost is $O(c_h \ln n)$ which is much better than the worst-case cost of $O(nc_h)$.

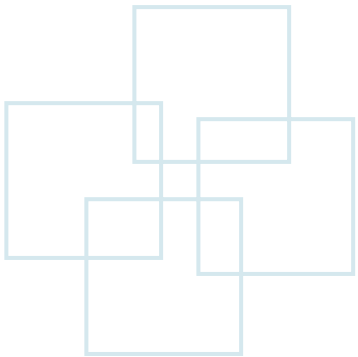
$$\begin{aligned}
 E[X] &= E\left[\sum_{i=1}^n X_i\right] \\
 &= \sum_{i=1}^n E[X_i] \\
 &= \sum_{i=1}^n 1/i \\
 &= \ln n + O(1)
 \end{aligned}$$

Harmonic series:

$$H_n = 1 + 1/2 + 1/3 + \dots + 1/n = \ln n + O(1)$$



Randomized Algorithms





Randomized Algorithms

- Instead of assuming a distribution of the inputs, we impose a distribution.
- ***The hiring problem (revisited)***
 - For the hiring problem, the algorithm is deterministic:
 - For any given input, the number of times we hire a new office assistant will always be the same.
 - The number of times we hire a new office assistant depends only on the input.
 - In fact, it depends only on the ordering of the candidates' ranks that it is given.
 - Some rank orderings will always produce a high hiring cost.
Example: $\langle 1, 2, 3, 4, 5, 6 \rangle$ where each candidate is hired.
 - Some will always produce a low hiring cost.
Example: any ordering in which the best candidate is the first one interviewed. Then only the best candidate is hired.
 - Some may be in between.



Randomized Algorithms (Cont.)

- Instead of always interviewing the candidates in the order presented, what if we first randomly permuted this order?
 - The randomization is now in the algorithm, not in the input distribution.
 - Given a particular input, we can no longer say what its hiring cost will be.
 - Each time we run the algorithm, we can get a different hiring cost.
 - In other words, each time we run the algorithm, the execution depends on the random choices made.
 - No particular input always elicits worst-case behavior.
 - Bad behavior occurs only if we get “unlucky” numbers from the *random number generator*.



Pseudocode for Randomized Hiring Problem

RANDOMIZED-HIRE-ASSISTANT(n)

randomly permute the list of candidates

$best = 0$ // candidate 0 is a least-qualified dummy candidate

for $i = 1$ **to** n

interview candidate i

if candidate i is better than candidate $best$ **HIRE-ASSISTANT**(n)

$best = i$

hire candidate i

- **Lemma**

- The expected hiring cost of RANDOMIZED-HIRE-ASSISTANT is $O(c_h \ln n)$.

- **Proof**

- After permuting the input array, we have a situation identical to the probabilistic analysis of deterministic HIRE-ASSISTANT.



Randomly Permuting an Array

- Two methods are introduced to randomly permute an n -element array:
 - First method: (***Priority-based method***)
 - Assigns a random priority in the range 1 to n^3 to each position and then reorders the array elements into increasing priority order.
 - Second method:
 - n random numbers in the range 1 to n rather than the range 1 to n^3
 - It works in place (unlike the priority-based method).
 - It runs in linear time without requiring sorting.
 - It needs fewer random bits.
- **Goal**
 - Produce a uniform random permutation (each of the $n!$ permutations is equally likely to be produced).



Priority-Based Method

- Assign each element $A[i]$ of the array a random priority $P[i]$, and sort the elements of A according to these priorities.
- **For example:**
 - If our initial array is $A = \langle 1, 2, 3, 4 \rangle$ and we choose random priorities $P = \langle 36, 3, 62, 19 \rangle$, we would produce an array $B = \langle 2, 4, 1, 3 \rangle$.

PERMUTE-BY-SORTING(A)

```

1   $n = A.length$ 
2  let  $P[1..n]$  be a new array
3  for  $i = 1$  to  $n$ 
4       $P[i] = \text{RANDOM}(1, n^3)$ 
5  sort  $A$ , using  $P$  as sort keys
  
```

All entries are unique is at least $1 - 1/n$:
 One unique entry: $(n^3 - n)/n^3 = 1 - 1/n^2$
 N unique entries: $(1 - 1/n^2) \times \dots \times (1 - 1/n^2)$

$O(n \ln n)$



Priority-Based Method (Cont.)

• Lemma

- Procedure PERMUTE-BY-SORTING produces a **uniform random permutation** of the input, assuming that all priorities are distinct.

