Module 6: Analysis of Algorithms

Reading from the Textbook: Chapter 4 Algorithms

Introduction

The focus of this module is the mathematical aspects of algorithms. Our main focus is *analysis of algorithms*, which means evaluating efficiency of algorithms by analytical and mathematical methods. We start by some simple examples of worst-case and average-case analysis. We then discuss the formal definitions for asymptotic complexity, used to characterize algorithms into different classes. And we present examples of asymptotic analysis.

The next module deals with recursive algorithms, their correctness proofs, analysis of algorithms by recurrence equations, and algorithmic divide-and-conquer technique.

Contents

Worst-case and Average-Case Analysis: Introductory Examples

Sequential Search Algorithm

Suppose we have an array of *n* elements $A[1{:}n]$, and a key element, *KEY.* The following program is a simple loop which goes through the array sequentially until it either finds the key, or determines that it is not found. (dtype is the type declaration for the array.)

```
int SequentialSearch (dtype $A[\,]$, int $n$, dtype $KEY$) {
for $i = 1$ $to$ $n$ {
    $if$ ($KEY == A[i]$)
        return ($i$);
    };
return ($-1$);   //The for-loop ended and the key was not found.
}
```

Let us analyze the number of key-comparisons (highlighted) in this algorithm. By counting this dominant operation, basically we are counting the number of times the loop is executed. There are two common ways of doing the analysis:

- Worst-case analysis: This determines the maximum amount of time the algorithm would ever take. This analysis is usually easier than the average-case analysis. At the same time, usually it is a good reflection of overall performance.
- Average-case analysis: This method determines the overall average performance. It considers all possible cases, assigns a probability to each case, and computes the weighted average (called *expected value*) for the random variable (in this case, the number of key-comparisons).

Worst-Case Analysis of Sequential Search

The worst-case number of key-comparison in this algorithm is obviously n. This happens if the key is found in the last position of the array, or if it is not found anywhere.

Since each iteration of the for-loop takes at most some constant amount of time, *C,* then the total worst-case time of the algorithm is

$$T(n) \leq Cn + D.$$

(The constant D represents the maximum amount of time for all statements that are executed only once, independent of the variable n.) This total time is characterized as "order of" n, denoted as $O(n)$. (We will shortly see the formal definition for the order.)

Average-Case Analysis of Sequential Search

Now let us compute the average number of key-comparisons. As a first estimate, one may think that since the worst-case number is n, and the best-case is 1 (found right away), then the average must be about $n/2$. Let us do a careful analysis and see how good this estimate is.

First, as a quick review of "expected value", suppose a random variable has the possible values $\{1,2,3\}$ with the following probabilities.

Value of the random variable r	Probability P_r
1	0.1
2	0.1
3	0.8

Then the *expected value* of r is

$$1 * 0.1 + 2 * 0.1 + 3 * 0.8 = 2.7$$

We may also refer to this as the *weighted average*. Note that a straight average (when there is no probability involved) would be simply $(1 + 2 + 3)/3 = 2$.

Now, to compute the expected value of the number of key-comparisons for the algorithm, let

$P =$ Probability that the key is found somewhere in the array

and let

$P_i =$ Probability that the key is found in position i of the array, $1 \leq i \leq n$.

Assuming that the array is random, a common assumption is that when the key is found, then it is *equally likely* that it is found in any of the n positions. So,

$$P_i = \frac{P}{n}, \quad \forall i.$$

Finally, the probability that the key is not found is

$$Q = 1 - P$$

So, the expected number of key-comparisons in this algorithm is:

$$\begin{aligned} f(n) &= \sum_{i=1}^{n} P_i \cdot i + Q \cdot n \\ &= \sum_{i=1}^{n} \frac{P}{n} \cdot i + (1-P) \cdot n \\ &= \frac{P}{n} \cdot \sum_{i=1}^{n} i + (1-P) \cdot n \qquad \text{(Use arithmetic sum formula)} \\ &= \frac{P}{n} \cdot \frac{n(n+1)}{2} + (1-P) \cdot n \\ &= P \cdot \frac{n+1}{2} + (1-P) \cdot n \end{aligned}$$

In the special case when $P = 1$, the expected number of comparisons is $\frac{n+1}{2}$, which agrees with our initial estimate. Otherwise, there is an additional term that takes into account the additional event when the key is not found. For example, if $P = {}^{1}/_{2}$, then the expected number of comparisons becomes $\frac{3n+1}{4}$.

Finding Max and Min of an Array

The following pseudocode is a simple program loop that finds the maximum and minimum elements in an array of n elements, $A[1{:}n]$. (*Max* and *Min* are the returned parameters.)

```
FindMaxMin (dtype $A[\,]$, int $n$, dtype $Max$, dtype $Min$) {
$Max = A[1]$;  $Min = A[1]$;
for  $i = 2\ to\ n$ {
    if $(A[i] > Max)$
        $Max = A[i]$;
    else if $(A[i] < Min)$
        $Min = A[i]$;
  }
```

In iteration i of the for-loop, $A[i]$ is first compared against Max. If $A[i]$ is greater, then Max is updated. Otherwise, a second comparison is made against Min, and if $A[i]$ is smaller, then Min is updated.

Let us analyze the worst-case, best-case, and average-case number of key-comparisons. (The key comparisons are highlighted.)

Worst-Case and Best-Case Analysis

In the worst-case, every iteration of the loop makes two comparisons. (This happens if the first element of the array has the largest value.) So the worst-case number of comparisons is $2(n-1)$.

In the best-case, every iteration makes only one comparison, so the best-case number of comparisons is $(n-1)$. This happens if the input is in sorted order.

Average-Case Analysis

The number of comparisons in each iteration of the loop is 2 in the worst-case, and 1 in the best-case. So is it a good estimate to figure that on the average, the number is 1.5 per iteration?! No, the average is not always half-way between the worst and the best. (If I buy a lottery ticket, do I have a 50-50 chance of winning the jackpot?!) We will prove that it is much more likely to make two comparisons per iteration. As a result, the expected number of comparisons is very close to the worst-case number.

Let us assume the array is random and the elements are all distinct. Iteration i of the loop compares element $A[i]$ against the current Max and possibly Min. Let us define the *prefix sequence* of $A[i]$ as the sequence of elements in the array starting with $A[1]$ and ending with $A[i]$ itself.

$$A[1], A[2], \cdots, A[i]$$

Since the array is random, element $A[i]$ is equally likely to be the smallest in its prefix, or the second smallest, or third smallest, ···, or the largest. So, the probability that $A[i]$ is the largest in its prefix sequence is

$$\frac{1}{i}$$

And the probability that $A[i]$ is not the largest in its prefix sequence is

$$\frac{i-1}{i}$$

If $A[i]$ is the largest in its prefix, iteration i makes only one comparison. Otherwise, iteration i makes two comparisons. Therefore, the expected number of comparisons is

$$f(n) = \sum_{i=2}^{n} \left(\frac{1}{i} \cdot 1 + \frac{i-1}{i} \cdot 2\right) = \sum_{i=2}^{n} \left(2 - \frac{1}{i}\right) = 2(n-1) - \sum_{i=2}^{n} \frac{1}{i} = 2n - 1 - \sum_{i=1}^{n} \frac{1}{i}$$

The latter summation is known as the Harmonic series H_n, and the value of the sum is approximately $\ln n$. (Here, the logarithm is the natural log.)

$$H_n = \sum_{i=1}^{n} \frac{1}{i} \cong \ln n$$

Therefore,

$$f(n) \cong 2n - 1 - \ln n.$$

Since the value of the log is negligible compared to $2n$, the expected number of comparisons is indeed very close to the worst-case value, as stated earlier. For example, if $n = 1000,$ the expected number of key-comparison is about 1992, and the worst-case number $(2n - 2)$ is 1998.

