Chapter 5
Probabilistic Analysis

Assume that we are hiring a new programmer through an employment agency. The agency sends us one candidate each day, we will then interview her and decide whether to hire this person.

Regarding the cost, we have to pay the agency a small fee for sending us each candidate, but it is much more costly if we actually hire her, since we have to fire the current programmer, get the new one trained, and also pay a large hiring fee to the head hunting agency.

The total cost actually depends on our hiring strategy, namely, whom do we want to hire (the first, the last, randomly pick one)?, which leads to how the hiring will be done.
One hiring strategy

Assume that we are committed to having the most qualified candidate, all the times, for the job.

The strategy is the following:

1. Hire the first one.

2. From the second on, after interviewing of a candidate, if she is deemed “better” than the current one, fire the current one and hire this better candidate.

**Question:** How much will this strategy cost us?

**Answer:** We will dig this out throughout this chapter.
An implementation

The afore-discussed hiring strategy can be implemented as follows:

HIRE(n)
1. best<-0
   //Candidate 0 is a dummy, worst, candidate
2. for i<-1 to n
3.   do interview candidate i
4. if candidate i is better then best
5. then best<-i
6. hire candidate i

Still the bear keeping the best ear of corn strategy.
Algorithm analysis

This time, we are not interested in the running time of this algorithm, which is clearly $\Theta(n)$. What we want to dig out is the cost incurred by the interviewing and hiring process.

As mentioned before, interviewing has a low cost, $c_i$, while the hiring cost, $c_h$, is high. Let $m$ be the number of people hired, the total cost associated with this algorithm will be $O(nc_i + mc_h)$.

The first item is fixed, since we have to interview this many; while the second item is more interesting, which varies with each run of the algorithm.

What we want to know is the order of $m$, which depends on the probabilistic distribution of the input.
The extreme cases

In the worst case, \textit{when the candidates are sent to us in the increasing order of “quality”}, we will hire everybody, while firing her predecessor, except the last one. Thus, the total hiring cost will be $nc_h$. 😞

It is easy to see that the best case is $c_h$. ☺

Thus, $m = \Omega(1)$ and $m = O(n)$.

These extreme cases are certainly \textit{possible}, but not \textit{probable}. In fact, we have no idea about the order in which those candidates show up, and we actually have no control over this order.

Thus, for this case, we should make a probabilistic analysis of $m$, which leads to an average, or a typical, case.
Probabilistic analysis

This refers to using probability in the analysis of problems. We can use it to analyze the running time of an algorithm, or we can also use it to analyze other quantities, such as the hiring cost associated with the above strategy.

To perform such an analysis of the running time, we have to know, or make an assumption of, the distribution of the input first. (Cf. Exercise 2.2-3) Then, we analyze the algorithm, and compute an expected running time, which is taken over the distribution of the possible inputs.

In fact, we are taking an average of the running time over all the possible inputs. Let’s see a simple example next and a more complicated one in Chapter 7.
The average case of the Insertion sort

Recall the following pseudocode for the insertion sort.

\begin{verbatim}
INSERTION-SORT(A)
1 for j <- 2 to length[A]
2 do key <- A[j]
3 //Insert A[j] into the sorted A[1..j-1].
4 i <- j-1
5 while i>0 and A[i]>key
6 do A[i+1]<-A[i]
7 i<-i-1
8 A[i+1]<-key
\end{verbatim}

Assume that for all \( i \neq j \), \( a[i] \neq a[j] \). Thus, the input list can be regarded as a permutation of \( \{1, 2, \ldots, n\} \).

We further assume that all permutations are \textit{equally likely}, i.e., each and every one of the \( n! \) permutations will appear with the probability of \( 1/n! \).
How much does it take?

Let $C(n)$ and $M(n)$ be the average number of comparisons and movements, respectively, and let $C(j)$ and $M(j)$ be the average number of comparisons and movements, respectively, to be carried out within the inside loop (Lines 5 through 7), to insert $a[j], j \in [2, n]$, into the right place of the sorted list.

We have the following:

$$C(n) = \sum_{j=2}^{n} C(j), \text{and,}$$

$$M(n) = \sum_{j=2}^{n} (M(j) + 2)$$

$$= \left[ \sum_{j=2}^{n} M(j) \right] + 2(n - 1),$$

where the extra two moves in $M(n)$ are done outside the inside loop, as made in Line 2 and 8, respectively.
Case analysis

To begin with, when $a[j] \geq a[j-1]$, $a[j]$ stays in position $j$ ($j-0$). For this case, there is exactly 1 comparison and 0 movement.

