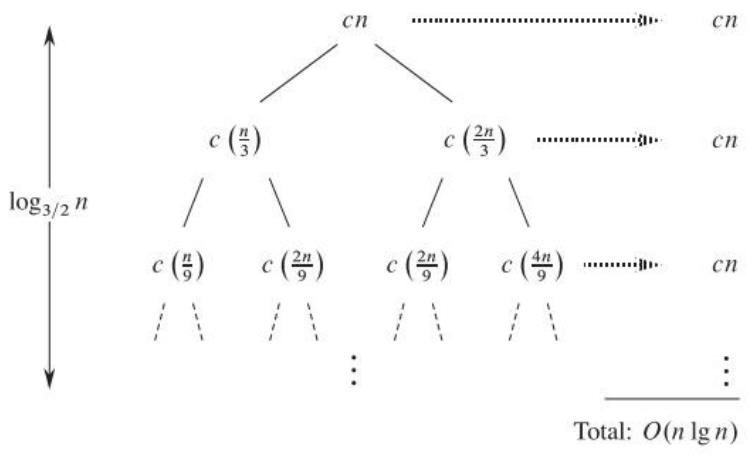
- A more intricate example is the recurrence
$$T(n) = T(n/3) + T(2n/3) + O(n)$$
- c represents the constant in the $O(n)$ term
- Adding the values across the levels of the recursion tree yields value cn for every level
- The longest simple path from the root to a leaf is $n \to (2/3)n \to (2/3)^2 n \to \cdots \to 1$
- $(2/3)^k n = 1$ when $k = \log_{3/2} n$
- The height of the tree is $\log_{3/2} n$



- We expect the solution to the recurrence to be ≤ number of levels × cost of each level:

$$O(cn \log_{3/2} n) = O(n \lg n)$$

- Not every level contributes a cost of cn
- If this recursion tree were a complete binary tree of height $\log_{3/2} n$, there would be $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$ leaves
- The cost of each leaf is a constant
- The total cost of all leaves would then be $\Theta(n^{\log_{3/2} 2})$ which, since $\log_{3/2} 2 > 1$, is $\omega(n \lg n)$

- This recursion tree is not a complete binary tree, however, and has fewer than $n^{\log_{3/2} 2}$ leaves
- As we go down from the root, more and more internal nodes are absent
- Consequently, levels toward the bottom contribute less than cn to the total cost
- we can use the substitution method to verify that $O(n \lg n)$ is an upper bound for the solution to the recurrence

4.5 The master method

Theorem 4.1 (***Master theorem***)

Let $a \geq 1$ and $b > 1$, and $f(n)$ a function, $T(n)$ is defined on nonneg. integers by recurrence

$$T(n) = aT(n/b) + f(n),$$

where we n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$ ■

- In each of the three cases, we compare the function $f(n)$ with the function $n^{\log_b a}$
- The larger of the two determines the solution to the recurrence
  - In case 1, the function $n^{\log_b a}$ is the larger, then the solution is $T(n) = \Theta\left(n^{\log_b a}\right)$
  - In case 3, the function $f(n)$ is the larger, then the solution is $T(n) = \Theta(f(n))$
  - In case 2, the two functions are the same size, we multiply by a logarithmic factor, and the solution is $T(n) = \Theta\left(n^{\log_b a} \lg n\right)$

- Be aware of some technicalities:
- In case 1, not only must $f(n)$ be smaller than $n^{\log_b a}$, it must be polynomially smaller
- $f(n)$ must be asymptotically smaller than $n^{\log_b a}$ by a factor of n^{ϵ} for some $\epsilon > 0$
- In case 3, not only must $f(n)$ be larger than $n^{\log_b a}$, it also must be polynomially larger and satisfy the "regularity" condition that $af(n/b) \leq cf(n)$

Using the master method

$T(n) = 9T(n/3) + n$

- We have $a = 9$, $b = 3$, $f(n) = n$, and thus $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$
- Since $f(n) = O(n^{\log_3 9 - \epsilon})$, where $\epsilon = 1$, we can apply case 1 of the master theorem and conclude that the solution is $T(n) = \Theta(n^2)$