(**Note**: Homework problems explore a technique for finding approximate value of a summation by converting the summation into integral. This technique may be used to find the approximate value for the Harmonic sum.)

Definitions of Asymptotic Complexity

When we study the running time of an algorithm, our focus is on the performance when the problem size gets large. This is because the performance for small problem sizes hardly matters since the running time will be very small anyway.

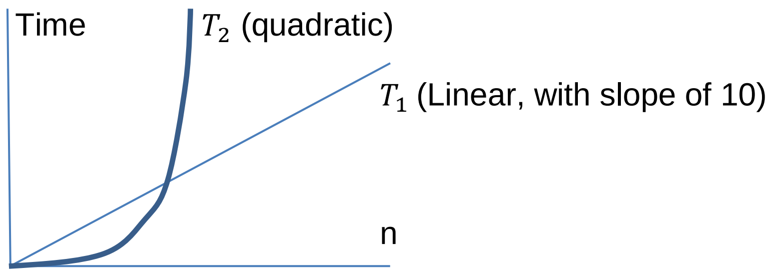
Suppose there are two algorithms for a problem of size *n* with the running times, respectively

$$T_1(n) = 10\, n,$$
$$T_2(n) = 2\, n^2$$

Which one of the two is faster (smaller) running time? Let's tabulate these two functions for some values of *n*.

$\boldsymbol{n}$	$\boldsymbol{10n}$	$\boldsymbol{2n^2}$
1	10	2
2	20	8
5	50	50
10	100	200
100	1,000	20,000
1,000	10,000	2,000,000
10,000	100,000	200,000,000
100,000	1×10^6	2×10^{10}

We observe that initially, for $n < 5$, T_1 is larger than T_2. The two equal at $n = 5$. And after that, as *n* gets larger, T_2 gets much larger than T_1. This may also be observed pictorially, by looking at the graphs of these functions (time-versus-*n*).



The quadratic function T_2 starts smaller than the linear one T_1. The two cross at $n = 5$. After that, T_2 starts growing much faster than T_1. As *n* gets larger and larger, T_2 gets much larger than T_1. We say that T_2 has a *faster growth rate* than T_1, or that T_2 is *asymptotically* larger than T_1.

The fact that T_1 has a slower growth rate than T_2 is due to the fact that T_1 is a linear function n and T_2 is a quadratic function n^2. The coefficients (also called constant factors) are not as critical in this comparison. For example, suppose we have a different pair of coefficients:

$$T_1(n) = 50\,n,$$
$$T_2(n) = n^2$$

The cross point between the two functions now is $n = 50$, and after that T_2 starts growing much faster than T_1 again.

So, we want the asymptotic complexity definitions to incorporate two issues:

1. Focus on large problem size, *n*. (Ignore small values of *n*.)
2. Ignore the constant coefficients (constant factors).

Before making the formal definitions, we consider one more example.

$$T(n) = 2\,n^2 + 10\,n + 20$$

Let us see how this function behaves as n gets large, by tabulating the function for some increasing values of n.

n	$2\,n^2$	$10\,n$	20	$T(n)$	$T(n)/n^2$
1	2	10	20	32	32
10	200	100	20	320	3.20000
100	20,000	1,000	20	21,020	2.10200
1,000	2,000,000	10,000	20	2,010,020	2.01002
10,000	200,000,000	100,000	20	200,100,020	2.00100
100,000	20,000,000,000	1,000,000	20	20,001,000,020	2.00010

From the last column, observe that as n gets larger, the value of $T(n)$ gets closer to $2\,n^2$. But we cannot find any constant C where

$$T(n) = 2\,n^2 + 10\,n + 20 = Cn^2.$$

That is, the ratio $T(n)/n^2$ is not a constant, but a function of n.

$$\frac{T(n)}{n^2} = \frac{2\,n^2 + 10\,n + 20}{n^2}$$

However, we can express an upper bound for $T(n)$. For example, for all $n \geq 100$,

$$T(n) \leq 2.102\,n^2.$$

We are now ready to make the formal definitions.

Definition: O( ) Upper Bound
Suppose there are positive constants C and n_0 such that

$$T(n) \le C \cdot f(n), \qquad \forall n \ge n_0$$

Then we say $T(n)$ is O($f(n)$).
The O( ) is read as "order of", or "big oh" of.

Example: Prove the following function is O(n^2).

$$T(n) = 5\,n^2 + 10\,n + 100$$

Proof: Intuitively, when *n* gets large, the total value of this polynomial is close to $5n^2$, because the remaining terms become negligible in comparison. Now, we formally prove that $T(n)$ is O(n^2) by finding positive constants C and n_0 such that $T(n) \le Cn^2, \forall n \ge n_0$.

$$\begin{aligned} T(n) &= 5\,n^2 + 10\,n + 100 \\ &\le 5n^2 + 10n\,(n) + 100\,(n^2), \; n \ge 1 \\ &\le 115n^2, \qquad n \ge 1 \end{aligned}$$

This satisfies the definition and proves $T(n)$ is O(n^2). (Here $C = 115$ and $n_0 = 1.$)
But, to satisfy our intuition, let us find the constant *C* closer to 5 by picking a larger n_0.
Let's arbitrarily pick $n \ge 100,$ so $\left(\frac{n}{100}\right) \ge 1$. Then,

$$\begin{aligned} T(n) &= 5\,n^2 + 10\,n + 100 \\ &\le 5n^2 + 10n\left(\frac{n}{100}\right) + 100\left(\frac{n}{100}\right)^2, \qquad n \ge 100 \\ &\le 5.11\,n^2\,, \; n \ge 100. \end{aligned}$$

Example: Prove the following polynomial is O(n^4).

$$T(n) = 5n^4 - 10n^3 + 20n^2 - 50\,n + 100$$

Proof: First we need to get rid of the negative terms.

$$\begin{aligned} T(n) &= 5n^4 - 10n^3 + 20n^2 - 50\,n + 100 \\ &\le 5n^4 + 20n^2 + 100, \qquad n \ge 0. \end{aligned}$$

Now, since we have only positive terms, we may multiply the smaller positive terms by anything ≥ 1, as we did in our earlier example. Suppose we pick $n \ge 10,$ so $\left(\frac{n}{10}\right) \ge 1$.

$$\begin{aligned} T(n) &\le 5n^4 + 20n^2 + 100 \\ &\le 5n^4 + 20\,n^2\left(\frac{n}{10}\right)^2 + 100\left(\frac{n}{10}\right)^4 \\ &\le 5.21\,n^4, \qquad n \ge 10. \end{aligned}$$

■

Next, we make the following definition for the lower bound, which is symmetrical to O( ).

