Probabilistic Analysis and Randomized Algorithms

Ref: Chapter 5 of text.

Probabilistic Analysis:

- Assumes that the inputs to the problem are chosen from a known probability distribution.
- The algorithm itself is deterministic.

Randomized Algorithm:

- No assumption about probability distribution of inputs.
- The algorithm uses random numbers (or coin tosses).
- May produce different outputs for the same input at different times.

Problem: Hiring an assistant.

- Number of candidates = \( n \).
- Interview one candidate per day. (Cost of interviewing a candidate = \( \alpha \)).
- If the candidate is better than the current assistant, must fire the assistant and hire the candidate. (Cost of hiring any candidate = \( c_h \)).
- Goal: Estimate the cost of this strategy.

Pseudocode:

HIRE-ASSISTANT \((n)\)
1. Current-Best = 0. /* Dummy. */
2. for \( i = 1 \) to \( n \) do
   3. (a) Interview candidate \( i \).
   4. (b) If candidate \( i \) better than Current-Best, hire candidate \( i \) and set Current-Best to \( i \).
Notes:

1. Since all candidates must be interviewed, the interview cost $n\alpha$ is unavoidable. (This cost is incurred for every input.)

2. If $m$ candidates are hired, the hiring cost is $mc_h$. This cost varies with the input. So, we focus on this cost.

Worst-case analysis:

- Candidate list is in increasing order of quality; that is, candidate $i$ is better than candidate $i - 1$, $1 \leq i \leq n$.
- Every candidate is hired. Therefore, hiring cost = $nc_h$.

Probabilistic analysis:

Assumptions:

- Each candidate $i$ has a rank, denoted by $r(i)$.
- Larger the rank, the better is the candidate.

- Candidates are totally ordered by the ranks; that is, no two candidates have the same rank.
- So, the sequence $\langle r(1), r(2), \ldots, r(n) \rangle$ can be considered as a permutation of $\langle 1, 2, \ldots, n \rangle$.
- Candidates come in a random order. More precisely, the ranks form a uniform random permutation of $\langle 1, 2, \ldots, n \rangle$; that is, each of the $n!$ permutations is equally likely (occurs with probability $= 1/n!$).

Theorem 1: Under the above assumptions, the expected hiring cost is $O(c_h \log n)$.

Proof steps: (Details in class)

- Use indicator random variables; let $x_i$ denote the indicator variable for candidate $i$. (So, $x_i = 1$ if the candidate is hired; $x_i = 0$ otherwise.)
- If $X$ denotes the random variable that represents the number of candidates hired, then $X = \sum_{i=1}^{n} x_i$. We want to compute $E[X]$. 
By linearity of expectation, $E[X] = \sum_{i=1}^{n} E[x_i]$.

$\Pr\{x_i = 1\} = 1/i$. So, $E[x_i] = 1/i$, $1 \leq i \leq n$.

Thus, $E[X] = \sum_{i=1}^{n} (1/i) = H_n$, the $n^{th}$ harmonic number.

Known fact: $H_n = \ln n + O(1)$.

Observation: Theorem 1 indicates that the expected hiring cost $O(ch \log n)$ is much better than the worst-case hiring cost $nc_h$.

A randomized algorithm:

Assumptions:

- The list of candidates is an input to the algorithm.
- Algorithm randomly permutes the list to generate the order in which candidates are interviewed.
- If this is done in such a way that each of the $n!$ permutations is equally likely, then the previous analysis can be used.

Key point: No particular input elicits worst-case behavior. (The algorithm performs badly only when the random permutation generated turns out to be "unlucky").

Pseudocode:

**RANDOMIZED-HIRE-ASSISTANT** ($n$)

1. Randomly permute the list of candidates.
2. Current-Best = 0. /* Dummy. */
3. for $i = 1$ to $n$ do
   (a) Interview candidate $i$.
   (b) If candidate $i$ better than Current-Best, hire candidate $i$ and set Current-Best to $i$.

**Theorem 2:** Under the above assumptions, the expected hiring cost for the randomized algorithm is $O(ch \log n)$. 

Randomly permuting an array:

**Goal:** Each of the $n!$ permutations must be equally likely.

**Assumptions:**
- Array $A[1..n]$ contains some permutation of $\langle 1, 2, \ldots, n \rangle$. (The initial order makes no difference.)
- A function (RANDOM) generating uniformly distributed random numbers is available. In particular, RANDOM($a, b$) generates a random integer in the range $[a .. b]$, where each integer in the range is generated with probability $1/(b - a + 1)$.

**Method I:**
- For each element $A[i]$, generate a random priority value $P[i]$ in the range $[1 .. n^3]$. (We assume that all the priority values are distinct.)

**Exercise:** Show that the probability of having all priorities distinct is at least $1 - (1/n)$.

**Pseudocode:**

**PERMUTE-BY-SORTING** ($A[1..n]$)

1. **for** $i = 1$ **to** $n$ **do**
   
   
   
   $P[i] =$ RANDOM($1, n^3$).

2. Sort $A$ into increasing order using array $P$ as key values.

3. Output $A$.

**Example:** Suppose

$A = \langle 1, 2, 3, 4 \rangle$

$P = \langle 30, 9, 53, 7 \rangle$.

Resulting permutation: $\langle 4, 2, 1, 3 \rangle$.

**Theorem 3:** Assuming that all priority values are distinct, the above algorithm produces a uniform random permutation.
**Proof steps:** (Details in class)
- Show that the identity permutation \(\langle 1, 2, 3, \ldots, n \rangle\) is produced with probability \(= 1/n!\).
- Observe that the same argument works for any permutation.

**Note:** The proof also uses the following lemma on conditional probabilities.

**Lemma 1:** For any collection of events \(X_1, X_2, \ldots, X_n\), the probability that all these events happen, that is, \(\Pr\{\bigcap_{i=1}^{n} X_i\}\), is given by

\[
\Pr\{X_1\} \cdot \Pr\{X_2 \mid X_1\} \cdot \Pr\{X_3 \mid X_1 \cap X_2\} \cdot \Pr\{X_4 \mid X_1 \cap X_2 \cap X_3\} \cdot \cdots \Pr\{X_n \mid X_1 \cap X_2 \cdots \cap X_{n-1}\}.
\]

**Method II:** (Simpler and faster)
- Permutates the given array in place.
- In iteration \(i\), the element \(A[i]\) is chosen randomly from among the elements \(A[i]\) through \(A[n]\).
- Subsequent to iteration \(i\), \(A[i]\) is never altered.

**Pseudocode:**

**RANDOMIZE-IN-PLACE** \((A[1 \ldots n])\)

1. for \(i = 1\) to \(n\) do
   
   Swap \(A[i]\) with \(A[\text{RANDOM}(i, n)]\).

2. Output \(A\).
**Theorem 4:** Algorithm **RANDOMIZED-IN-PLACE** produces a uniform random permutation.

**Proof steps:** (Details in class)

- **Loop invariant:** Just prior to iteration $i$ of the **for** loop, for each possible $(i-1)$-permutation, the subarray $A[1..i-1]$ contains this $(i-1)$-permutation with probability $(n-i+1)!/n!$.
- We show the following:
  - The loop invariant holds prior to the first iteration of the loop.
  - Each iteration of the loop maintains the invariant.