$T(n) = T(2n/3) + 1$

- Here $a = 1$, $b = 3/2$, $f(n) = 1$, and $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$
- Case 2 applies, since $f(n) = \Theta\left(n^{\log_b a}\right) = \Theta(1)$, and thus the solution to the recurrence is $T(n) = \Theta(\lg n)$

$T(n) = 3T(n/4) + n \lg n$

- We have $a = 3$, $b = 4$, $f(n) = n \lg n$, and $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
- Since $f(n) = \Omega(n^{\log_4 3 + \epsilon})$, where $\epsilon \approx 0.2$, case 3 applies provided that the regularity condition holds for $f(n)$
- For sufficiently large n, we have that $af(n/b) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n = cf(n)$ for $c = 3/4$
- By case 3, the solution is $T(n) = \Theta(n \lg n)$

- The master method does not apply to
$$T(n) = 2T(n/2) + n \lg n$$
even if it seems to have the proper form: $a = 2$, $b = 2$, $f(n) = n \lg n$, and $n^{\log_b a} = n$
- One might think that case 3 applies, since $f(n)$ is asymptotically larger than $n^{\log_b a}$
- It is, however, not *polynomially* larger: ratio $f(n)/n^{\log_b a} = \lg n$ is asymptotically less than n^{ϵ} for any positive constant ϵ
- Consequently, the recurrence falls into the gap between case 2 and case 3

- The recurrence of the running time of the simple divide-and-conquer algorithm for matrix multiplication

$$T(n) = 8T(n/2) + \Theta(n^2)$$

- Now $a = 8$, $b = 2$, and $f(n) = \Theta(n^2)$ and so $n^{\log_b a} = n^{\log_2 8} = n^3$
- Since n^3 is polynomially larger than $f(n)$
  – i.e., $f(n) = O(n^{3-\epsilon})$ for $\epsilon = 1$
  case 1 applies, and $T(n) = \Theta(n^3)$

- The recurrence of the running time of Strassen's algorithm

$$T(n) = 7T(n/2) + \Theta(n^2)$$

- Here we have $a = 7$, $b = 2$, and $f(n) = \Theta(n^2)$ and so $n^{\log_b a} = n^{\log_2 7} = n^{\lg 7}$
- Recalling that $2.80 < \lg 7 < 2.81$, we see that $f(n) = O(n^{\lg 7 - \epsilon})$ for $\epsilon = 0.8$
- Again, case 1 applies, and we have the solution $T(n) = \Theta(n^{\lg 7})$

5 Probabilistic Analysis and Randomized Algorithms

- We need to hire a new office assistant with the help of an employment agency that sends us a candidate each day
- We interview that person and then decide either to hire her or not
- We pay the agency a fee for each interview
- To actually hire an applicant is more costly
  - We must fire the current office assistant and
  - pay a substantial hiring fee to the agency

5.1 The hiring problem

- We are committed to having, at all times, the best possible person for the job
- After interviewing an applicant, if she is better qualified than the current assistant, we will
  - fire the current office assistant and hire the new applicant
- We are willing to pay the price of this strategy, but wish to estimate what that price will be

HIRE-ASSISTANT(n)

1. $best \leftarrow 0$ // candidate 0 is a least-qualified dummy candidate
2. **for** $i \leftarrow 1$ **to** n
3. interview candidate i
4. **if** candidate i is better than $best$
5. $best \leftarrow i$
6. hire candidate i

- Interviewing has a low cost, c_i, whereas hiring is expensive, costing c_h
- Letting m be the number of people hired, the total cost of the algorithm is $O(c_i n + c_h m)$
- No matter how many people we hire, we always interview n candidates and thus always incur the cost $c_i n$
- Let us concentrate on analyzing $c_h m$, the hiring cost
- This quantity varies with each run of the algorithm

Worst-case analysis

- In the worst case, we hire every candidate that we interview
  - candidates come in increasing order of quality
  - we hire n times, for a total hiring cost of $c_h n$
- We have no idea about the order in which the candidates arrive, nor do we have any control over this order
- It is natural to ask what we expect to happen in a typical or average case