Definition: $\boldsymbol{\Omega}(\,)$ Lower Bound
Suppose there are positive constants C and n_0 such that

$$T(n) \geq C \cdot f(n), \qquad \forall n \geq n_0$$

Then we say $T(n)$ is $\Omega(f(n))$.

Example: Prove the following polynomial is $\Omega(n^4)$.

$$T(n) = 5n^4 - 10n^3 + 20n^2 - 50\,n + 100$$

Proof: We must show that $T(n) \geq Cn^4, \forall n \geq n_0$ for some positive constants C, n_0. Here, we need to pick n_0 carefully so that the constant C becomes positive.

$$\begin{aligned}
&T(n) = 5n^4 - 10n^3 + 20n^2 - 50\,n + 100 \quad \text{(discard smaller positive terms)}\\
&\geq 5n^4 - 10n^3 - 50n\\
&\geq 5n^3 - 10n^3\left(\frac{n}{100}\right) - 50n\left(\frac{n}{100}\right)^3, \qquad n \geq 100\\
&\geq 4.89995\,n^4, \quad n \geq 100.
\end{aligned}$$

■

Definition: $\boldsymbol{\Theta}(\,)$ Tight Bound
Suppose there are positive constants C_1, C_2, n_0 such that

$$C_1 f(n) \leq T(n) \leq C_2 f(n), \;\; \forall n \geq n_0$$

That is, $T(n)$ is both O($f(n)$) and $\Omega\big(f(n)\big)$.
Then, we say that $T(n)$ is $\Theta(f(n))$.

Example: We proved the following polynomial is both $O(n^4)$and $\Omega(n^4)$.
Therefore, $T(n)$ is $\Theta(n^4)$.

$$T(n) = 5n^4 - 10n^3 + 20n^2 - 50\,n + 100$$

Note: For the upper bound, we proved

$$T(n) \leq 5.21\,n^4, \qquad n \geq 10.$$

And for the lower bound, we proved

$$T(n) \geq 4.89995\,n^4, \quad n \geq 100.$$

The upper bound holds for $n \geq 10$, and the lower bound holds for $n \geq 100$.
Therefore, for $n \geq 100$, they both hold.

■

Example: Suppose

$$2n \le T(n) \le 5n^2$$

In this case, $T(n)$ is $\Omega(n)$ and O(n^2). This function does not have a tight bound.

Example: Prove the following summation is $\Theta(n^2)$, without using the arithmetic sum formula, but rather by manipulating the terms to find the needed upper bound and lower bound.

$$S(n) = 1 + 2 + 3 + \cdots + n$$

1. Prove O(n^2)

$$\begin{aligned} S(n) &= 1 + 2 + \cdots + n \\ &\le n + n + \cdots + n \\ &\le n^2. \end{aligned}$$

2. Prove $\Omega(n^2)$

In the above proof for the upper bound, we raised all terms to the largest term. If we try to mimic that approach and lower all terms to the smallest term, we get $S(n) \ge 1 + 1 + \cdots + 1 = n$, which will not give the desired lower bound. Instead, we will first discard the first half of the terms, and then lower all terms to the smallest.

$$\begin{aligned} S(n) &= 1 + 2 + \cdots + n = \sum_{i=1}^{n} i \\ &\ge \sum_{i=\lceil \frac{n}{2} \rceil}^{n} i \\ &\ge \sum_{i=\lceil \frac{n}{2} \rceil}^{n} \left\lceil \frac{n}{2} \right\rceil \\ &\ge \left\lceil \frac{n}{2} \right\rceil \cdot \left\lceil \frac{n}{2} \right\rceil \ge \frac{n^2}{4} \end{aligned}$$

We proved $\frac{n^2}{4} \le S(n) \le n^2$. Therefore, $S(n)$ is $\Theta(n^2)$.

■

Sum-Rule and Product-Rule

Sum Rule for O( )
Suppose $T_1(n)$ is O($f(n)$) and $T_2(n)$ is O($g(n)$).
Then,

$$T_1(n) + T_2(n) \text{ is } \mathrm{O}\big(f(n) + g(n)\big).$$

Proof: we first express $T_1(n)$ and $T_2(n)$ in terms of the definitions of O():

$$\begin{aligned} &T_1(n) \le C_1 f(n), \quad &\forall n \ge n_1 \\ &T_2(n) \le C_2 g(n), \quad &\forall n \ge n_2 \end{aligned}$$

For $n \ge \max\{n_1, n_2\}$, the two inequalities will both hold. So,

$$\begin{aligned} &T_1(n) + T_2(n) \le C_1 f(n) + C_2 g(n), \quad &n \ge \max\{n_1, n_2\} \\ &\le Cf(n) + Cg(n), \quad &C = \max\{C_1, C_2) \\ &\le C\big(f(n) + g(n)\big). \end{aligned}$$

(**Note**: The sum rule similarly applies to Ω and Θ.)

Application of sum rule: Suppose a program has two parts: Part 1, which is executed first, and then Part 2.

Part 1 : Time $T_1(n)$ is $\mathrm{O}(n)$
Part 2: Time $T_2(n)$ is $\mathrm{O}(n^2)$

Then, the total time is $\mathrm{O}(n + n^2)$, which is $\mathrm{O}(n^2)$.

Product Rule for O( )
If $T_1(n)$ is O($f(n)$) and $T_2(n)$ is O($g(n)$),
then,

$$T_1(n) * T_2(n) \text{ is } \mathrm{O}\big(f(n) * g(n)\big).$$

Proof: Similar to the above proof; left to the student.

(**Note**: The product rule similarly applies to Ω and Θ.)

An application of the product rule is in **nested loops**, as in the following example.

Example of Analysis: A Nested Loop

Let's analyze the running time of the following program (nested loops).

```
$C = 0$
for $i = 1$ to $n+1$
    for $j = i$ to $3n-1$          Inner loop number of times = $(3n-1)-(i)+1 = 3n-i$
        $C = C+1$
```

Method 1 (Detailed Analysis): Let us find the total number of times the innermost statement ($C = C + 1$) is executed. That is, find the final value for *C.* Let $F(n)$ denote this final count. We find this count by the following double summation.

$$F(n) = \sum_{i=1}^{n+1} \sum_{j=i}^{3n-1} 1$$

The inner summation for j is $1 + 1 + \cdots + 1$, so we find the count by:

$$\text{upper limit of summation} - \text{lower limit of summation} + 1$$
$$= (3n-1) - (i) + 1 = 3n - i.$$

And the outer summation for i is arithmetic sum, so we apply the formula for it.

$$F(n) = \sum_{i=1}^{n+1} \sum_{j=i}^{3n-1} 1 = \sum_{i=1}^{n+1} (3n - i) = (num\ of\ terms) * \frac{(first + last)}{2}$$

$$= (n+1)\,\frac{(3n-1)+(3n-n-1)}{2} = (n+1)\frac{(5n-2)}{2}$$
$$= \frac{5n^2 + 3n - 2}{2}$$

The total number of times the innermost statement is executed is $O(n^2)$, which means the total running time of the program is also $O(n^2)$.

Method 2 (Loop Analysis): In this method, we don't bother to find the exact total number of times the innermost statement is executed. Rather, we analyze the loops by using the sum-rule and product-rule.