If $a[j-2] \leq a[j] \leq a[j-1]$, then, $a[j]$ moves to position $j-1$, where it takes 2 comparisons and 1 movement.

In general (pattern! 🧐), if $a[j]$ moves to position $i$ ($\in [2,j]$), then there are $j-i+1$ comparisons and $j-i$ moves. The previous two cases correspond to $i=j$ and $i=j-1$, respectively.

Finally, if it will be moved to the first position, i.e., $i=1$, there will be $j-1$ comparisons and $j-1$ moves, as all the previous $j-1$ pieces are shifted one position to the right as a result of one comparison.
Now, we know...

\[
C(j) = \frac{1}{j} \left[ \sum_{i=2}^{j} (j + 1 - i) + (j - 1) \right] = \frac{j^2 + j - 4}{2j}, \text{ and}
\]

\[
M(j) = \frac{1}{j} \left[ \sum_{i=2}^{j} (j - i) + (j - 1) \right] = \frac{j - 1}{2}.
\]

Notice that this $\frac{1}{j}$ factor for all the $j$ cases reflects our equal likelihood assumption.

Finally,

\[
C(n) = \frac{n(n + 3)}{4} - 2\mathcal{H}_n + 1 = \frac{n^2}{4} + \Theta(n), \text{ and}
\]

\[
M(n) = \frac{(n - 1)(n + 8)}{4} = \frac{n^2}{4} + \Theta(n).
\]

Recall that from an earlier chapter that the worst cases of both comparison and movement of insertion sort are $\frac{n^2}{2} + \Theta(n)$.  

10
Randomized algorithms

To make a probabilistic analysis of an algorithm, we have to have some knowledge about the input distribution.

We often know little about such distributions in many cases. Even if we know some, it might not be enough for us to model such knowledge in a computational model.

In such cases, we can still use probability and randomness as a tool for algorithm design and analysis, by randomizing part of the behavior of the algorithm.
An example

In the hiring problem, we do not know how the hiring agency will send us the candidates, which might not be random at all.

We can change the situation: instead of getting a candidate from the agency every day, we simply collect all the candidates who have applied by a certain date from the agency, then we randomly select one every day for interview.

Thus, although we still know nothing about those candidates beforehand, we now have a total control of the process and have enforced a random order: *It is equally likely that any candidate will be chosen for interview.*
How to do it?

In general, we call an algorithm randomized if its behavior is determined not only by its inputs, but also by values coming from a random number generator.

We will assume that we have such a generator RANDOM(a, b), that returns an integer within the range of \([a, b)\). Most of the programming languages provide such random number generators, although always a pseudo one.

Java provides one via java.util.Random.

For example, to randomly select an index for a candidate in the range \([0, 99]\), we simply do the following:

```java
randomGenerator.nextInt(100);
```
Random order

Thus, for the hiring problem, we assume that the candidates come in a random order, with the assumption that we can compare the quality of any two candidates.

In particular, we assign \( \text{rank}(i) \), a unique number to candidate \( i \), from 1 through \( n \), to represent her rank. This sequence of \( \text{rank}(i) \) constitutes a permutation of \( \{1, 2, \cdots, n\} \).

Now, when saying the candidates come in a random order, we mean the order they come can be any of the \( n! \) permutations of \( \{1, 2, \cdots, n\} \) with the equal likelihood.

We now need to review a bit about this probability stuff. Still remember anything? 😊
Random variables

Assume that we are given a sample space $S$ and an event $A$. Then the *indicate random variable*, $I(A)$, associated with event $A$ is defined as follows:

$$I(A) = \begin{cases} 1 & \text{if } A \text{ occurs,} \\ 0 & \text{otherwise.} \end{cases}$$

As an example, if we are throwing a coin to the floor, let $H$ stand for the event that “The head is up.” and $T$ for the event “The tail is up.”, then we can define a random variable $X_H$ and

$$X_H = I(Y = H) = \begin{cases} 1 & \text{if } Y = H \\ 0 & \text{if } Y = T. \end{cases}$$

**Question:** What about the average value of $X_H$?

**Answer:** It is the *expectation value* of $X_H$. 
Expectation value

The expected number of heads obtained in one flip of the coin is then simply the expected value of the indicate random variable associated with the event $Y = H$.

$$E[X_H] = E[I(Y = H)] = 1 \times Pr[Y = H] + 0 \times Pr[Y = T] = 1 \times \frac{1}{2} + 0 \times \frac{1}{2} = \frac{1}{2}.$$  

Thus, the expected number of heads obtained in one flip is simply half, confirming our gut feeling.