Probabilistic analysis

- We can assume that the applicants come in a random order
- Assume that we can compare any two candidates and decide the better qualified
  - there is a total order on the candidates
- Rank each candidate with a unique number $1, \ldots, n$
- Use $rank(i)$ to denote the rank of applicant i

- A higher rank corresponds to a better qualified applicant
- Ordered list $\langle rank(1), rank(2), \dots, rank(n) \rangle$ is a permutation of the list $\langle 1,2, \dots, n \rangle$
- "The applicants come in a random order" ≡
  - "this list of ranks is equally likely to be any one of the $n!$ permutations of the numbers 1 through n"
- Alternatively, the ranks form a uniform random permutation; i.e., each of the possible $n!$ permutations appears with equal probability

Randomized algorithms

- We call an algorithm ***randomized*** if its behavior is determined also by values produced by a random-number generator RANDOM
- A call to RANDOM(a, b) returns an integer between a and b, inclusive, with each such integer being equally likely
- E.g., RANDOM(0,1) produces 0 with probability $1/2$, and 1 with probability $1/2$

- A call to RANDOM(3,7) returns either 3,4,5,6,7, each with probability $1/5$
- Each integer returned by RANDOM is independent of the integers returned on previous calls
- In analyzing the running time of a randomized algorithm, we take the expectation of the running time over the distribution of values returned by the RANDOM
- We refer to the running time of a randomized algorithm as an *expected running time*

5.2 Indicator random variables

- Indicator random variables provide a convenient method for converting between probabilities and expectations
- Suppose we are given a sample space S and an event A
- Then the indicator random variable $I\{A\}$ associated with A is defined as

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

- Let us determine the expected number of heads that we obtain when flipping a fair coin
- Our sample space is $S = \{H, T\}$, with
$$\Pr\{H\} = \Pr\{T\} = 1/2$$
- We define an indicator random variable X_H, associated with the coin coming up heads (H)
- This variable counts the number of heads obtained in this flip
$$X_H = I\{H\} = \begin{cases} 1 & \text{if } H \text{ occurs} \\ 0 & \text{if } T \text{ occurs} \end{cases}$$

- Recall that $\mathrm{E}[X] = \sum_x x \cdot \Pr\{X = x\}$
- The expected number of heads obtained in one flip of the coin is simply the expected value of our indicator variable X_H
$$\begin{aligned} \mathrm{E}[X_H] &= \mathrm{E}[I\{H\}] \\ &= 1 \cdot \Pr\{H\} + 0 \cdot \Pr\{T\} \\ &= 1 \cdot (1/2) + 0 \cdot (1/2) \\ &= 1/2 \end{aligned}$$
- Thus the expected number of heads obtained by one flip of a fair coin is $1/2$

- The expected value of an indicator random variable associated with an event A is equal to the probability that A occurs

Lemma 5.1 *Given a sample space S and an event A in S, let $X_A = I\{A\}$. Then $\mathrm{E}[X_A] = \Pr\{A\}$*

Proof By the definition of an indicator and the definition of expected value, we have $E[X_A] = E[I\{A\}] = 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\} = \Pr\{A\}$, where $\bar{A}$ denotes $S - A$, the complement of A ■

- Compute the number of heads in n coin flips
- Let X_i be the indicator associated with the event in which the ith flip comes up heads:
  $X_i = I\{\text{the } i\text{th flip results in the event } H\}$
- Let X denote the total number of heads in the n coin flips, so that
$$X = \sum_{i=1}^{n} X_i$$
- To compute the expected number of heads, take the expectation of both sides to obtain
$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n} X_i\right]$$

- Linearity of expectation

$$\mathrm{E}[X_1 + X_2] = \mathrm{E}[X_1] + \mathrm{E}[X_2]$$

  makes the use of indicator random variables a powerful analytical technique
  - it applies even when there is dependence among the variables
- We now can easily compute the expected number of heads:

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathrm{E}[X_i]$$

$$\sum_{i=1}^{n} 1/2 = n/2$$

Analysis of the hiring problem

- Let X be the random variable whose value ≡ number of times we hire a new assistant
- let X_i be the indicator random variable associated with the event in which the ith candidate is hired

$$X_i = I\{\text{candidate } i \text{ is hired}\}$$

$$= \begin{cases} 1 & \text{if candidate } i \text{ is hired} \\ 0 & \text{if candidate } i \text{ is not hired} \end{cases}$$