- The inner loop is executed $3n - i$ times. Since the range of i values is 1 to $n + 1$, then we know $3n - i$ is $O(n)$.
- The outer loop is executed $n + 1$ times, which is $O(n)$.
- Therefore, by the product rule, the total running time is $O(n^2)$.

Insertion Sort Algorithm

We have an array of *n* elements, $A[0{:}n-1]$. Insertion sort starts by sorting the first two elements. Then the third element is inserted into the sorted part, so that the first 3 elements are sorted. Then, the next element is inserted into the sorted portion, so that the first 4 elements become sorted, and so on. Let us first illustrate by a numerical example.

[7]	5	6	2	4	3	Sort the first two
[5	7]	6	2	4	3	Insert 6 into [5 7]
[5	6	7]	2	4	3	Insert 2 into [5 6 7]
[2	5	6	7]	4	3	Insert 4 into [2 5 6 7]
[2	4	5	6	7]	3	Insert 3 into [2 4 5 6 7]
[2	3	4	5	6	7]	

Let's look at the details of each insertion phase. For example, let us see how the last insertion is carried out. At the start of this phase, the sorted part is [2 4 5 6 7] and 3 needs to be inserted into the sorted part. This is done by a sequence of compare/swap operations between pairs, starting at the end with the pair [7 3].

2	4	5	6	[7	3] Cm/Swp
2	4	5	[6	3] Cm/Swp	7
2	4	[5	3] Cm/Swp	6	7
2	[4	3] Cm/Swp	5	6	7
[2	3] Compare	4	5	6	7
2	3	4	5	6	7

We present the pseudocode next, and then analyze both the worst-case and average-case time complexity.

```
Insertion Sort (datatype $A[\,]$, int $n$) { //Input array is $A[0{:}n-1]$
for $i = 1$ to $n-1$
    {// Insert $A[i]$.  Everything to the left of $A[i]$ is already sorted.
     $j = i$;
     while ($j > 0$ and $A[j] < A[j-1]$)
          {swap($A[j], A[j-1]$);
           $j = j-1$};
     };
}
```

Worst-Case Analysis of Insertion Sort

Method 1 (Find the worst-Case Exact Number of Operations): One way to analyze an algorithm is in terms of some dominant operation, in this case a *key comparison*. This is a comparison between a pair of elements in the array, which is highlighted in the above algorithm. By counting this operation, we are basically counting the number of times the inner loop (while loop) is executed.

Let $f(n)$ be the worst-case number of key comparisons in this algorithm. The worst-case number of key comparisons in the while loop is exactly i, because the loop starts with $j = i$ and in the worst case goes down to $j = 1$. (When $j = 0$, the comparison is skipped.) Therefore, the worst-case number of key-comparisons is:

$$f(n) = \sum_{i=1}^{n-1} i = \frac{n\,(n-1)}{2} = \frac{n^2 - n}{2}$$

(Arithmetic sum formula was used to find the sum.) We conclude that the worst-case total time of the algorithm is $O(n^2)$.

Method 2 (Analyze the Loops):

- The worst-case time of the inner while-loop is $O(i)$, thus $O(n)$.
  This is because the range of i values is 1 to $n-1$.
- The outer loop (for loop) is executed $O(n)$ times.
- By the product rule, the total worst-case time of the algorithm becomes $O(n^2)$.

Average-Case Analysis of Insertion Sort

We carry out the average case analysis in two ways, one in terms of the number of key comparisons, and the other in terms of the number of swaps.

Expected Number of Key-Comparisons: Consider the while loop, where element $A[i]$ is inserted into the sorted part. If we assume the array is random, then $A[i]$ has equal probability of being the largest, second largest, third largest, and so on. There are $i+1$ cases, as listed in the table below. The table also shows the number of key-comparisons made by the while loop for each case. Note that the last two cases both make the worst-case number of key comparisons, which is i.

Event	If element $A[i]$ is:	Number of key comparisons made by the while-loop
1	Largest	1
2	Second largest	2
3	Third Largest	3
⋮	⋮	⋮
$i-1$	Third smallest	$i-1$
i	Second smallest	i
$i+1$	Smallest	i

Since these $(i+1)$ cases all have equal probabilities, the expected number of key comparisons made by the while loop is:

$$\frac{(1+2+3+\cdots+i)+i}{i+1} = \frac{\frac{i(i+1)}{2}+i}{i+1} = \frac{i}{2}+\frac{i}{i+1}$$

(Since $\frac{i}{i+1}$ is smaller than 1, the expected number of key comparisons made by the while loop is about $i/2$, which is half of the worst-case.) Therefore, the expected number of key-comparisons for the entire algorithm is

$$F(n) = \sum_{i=1}^{n-1}\left(\frac{i}{2}+\frac{i}{i+1}\right) = \sum_{i=1}^{n-1}\frac{i}{2}+\sum_{i=1}^{n-1}\left(1-\frac{1}{i+1}\right) = \frac{n\,(n-1)}{4}+n-1-\sum_{i=1}^{n-1}\frac{1}{i+1}$$
$$= \frac{n\,(n-1)}{4}+n-\left(1+\sum_{i=1}^{n-1}\frac{1}{i+1}\right) = \frac{n\,(n-1)}{4}+n-\sum_{i=1}^{n}\frac{1}{i}$$

Recall that the latter summation is the harmonic series, $H_n = \sum_{i=1}^{n}\frac{1}{i} \cong \ln n$. So the expected number of key-comparisons for the entire algorithm is

$$F(n) \cong \frac{n\,(n-1)}{4}+n-\ln n = \frac{n^2}{4}+\frac{3n}{4}-\ln n$$

Expected Number of Swaps

In the above table, we saw the last two cases both have i comparisons. This non-uniformity resulted in a slight complication in the analysis. This complication is avoided by analyzing the expected number of swaps. Let us again look at the table of possible events.

Event	If element $A[i]$ is:	Number of SWAPS made by the while-loop
1	Largest	0
2	Second largest	1
3	Third Largest	2
$\vdots$	$\vdots$	3
$i-1$	Third smallest	$\vdots$
i	Second smallest	$i-1$
$i+1$	Smallest	i

Since all $i+1$ events have equal probability, the expected number of swaps for the while loop is

$$\frac{0+1+2+\cdots+i}{i+1}=\frac{i}{2}$$

So the expected number of swaps for the entire algorithm is

$$S(n)=\sum_{i=1}^{n-1}\frac{i}{2}=\frac{n(n-1)}{4}=\frac{n^2}{4}-\frac{n}{4}$$

Alternative Analysis for the Expected Number of Swaps:

Suppose the input sequence is a random ordering of integers 1 to *n*. This analysis considers all $n!$ possible input orderings, counts the number of swaps for each case, and averages it over all possible cases. (This analysis provides additional insight on how the expected performance is computed.)