If we know the following result, we can apply it directly to obtain the last result.

**Lemma 5.1.** Given a sample space $S$ and an event $A$, and let $X_A = I(A)$. Then,

$$E[X_A] = 1 \times Pr[X_A = 1] + 0 \times Pr[X_A = 0] = Pr[A].$$
How about two coins?

**Question:** If we throw, independently, two coins on the floor, what would be the “average value” of the sum of their respective indicate variables, $X_{H_1}$ and $X_{H_2}$?

<table>
<thead>
<tr>
<th>$X_{H_1}$</th>
<th>$X_{H_2}$</th>
<th>$X_{H_1} + X_{H_2}$</th>
<th>$Pr(X_{H_1} + X_{H_2})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1/4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1/4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1/4</td>
</tr>
</tbody>
</table>

Then, we have

$$E[X_{H_1} + X_{H_2}] = 0 \times 1/4 + 1 \times 1/4 + 1 \times 1/4 + 2 \times 1/4 = (0 + 1 + 1 + 2) \times 1/4 = 1.$$  

If we throw $n$ coins on the floor, we would have to work with $2^n$ cases. 😊
A better way...

Let $Y_i$ be the random variable denoting the outcome of the $i^{\text{th}}$ throw of a coin, we have that

$$Y_i = I(Y_i = H).$$

Let $X$ be the random variable denoting the total number of heads in the $n$ throws, we have that

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

By a well-known result in the probability theory, we have that

$$E[X] = E\left[ \sum_{i=1}^{n} Y_i \right] = \sum_{i=1}^{n} E[Y_i] = \sum_{i=1}^{n} \frac{1}{2} = \frac{n}{2}.$$

In particular, when $n = 2$, we have 1, verifying the previous result. 😊

**Homework:** Exercise 5.2-3.
How many to hire?

We assume that the candidates come in a random order, and let $X$ be a random variable whose value equals the number of times we hire a new programmer, thus the total number of programmers that we will hire.

We want to find out $m$, the expected value of $X$.

By definition of the expected value of $X$, we have that

$$E[X] = \sum_{x=1}^{n} x \times Pr[X = x].$$

Since this will lead to a complicated process: What is the probability of hiring one (Exercise 5.2-1), two (Exercise 5.2-2), three (?), ... all of them (Exercise 5.2-1), we use the idea of indicate random variable instead.

**Homework:** Exercise 5.2-1 and think about Exercise 5.2-2.
I hate math...

Let $X_i$ be the indicate variable associated with the event that the $i^{th}$ candidate is hired, i.e.,

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

Thus, $X_i = I(\text{Candidate } i \text{ is hired})$.

By Lemma 5.1,

$$E[X_i] = Pr[\text{Candidate } i \text{ is hired}].$$

Moreover,

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

Thus,

$$m = E[X] = E \left[ \sum_{i=1}^{n} X_i \right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} Pr[\text{Candidate } i \text{ is hired}].$$
Probability of hiring $i$

We now calculate this probability that Candidate $i$ is hired.

Based on the algorithm, she is hired exactly when she is better than each of the candidates 1 through $i - 1$. Since, by assumption, they come in a random order, any one of the first $i$ candidates could be the best qualified one with equal likelihood. Hence, candidate $i$ has a probability of $1/i$ to be the best among the first $i$ candidates, which means

$$Pr[\text{Candidate } i \text{ is hired.}] = \frac{1}{i}. $$

Another way to look at is that

$$Pr[\text{Candidate } i \text{ is hired.}] = \frac{(i-1)!}{i!} = \frac{1}{i}. $$
Finally,...

we can calculate the expected number of hiring as follows:

\[ E[X] = \sum_{i=1}^{n} \frac{1}{i} = \mathcal{H}_n = \ln n + O(1). \]

The above sum is simply the Harmonic sequence, as discussed in Page 6 of the math preview chapter.

Therefore, assuming that the candidates show up in a random order, algorithm \texttt{HIRE(n)} has a total hiring cost of \( O(c_h \ln n) \).

In other words, if we have to interview one million candidates, with our approach, we will hire, on average, 20 candidates.