- $X = X_1 + X_2 + \cdots + X_n$

- By Lemma 5.1

$$\mathrm{E}[X_i] = \Pr\{\text{candidate } i \text{ is hired}\}$$

and we must compute the probability that lines 5–6 of HIRE-ASSISTANT are executed

- Candidate i is hired (line 6) exactly when she is better than each of candidates $1, \ldots, i-1$
- We assume that the candidates arrive in a random order: the first i candidates have appeared in a random order
- Any one of these first i candidates is equally likely to be the best-qualified so far

- Candidate i has a probability of $1/i$ of being better qualified than candidates $1, \ldots, i-1$ and thus a probability of $1/i$ of being hired.
- By Lemma 5.1, we conclude that $\mathrm{E}[X_i] = 1/i$
- Now we can compute $\mathrm{E}[X]$:

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathrm{E}[X_i]$$

$$= \sum_{i=1}^{n} 1/i = \ln n + O(1)$$

by harmonic series

$$H_n = \sum_{k=1}^{n} 1/k = \ln n + O(1)$$

- We interview n people, but actually hire only approx. $\ln n$ of them, on average
- The following lemma summarize this

Lemma 5.2 *Assuming that candidates come in random order, algorithm* HIRE-ASSISTANT *has an average-case total hiring cost of* $O(c_h \ln n)$

Proof Follows immediately from our definition of the hiring cost and the equation, which shows that the expected number of hires is approximately $\ln n$ ■

5.3 Randomized algorithms

- The algorithm above is deterministic
- The number of times we hire a new office assistant differs for different inputs
- Given the rank list $A_1 = \langle 1,2,3,4,5,6,7,8,9,10 \rangle$, a new assistant is always hired 10 times
- For the list of ranks $A_2 = \langle 10,9,8,7,6,5,4,3,2,1 \rangle$, a new assistant is hired only once
- $A_3 = \langle 5,2,1,8,4,7,10,9,3,6 \rangle$, a new assistant is hired three times

- Consider an algorithm that first permutes the input before determining the best candidate
- Now we randomize in the algorithm, not in the input distribution
- For input A_3 we cannot say how many times the best is updated – it differs with each run
- Not even an enemy can produce a bad input, random permutation yields input order irrelevant
- The algorithm performs badly only if the random-number generator produces an "unlucky" permutation

RANDOMIZED-HIRE-ASSISTANT(n)

1. randomly permute the list of candidates
2. *best* $\leftarrow$ 0 // candidate 0 is a least-qualified // dummy candidate
3. **for** $i \leftarrow 1$ **to** n
4. interview candidate i
5. **if** candidate i is better than candidate *best*
6. *best* $\leftarrow i$
7. hire candidate i

Randomly permuting arrays

- We assume that we are given an array A which, w.l.o.g., contains the elements $1, 2, \ldots, n$
- Produce a random permutation of the array
- Assign element $A[i]$ a random priority $P[i]$, and sort the elements according to priorities
- E.g., 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 array $B = \langle 2,4,1,3 \rangle$
- We call this procedure PERMUTE-BY-SORTING

PERMUTE-BY-SORTING(A)

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

- We use a range of 1 to n^3 for random numbers to make it likely that all the priorities in P are unique

- It remains to prove that the procedure produces a ***uniform random permutation***,
  - It is equally likely to produce every permutation of the numbers 1 through n

Lemma 5.4 *PERMUTE-BY-SORTING produces a uniform random permutation of the input, assuming that all priorities are distinct*

Proof See the book.