Define a pair of elements $(A[i], A[j])$ in the array to be ***inverted*** (or ***out-of-order***) if

$$(i<j) \text{ and } A[i]>A[j].$$

And define the total number of ***inversions*** in an array as the number of inverted pairs. One way to count the number of inversions is:

$$\sum_{i=1}^{n-1} \text{Number of elements to the left of } A[i] \text{ which are greater than } A[i].$$

From this formulation, it should be obvious that the number of inversions in a sequence is exactly equal to the number of swap operations made by the insertion-sort algorithm for that input sequence. Suppose there are k elements in the original input sequence to the left of $A[i]$ with values greater than $A[i]$. Then, at the start of the while loop for inserting $A[i]$, these k elements will all be on the rightmost part of the sorted part, immediately to the left of $A[i]$, as depicted below.

$$\underbrace{(\text{Elements} < A[i]) \quad \text{followed by} \quad (k \text{ elements} > A[i])}_{Sorted\ Part} \qquad \underbrace{A[i]}_{Element\ to\ be\ inserted}$$

Then $A[i]$ has to hop over the k elements in order to get to where it needs to be in the sorted part, which means exactly k swaps.

The following table shows an example for $n = 3$. There are $n! = 6$ possible input sequences. The number of inversions for each sequence is shown, as well as the overall average number of inversions.

	Input Sequence	Number of Inversions
1	1 2 3	0
2	1 3 2	1
3	2 1 3	1
4	2 3 1	2
5	3 1 2	2
6	3 2 1	3
		Overall Average $= 9/6 = 1.5$

The input sequences may be partitioned into pairs, such that each pair of sequences are reverse of each other. For example, the reverse of [1 3 2] is the sequence [2 3 1]. The following table shows the pairing of the 6 input sequences.

	Pairs of input sequences which are reverse of each other	Number of inversions	Average number of inversions for each pair
	1 2 3 3 2 1	0 3	1.5
	1 3 2 2 3 1	1 2	1.5
	2 1 3 3 1 2	1 2	1.5
			Overall Average: 1.5

In general, if a pair of elements is inverted in one sequence, then the pair is not inverted in the reverse sequence, and vice versa. For example:

- In the sequence [1 3 2], the pair $(3, 2)$ is inverted
- In the reverse sequence, [2 3 1], the pair $(2, 3)$ is not inverted.

This means that each pair of values is inverted in only one of the two sequences. So each pair contributes 1 to the sum of inversions in the two sequences. Therefore, the sum of inversions in each pair of reverse sequences is the number of pairs in a sequence of n elements, which is $\frac{n(n-1)}{2}$. So, the average number of inversions for each pair of sequences is

$$S(n) = \frac{n(n-1)}{4}$$

The overall average number of inversions is also $S(n)$. Therefore, the expected number of swaps in the algorithm is this exact number.

Deriving the Expected Number of Comparisons from the Number of Swaps

Earlier, we derived the expected number of comparisons. As an alternative approach, we now show how we may derive the same results rather quickly from the expected number of swaps. Every key-comparison made by the while loop is followed by a swap, except possibly the last comparison before the termination of the while loop. That is, if the while loop terminates with $(j > 0 \text{ and } A[j] \geq A[j-1])$, then this last comparison is not followed by a swap. And this happens when the element being inserted is not the smallest in its prefix sequence, which has the probability $i/(i+1)$, as indicted earlier. So, with this probability, there is a last comparison in the while loop which is not followed by a swap. Therefore, the expected number of comparisons equals the expected number of swaps, $S(n)$, plus $\sum_{i=1}^{n-1} \frac{i}{i+1}$.

$$F(n) = S(n) + \sum_{i=1}^{n-1} \frac{i}{i+1} = \frac{n(n-1)}{4} + \sum_{i=1}^{n-1} (1 - \frac{1}{i+1})$$

$$= \frac{n(n-1)}{4} + n - \sum_{i=1}^{n} \frac{1}{i} = \frac{n(n-1)}{4} + n - H_n \cong \frac{n(n-1)}{4} + n - \ln n$$

(This is the same result which we derived earlier by a different method.)

Module 7: Recursive Algorithms, Recurrence Equations, and Divide-and-Conquer Technique

Reading from the Textbook: Chapter 4 Algorithms

Introduction

In this module, we study recursive algorithms and some related concepts. We show how recursion ties in with induction. That is, the correctness of a recursive algorithm is proved by induction. We show how recurrence equations are used to analyze the time complexity of algorithms. Finally, we study a special form of recursive algorithms based on the divide-and-conquer technique.

Contents

Simple Examples of Recursive Algorithms

Factorial: Consider the factorial definition

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

This formula may be expressed recursively as

$$n! = \begin{cases} n \times (n-1)!\,, & n > 1 \\ 1, & n = 1 \end{cases}$$

Below is a recursive program to compute $n!$.

```
int  Factorial (int $n$) {
if ($n == 1$) return 1;
else {
  Temp = Factorial ($n - 1$);
  return ($n$ * Temp);
  }
}
```

The input parameter of this function is n, and the return value is type integer. When $n = 1$, the function returns 1. When $n > 1$, the function calls itself (called a recursive call) to compute $(n-1)!$. It then multiplies the result by n, which becomes $n!$.

The above program is written in a very basic way for clarity, separating the recursive call from the subsequent multiplication. These two steps may be combined, as follows.

```
int  Factorial (int $n$) {
if ($n == 1$) return 1;
else return (n* Factorial ($n - 1$));
 }
```

Correctness Proof: The correctness of this recursive program may be proved by induction.

- Induction Base: From line 1, we see that the function works correctly for $n = 1$.
- Hypothesis: Suppose the function works correctly when it is called with $n = m$, for some $m \geq 1$.
- Induction step: Then, let us prove that it also works when it is called with $n = m + 1$. By the hypothesis, we know the recursive call works correctly for $n = m$ and computes $m!$. Subsequently, it is multiplied by $n = m + 1$, thus computes $(m+1)!$. And this is the value correctly returned by the program.

Finding Maximum Element of an Array

As another simple example, let us write a recursive program to compute the maximum element in an array of *n* elements, $A[0{:}n-1]$. The problem is broken down as follows.

To compute the Max of n elements for $n > 1$,

- Compute the Max of the first $n-1$ elements.
- Compare with the last element to find the Max of the entire array.

Below is the recursive program (pseudocode). It is assumed that the array type is dtype, declared earlier.

```
dtype  Max (dtype $A[\,]$, int $n$) {
if ($n == 1$) return $A[0]$;
else{
  $T = \mathrm{Max}(A, n-1)$;     //Recursive call to find max of the first $n-1$ elements
  If ($T < A[n-1]$)       //Compare with the last element
      return $A[n-1]$;
  else return $T$;
  }
}
```

Computing Sum of Elements in an Array

Below is a recursive program for computing the sum of elements in an array $A[0{:}n-1]$.

$$S = \sum_{i=0}^{n-1} A[i]$$

```
dtype Sum (dtype $A[\,]$, int $n$) {
if ($n == 1$) return $A[0]$;
else{
  $S$ = Sum($A, n-1$);   //Recursive call to compute the sum of the first $n-1$ elements
  $S = S + A[n-1]$;    //Add the last element
  return ($S$)
  }
}
```

The above simple problems could be easily solved without recursion. They were presented recursively only for pedagogical purposes. The next example problem, however, truly needs the power of recursion. It would be very difficult to solve the problem without recursion.