• Proof

- We start by considering the particular permutation in which each element $A[i]$ receives the i th smallest priority.
- We shall show that this permutation occurs with probability exactly $1/n!$.
 - For $i = 1, 2, \dots, n$, let E_i be the event that element $A[i]$ receives the i th smallest priority. Then we wish to compute the probability that for all i , event E_i occurs, which is

$$\Pr\{E_1 \cap E_2 \cap E_3 \cap \dots \cap E_{n-1} \cap E_n\}$$

this probability is equal to

$$\Pr\{E_1\} \cdot \Pr\{E_2 \mid E_1\} \cdot \Pr\{E_3 \mid E_2 \cap E_1\} \cdot \Pr\{E_4 \mid E_3 \cap E_2 \cap E_1\} \cdot \dots \cdot \Pr\{E_i \mid E_{i-1} \cap E_{i-2} \cap \dots \cap E_1\} \cdot \dots \cdot \Pr\{E_n \mid E_{n-1} \cap \dots \cap E_1\} = 1/n!$$

$$\Pr\{E_1\} = 1/n$$

$$\Pr\{E_2 \mid E_1\} = 1 / (n-1)$$

$$\Pr\{E_3 \mid E_2 \cap E_1\} = 1 / (n-1)$$

$$\Pr\{E_2 \mid E_1\} \cdot \Pr\{E_1\} = \Pr\{E_1 \cap E_2\}$$



A Better Method

- A better method for generating a random permutation is to permute the given array in place.

```
RANDOMIZE-IN-PLACE ( $A, n$ )  
for  $i = 1$  to  $n$   
    swap  $A[i]$  with  $A[\text{RANDOM}(i, n)]$ 
```

- **Idea:**
 - In iteration i , choose $A[i]$ randomly from $A[i..n]$.
 - Will never alter $A[i]$ after iteration i .
- **Time:**
 - $O(1)$ per iteration $\rightarrow O(n)$ total.



A Better Method (Cont.)

- **Correctness**

- Given a set of n elements, a ***k-permutation*** is a sequence containing k of the n elements.

There are $n! / (n-k)!$ possible k -permutations.

- **Lemma**

- RANDOMIZE-IN-PLACE computes a uniform random permutation.

- **Proof** (Use a loop invariant)

Loop invariant: Just prior to the i th iteration of the **for** loop, for each possible $(i - 1)$ -permutation, subarray $A[1 .. i - 1]$ contains this $(i - 1)$ -permutation with probability $(n - i + 1)! / n!$.



A Better Method (Cont.)

Initialization: Just before first iteration, $i = 1$. Loop invariant says that for each possible 0-permutation, subarray $A[1..0]$ contains this 0-permutation with probability $n!/n! = 1$. $A[1..0]$ is an empty subarray, and a 0-permutation has no elements. So, $A[1..0]$ contains any 0-permutation with probability 1.

Maintenance: Assume that just prior to the i th iteration, each possible $(i - 1)$ -permutation appears in $A[1..i - 1]$ with probability $(n - i + 1)!/n!$. Will show that after the i th iteration, each possible i -permutation appears in $A[1..i]$ with probability $(n - i)!/n!$. Incrementing i for the next iteration then maintains the invariant.

Consider a particular i -permutation $\pi = \langle x_1, x_2, \dots, x_i \rangle$. It consists of an $(i - 1)$ -permutation $\pi' = \langle x_1, x_2, \dots, x_{i-1} \rangle$, followed by x_i .

Let E_1 be the event that the algorithm actually puts π' into $A[1..i - 1]$. By the loop invariant, $\Pr\{E_1\} = (n - i + 1)!/n!$.

Let E_2 be the event that the i th iteration puts x_i into $A[i]$.



A Better Method (Cont.)

We get the i -permutation π in $A[1..i]$ if and only if both E_1 and E_2 occur \Rightarrow the probability that the algorithm produces π in $A[1..i]$ is $\Pr\{E_2 \cap E_1\}$.

$$\Rightarrow \Pr\{E_2 \cap E_1\} = \Pr\{E_2 \mid E_1\} \Pr\{E_1\}.$$

The algorithm chooses x_i randomly from the $n - i + 1$ possibilities in $A[i..n]$ $\Rightarrow \Pr\{E_2 \mid E_1\} = 1/(n - i + 1)$. Thus,

$$\begin{aligned} \Pr\{E_2 \cap E_1\} &= \Pr\{E_2 \mid E_1\} \Pr\{E_1\} \\ &= \frac{1}{n - i + 1} \cdot \frac{(n - i + 1)!}{n!} \\ &= \frac{(n - i)!}{n!}. \end{aligned}$$

A randomized algorithm is often the simplest and most efficient way to solve a problem.

Termination: At termination, $i = n + 1$, so we conclude that $A[1..n]$ is a given n -permutation with probability $(n - n)!/n! = 1/n!$

Uniform random permutation