- It is better to permute the given array in place
- RANDOMIZE-IN-PLACE does so in $O(n)$ time
- In its ith iteration, it chooses the element $A[i]$ randomly from among elements $A[i]$ through $A[n]$
- After the ith iteration, $A[i]$ is never altered

RANDOMIZE-IN-PLACE(A)

1 $n \leftarrow A.length$
2 **for** $i \leftarrow 1$ **to** n
3 swap $A[i]$ with A[RANDOM(i, n)]

- A k-permutation on a set of n elements is a sequence containing k of the n elements, with no repetitions
- The number of k-permutations is $n!/(n-k)!$

loop invariant:

- Just prior to the ith iteration of the for loop of lines 2–3, for each possible $(i-1)$-permutation of the n elements, the subarray $A[1..i-1]$ contains this $(i-1)$-permutation with probability $(n-i+1)!/n!$

Initialization: loop invariant trivially holds

Maintenance:

- Consider a particular i-permutation, and denote the elements in it by $\langle x_1, x_2, \ldots, x_i \rangle$
- This permutation consists of an $(i-1)$-permutation $\langle x_1, x_2, \ldots, x_{i-1} \rangle$ followed by the value x_i that the algorithm places in $A[i]$
- Let E_1 denote the event in which the first $(i-1)$ iterations have created the particular $(i-1)$-permutation $\langle x_1, x_2, \ldots, x_{i-1} \rangle$ in $A[1..i-1]$

- By the loop invariant, $\Pr\{E_1\} = (n - i + 1)!/n!$
- Let E_2 be the event that ith iteration puts x_i in position $A[i]$
- The i-permutation $\langle x_1, x_2, \ldots, x_i \rangle$ appears in $A[1..i]$ precisely when both E_1 and E_2 occur

$$\Pr\{E_2 \cap E_1\} = \Pr\{E_2|E_1\}\Pr\{E_1\}$$

- $\Pr\{E_2|E_1\} = 1/(n - i + 1)$ because in line 3 the algorithm chooses x_i randomly from the $n - i + 1$ values in positions $A[i..n]$

$$\Pr\{E_2 \cap E_1\} = \frac{1}{n - i + 1} \cdot \frac{(n - i + 1)!}{n!} = \frac{(n - i)!}{n!}$$

Termination:

- At termination, $i = n + 1$, and we have that the subarray $A[1..n]$ is a given n-permutation with probability

$$(n - (n + 1) + 1)!/n! = 0!/n! = 1/n!$$

- Thus, RANDOMIZE-IN-PLACE produces a uniform random permutation

5.4.1 The birthday paradox

- How many people must there be in a room before there is a 50% chance that two of them were born on the same day of the year?

- The answer is surprisingly few
- The paradox is that it is in fact far fewer
  - than the number of days in a year, or
  - even half the number of days in a year

An analysis using indicator random variables

- We use indicator random variables to provide a simple but approximate analysis of the birthday paradox
- For each pair (i, j) of the k people in the room, define the indicator random variable X_{ij}, for $1 \leq i < j \leq k$, by

$$X_{ij} = I\{ i \text{ and } j \text{ have the same birthday}\}$$
$$= \begin{cases} 1 & i \text{ and } j \text{ have the same birthday} \\ 0 & \text{otherwise} \end{cases}$$

- Once birthday b_i for i is chosen, the probability that b_j is chosen to be the same day is $1/n$, where $n = 365$

$$\mathrm{E}[X_{ij}] = \Pr\{i \text{ and } j \text{ have the same birthday}\} = 1/n$$

- Let X be a random variable counting the number of pairs of individuals having the same birthday

$$X = \sum_{i=1}^{k} \sum_{j=i+1}^{k} X_{ij}$$

- Taking expectations of both sides and applying linearity of expectation, we obtain

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=1}^{k} \sum_{j=i+1}^{k} X_{ij}\right] = \sum_{i=1}^{k} \sum_{j=i+1}^{k} E[X_{ij}]$$
$$= \binom{k}{2}\frac{1}{n} = \frac{k(k-1)}{2n}$$

- When $k(k-1) \geq 2n$, the expected number of pairs of people with the same birthday is at least 1

- Thus, if we have at least $\sqrt{2n} + 1$ individuals in a room, we can expect at least two to have the same birthday
- For $n = 365$, if $k = 28$, the expected number of pairs with the same birthday is
$$(28 \cdot 27)/(2 \cdot 365) \approx 1.0356$$
- With at least 28 people, we expect to find at least one matching pair of birthdays
- Mars has $n = 687$ days, need $k = 38$ aliens
- Analysis using only probabilities gives a different exact number of people, but same asymptotically: $\Theta(\sqrt{n})$