Towers of Hanoi Problem

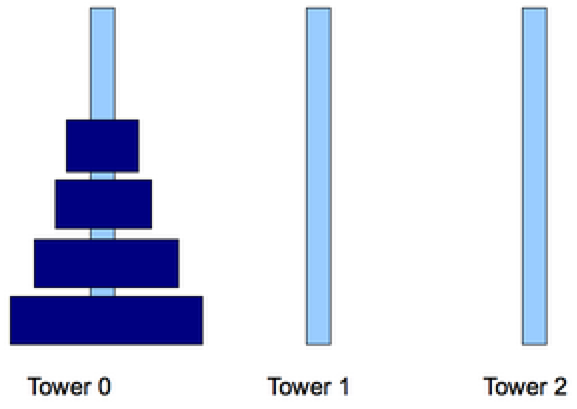
This is a toy problem that is easily solved recursively. There are three towers (posts) $A, B,$ and C. Initially, there are *n* disks of varying sizes stacked on tower *A,* ordered by their size, with the largest disk in the bottom and the smallest one on top. The object of the game is to have all *n* discs stacked on tower *B* in the same order, with the largest one in the bottom. The third tower is used for temporary storage. There are two rules:

- Only one disk may be moved at a time in a restricted manner, from the top of one tower to the top of another tower. If we think of each tower as a stack, this means the moves are restricted to a *pop* from one stack and *push* onto another stack.
- A larger disk must never be placed on top of a smaller disk.

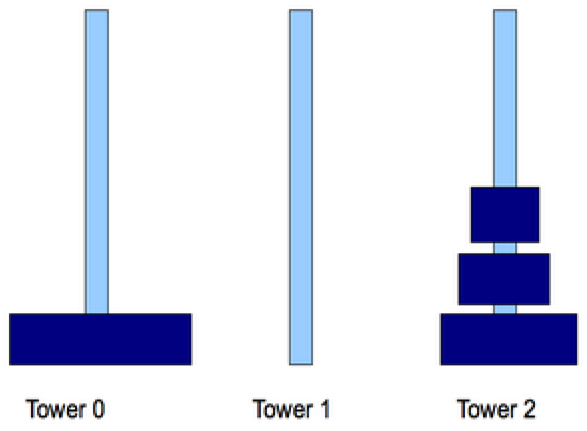
The recursive algorithm for moving *n* disks from tower *A* to tower *B* works as follows. If $n = 1$, one disk is moved from tower *A* to tower *B.* If $n > 1$,

1 Recursively move the top $n - 1$ disks from $A\ to\ C$. The largest disk remains on tower A by itself.
2 Move a single disk from $A\ to\ B$.
3 Recursively move back $n - 1$ disks from $C\ to\ B$.

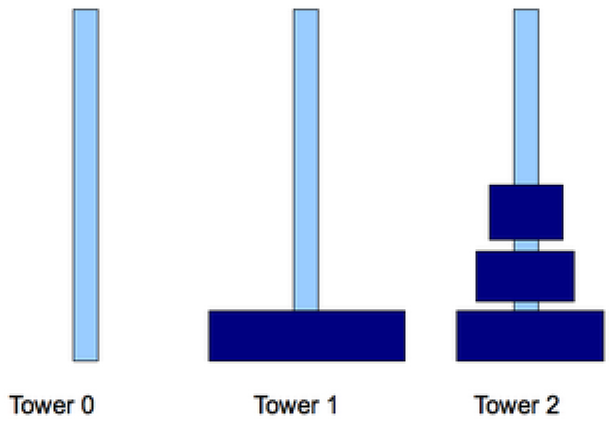
An illustration is shown below for $n = 4$.



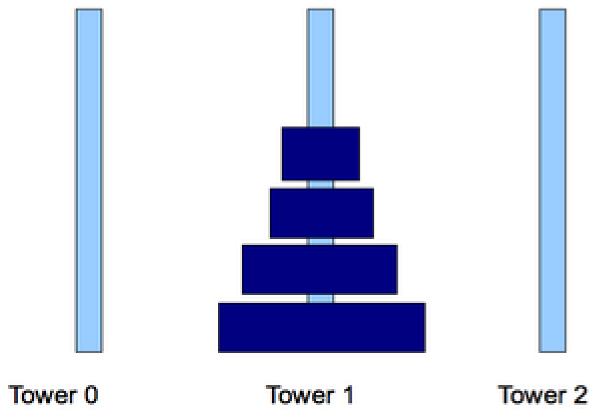
(a) Initial configuration with 4 disks on Tower 0



(b) After recursively moving the top 3 disks from Tower 0 to Tower 2



(c) After moving the bottom disk from Tower 0 to Tower 1



(d) After recursively moving back 3 disks from Tower 2 to Tower 1.

Below is the recursive algorithm. The call Towers (A, B, C, n) moves *n* disks from tower A to B, using C as temporary storage.

```
Towers $(A, B, C, n)$ {
1   if $(n == 1)$ {
2       MoveOne $(A, B)$;
3       return};
4   Towers $(A, C, B, n - 1)$;
5   MoveOne $(A, B)$;
6   Towers $(C, B, A, n - 1)$;
  }
```

Proof of Correctness

The correctness of the algorithm is proved by induction. For $n = 1$, a single move is made from *A* to *B*. So the algorithm works correctly for $n = 1$. To prove the correctness for any $n \geq 2$, suppose the algorithm works correctly for $n - 1$. Then, by the hypothesis, the recursive call of line 4 works correctly and moves the top $n - 1$ disks to *C*, leaving the bottom disk on tower *A.* The next step, line 5, moves the bottom disk to *B.* Finally, the recursive call of line 6 works correctly by the hypothesis and moves back $n - 1$ disks from *C* to *B.* Thus, the entire algorithm works correctly for n.

An Improvement in Algorithm Style

The above algorithm has a single move appearing twice in the code, once for $n = 1$ and once for $n > 1$. This repetition may be avoided by making $n = 0$ as the termination criteria for recursion.

```
Towers $(A, B, C, n)$ {
if $(n == 0)$ return;
Towers $(A, C, B, n - 1)$;
MoveOne $(A, B)$;
Towers $(C, B, A, n - 1)$;
}
```

Recurrence Equation to Analyze Time Complexity

Let us analyze the time complexity of the algorithm. Let

$$f(n) = \text{number of single moves to solve the problem for } n \text{ disks.}$$

Then, the number of moves for each of the recursive calls is $f(n-1)$. So, we set up a recurrence equation for $f(n)$.

$$f(n) = \begin{cases} 1, & n = 1 \\ 2f(n-1)+1, & n \geq 2 \end{cases}$$

We need to solve this recurrence equation to find $f(n)$ directly in terms of n.

Method 1: Repeated Substitution

$$f(n) = 1 + 2 \cdot \underbrace{f(n-1)}_{1+2f(n-2)}$$
$$f(n) = 1 + 2 + 4 \cdot \underbrace{f(n-2)}_{1+2f(n-3)}$$
$$f(n) = 1 + 2 + 4 + 8 \cdot f(n-3)$$
$$f(n) = 1 + 2 + 2^2 + 2^3 \cdot f(n-3)$$

After a few substitutions, we observe the general pattern, and see what is needed to get to the point where the last term becomes $f(1)$.

$$\vdots$$
$$f(n) = 1 + 2 + 2^2 + 2^3 + \cdots + 2^{n-1} \cdot f(1)$$

Then we can use the base case of the recurrence equation, $f(1) = 1$.

$$f(n) = 1 + 2 + 2^2 + 2^3 + \cdots + 2^{n-1}$$

We use geometric sum formula for this summation (where each term equals the previous term times a constant).

$$f(n) = \frac{2^n - 1}{2 - 1}$$

$$f(n) = 2^n - 1.$$

The repeated substitution method may not always be successful. Below is an alternative method.

Method 2: Guess the solution and prove it correct by induction

Suppose we guess the solution to be exponential, but with some constants to be determined.

***Guess**:* $f(n) = A\,2^n + B$

We try to prove the solution form is correct by induction. If the induction is successful, then we find the values of the constant *A* and *B* in the process.

Induction Proof:

Induction Base, $n = 1$:

$$\begin{aligned} &f(1) = 1 && \text{(from the recurrence)} \\ &f(1) = 2A + B && \text{(from the solution form)} \end{aligned}$$

So we need

$$\boxed{2A + B = 1}$$

Induction Step: Suppose the solution is correct for some $n \geq 1$:

$$f(n) = A\,2^n + B \qquad \text{(hypothesis)}$$

Then we must prove the solution is also correct for $n + 1$:

$$f(n+1) = A\,2^{n+1} + B \qquad \text{(Conclusion)}$$

To prove the conclusion, we start with the recurrence equation for $n + 1$, and apply the hypothesis:

$$\begin{aligned} f(n+1) &= 2f(n) + 1 && \text{(from the recurrence equation)} \\ &= 2[A\,2^n + B] + 1 && \text{(Substitute hypothesis)} \\ &= A\,2^{n+1} + (2B + 1) \\ &= A\,2^{n+1} + B && \text{(equate to conclusion)} \end{aligned}$$

To make the latter equality, we equate term-by-term.

$$\boxed{2B + 1 = B}$$

So we have two equations for *A* and *B.*

$$\begin{cases} 2A + B = 1 \\ 2B + 1 = B \end{cases}$$

We get $B = -1$ and $A = 1$. So we proved that

$$f(n) = A\,2^n + B$$
$$f(n) = 2^n - 1.$$

Computing Power by Repeated Multiplications

Given a real number X and an integer n, let us see how to compute X^n by repeated multiplications. The following simple loop computes the power, but is very inefficient because the power goes up by 1 by each multiplication.

```
$T = X$
for $i = 2$ to $n$
    $T = T * X$
```

The algorithm is made much more efficient by repeated squaring, thus doubling the power by each multiplication. First consider the **special case** when $n = 2^k$ for some integer $k \geq 1$.

```
$T = X$
for $i = 1$ to $k$
    $T = T * T$
```

Now, let us see how to generalize this algorithm for any integer *n*. First consider a numerical example. Suppose we want to compute X^{13}, where $13 = 1101$ (binary). Since $13 = 8 + 4 + 1$, we first apply repeated squaring to get X^2, X^4, X^8. Then multiply together (X, X^4, X^8) to get X^{13}. Informally, the computation may be done as follows.

- Square: $X^2 = X * X$
- Square: $X^4 = X^2 * X^2$
- Square: $X^8 = X^4 * X^4$
- Multiply together ($X^1 * X^4 * X^8$) to get X^{13}. (This takes 2 basic multiplications.)

This algorithm may be formalized non-recursively, with some effort. But a recursive implementation makes the algorithm much simpler and more eloquent.

```
real Power (real $X$, int $n$) { // It is assumed that $n > 0$.
if ($n == 1$)   return $X$;
$T = \text{Power}\left(X, \lfloor n/2 \rfloor\right)$;
$T = T * T$;
If ($n \bmod 2 == 1$)
   $T = T * X$;
return $T$ }
```

Let $n = 2m + r$, where $r \in \{0,1\}$. The algorithm first makes a recursive call to compute $T = X^m$. Then it squares T to get $T = X^{2m}$. If $r = 0$, this is returned. Otherwise, when $r = 1$, the algorithm multiplies T by X, to result in $T = X^{2m+1}$.

Analysis

Let $f(n)$ be the worst-case number of multiplication steps to compute X^n. The number of multiplications made by the recursive call is $f(\lfloor n/2 \rfloor)$. The recursive call is followed by one more multiplication. And in the worst-case, if n is odd, one additional multiplication is performed at the end. Therefore,

$$f(n) = \begin{cases} 0, & n = 1 \\ f(\lfloor n/2 \rfloor) + 2, & n \geq 2 \end{cases}$$

Let us prove by induction that the solution is as follows, where the log is in base 2.

$$f(n) = 2\lfloor \log n \rfloor$$

Induction Base, $n = 1$: From the recurrence, $f(1) = 0$. And the claimed solution is $f(1) = 2\lfloor \log 1 \rfloor = 0$. So the base is correct.

Induction step: Integer n may be expressed as follows, for some integer k.

$$2^k \leq n < 2^{k+1}$$

This means $\lfloor \log n \rfloor = k$. And,

$$2^{k-1} \leq \lfloor n/2 \rfloor < 2^k$$

Thus, $\lfloor \log \lfloor n/2 \rfloor \rfloor = k - 1$. To prove the claimed solution for any $n \geq 2$, suppose the solution is correct for all smaller values. That is,

$$f(m) = 2\lfloor \log m \rfloor, \quad \forall\, m < n$$

In particular, for $m = \lfloor n/2 \rfloor$,

$$f(\lfloor n/2 \rfloor) = 2\lfloor \log \lfloor n/2 \rfloor \rfloor = 2(k-1) = 2k - 2$$

Then,

$$f(n) = f(\lfloor n/2 \rfloor) + 2 = 2k - 2 + 2 = 2k = 2\lfloor \log n \rfloor$$

This completes the induction proof.

Misuse of Recursion

Consider again the problem of computing the power X^n by repeated multiplication. We saw an efficient recursive algorithm to compute the power with $(2\log n)$ multiplications. Now, suppose a naive student writes the following recursive algorithm.

```
real Power (real $X$, int $n$) { // It is assumed that $n > 0$.
if $(n == 1)$   return $X$;
return (Power$(X, \lfloor n/2 \rfloor)$ * Power$(X, \lceil n/2 \rceil)$);
}
```

Although this program correctly computes the power, and it appears eloquent and clever, it is very inefficient. The reason for the inefficiency is that it performs a lot of repeated computations. The first recursive call computes $X^{\lfloor n/2 \rfloor}$ and the second recursive call computes $X^{\lceil n/2 \rceil}$, but there is a lot of overlap computations between the two recursive calls. Let us analyze the number of multiplications, $f(n)$, for this algorithm. Below is the recurrence.

$$f(n) = \begin{cases} 0, & n = 1 \\ f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) + 1, & n \geq 2 \end{cases}$$

The solution is $f(n) = n - 1$. (This may be easily proved by induction.) This shows the terrible inefficiency introduced by the overlapping recursive calls.

As another example, consider the Fibonacci sequence, defined as

$$F_n = \begin{cases} 1, & n = 1 \\ 1, & n = 2 \\ F_{n-1} + F_{n-2}, & n \geq 3 \end{cases}$$

It is easy to compute F_n with a simple loop in $O(n)$ time. But suppose a naïve student, overexcited about recursion, implements the following recursive program to do the job.

```
int Fib (int $n$) {
if $(n \leq 2)$ return (1);
return (Fib$(n-1)$ + Fib$(n-2)$)
```

This program makes recursive calls with a great deal of overlapping computations, causing a huge inefficiency. Let us verify that time complexity becomes exponential! Let $T(n)$ be the total number of addition steps by this algorithm for computing F_n.

$$T(n) = \begin{cases} 0, & n = 1 \\ 0, & n = 2 \\ T(n-1) + T(n-2) + 1, & n \geq 3 \end{cases}$$

We leave it as an exercise for the student to prove by induction that the solution is

$$T(n) \geq (1.618)^{n-3}, \ \ n \geq 3$$

Note that an exponential function has an extremely large growth rate. For example, for $n = 50, \ T(n) > 6.6 * 10^9$.

Recursive Insertion-Sort

An informal recursive description of insertion sort is as follows.

To sort an array of *n* elements, where $n \geq 2,$ do:

1. Sort the first $n - 1$ elements recursively.
2. Insert the last element into the sorted part. (We do this with a simple loop, without recursion. This loop is basically the same as what we had for the non-recursive version of insertion-sort earlier.)

Below is a formal pseudocode.

```
ISort (dtype $A[\,], \text{int } n$) {
if ($n == 1$) return;
ISort ($A, n-1$);
$j = n - 1;$
while ($j > 0$ and $A[j] < A[j-1]$) {
    SWAP($A[j], A[j-1]$);
    $j = j - 1;$
    }
}
```

Note: Our purpose in presenting recursive insertion-sort is to promote **recursive thinking**, as it simplifies the formulation of algorithms. However, for actual implementation, recursive insertion sort is not recommended. The algorithm makes a long chain of recursive calls before any return is made. (This long chain is called **depth of recursion.)** And the long chain of recursive calls may easily cause stack-overflow at run-time when *n* is large.

Time Complexity Analysis

Let $f(n)$ be the worst-case number of key-comparisons to sort *n* elements. As we discussed for the non-recursive implementation, we know the while loop in the worst-case makes $(n-1)$ key-comparisons. So,

$$f(n) = \begin{cases} 0, & n = 1 \\ f(n-1) + n - 1, & n \geq 2 \end{cases}$$

Method 1: Solution by repeated substitution

$$\begin{aligned}
& f(n) = (n-1) + f(n-1) \\
& = (n-1) + (n-2) + f(n-2) \\
& = (n-1) + (n-2) + (n-3) + f(n-3) \\
& \vdots \\
& = (n-1) + (n-2) + (n-3) + \cdots + 1 + \underbrace{f(1)}_{=0} \\
& = (n-1) + (n-2) + (n-3) + \cdots + 1 \qquad \text{(Apply arithmetic sum formula)} \\
& = \frac{(n-1)n}{2} \\
& = \frac{n^2 - n}{2}
\end{aligned}$$

Method 2: Guess the solution and prove correctness by induction

Suppose we guess the solution as $O(n^2)$ and express the solution as below, in terms of some constants A, B, C to be determined.

$$f(n) = An^2 + Bn + C$$

Proof by induction:

Base, $n = 1$:

$$\begin{aligned}
& f(1) = 0 && \text{(from the recurrence)} \\
& = A + B + C && \text{(from the solution form)}
\end{aligned}$$

So we need $A + B + C = 0$.

Next, to prove the solution is correct for any $n \geq 2$, suppose the solution is correct for $n - 1$. That is, suppose

$$\begin{aligned}
& f(n-1) = A(n-1)^2 + B(n-1) + C \\
& = A(n^2 - 2n + 1) + B(n-1) + C
\end{aligned}$$

Then,

$f(n) = f(n-1) + (n-1)$ from the recurrence equation
$= A(n^2 - 2n + 1) + B(n-1) + C + (n-1)$ Use hypothesis to replace for f(n-1)
$= A\,n^2 + (-2A + B + 1)\,n + (A - B + C - 1)$
$= An^2 + Bn + C$

To make the latter equality, we equate term-by-term. That is, equate the n^2 terms, the linear terms, and the constants. So,

$$-2A + B + 1 = B$$
$$A - B + C - 1 = C$$

We have three equations to solve for A, B, C.

$-2A + B + 1 = B \quad \rightarrow \quad A = 1/2$
$A - B + C - 1 = C \quad \rightarrow \quad B = A - 1 = -1/2$
$A + B + C = 0 \quad \rightarrow \quad C = 0$

Therefore, $f(n) = \frac{n^2}{2} - \frac{n}{2}$.

Alternative Guess:

Suppose we guess the solution as $O(n^2)$ and use the definition of O( ) to express the solution with an upper bound:

$$f(n) \leq A\,n^2$$

We need to prove by induction that this solution works, and in the process determine the value of the constant *A*.

Induction base, $n = 1$:

$$f(1) = 0 \leq A \cdot 1^2$$

Therefore,

$$\boxed{A \geq 0}$$

Next, to prove the solution is correct for any $n \geq 2$, suppose the solution is correct for $n - 1$. That is, suppose

$$f(n-1) \leq A\,(n-1)^2$$

Then,

$$
\begin{aligned}
f(n) &= f(n-1) + n - 1 \\
&\le A(n-1)^2 + n - 1 \\
&\le A(n^2 - 2n + 1) + n - 1 \\
&\le An^2 + (-2A+1)n + (A-1) \\
&\le An^2
\end{aligned}
$$

To satisfy the latter inequality, we need to make the linear term ≤ 0, and the constant term ≤ 0.

$-2A + 1 \le 0 \quad \rightarrow \quad \boxed{A \ge 1/2}$

$A - 1 \le 0 \quad \rightarrow \quad \boxed{A \le 1}$

The three (boxed) inequalities on *A* are all satisfied by $\frac{1}{2} \le A \le 1$. Any value of *A* in this range satisfies the induction proof. We may pick the smallest value, $A = \frac{1}{2}$. Therefore, we have proved $f(n) \le n^2/2$.

Divide-and-Conquer Algorithms

The *divide-and-conquer* strategy divides a problem of a given size into one or more subproblems of the same type but smaller size. Then, supposing that the smaller size subproblems are solved recursively, the strategy is to try to obtain the solution to the original problem. We start by a simple example.

Finding MAX by Divide-and-Conquer

The algorithm divides an array of n elements, $A[0{:}n-1]$, into two halves, finds the max of each half, then makes one comparison between the two maxes to find the max of the entire array.

```
dtype FindMax (dtype $A[\,]$, $\text{int}\, S$, $\text{int}\, n$)
{ // $S$ is the starting index in the array, and $n$ is the number of elements
if ($n == 1$) return $A[S]$;
$T_1 = \text{FindMax}\left(A, S, \lfloor n/2 \rfloor\right)$;                  //Find max of the first half
$T_2 = \text{FindMax}\left(A, S + \lfloor n/2 \rfloor, n - \lfloor n/2 \rfloor\right)$;   //Find max of the second half
if ($T_1 \ge T_2$)                                    // Comparison between the two maxes
    return $T_1$
else return $T_2$; }
```

Analysis (Special case when $n = 2^k$)

Let $f(n)$ be the number of key-comparisons to find the max of an array of n elements. Initially, to simplify the analysis, we assume that $n = 2^k$ for some integer k. In this case, the size of each half is exactly $n/2$, and the number of comparisons to find the max of each half is $f(n/2)$.

$$f(n) = \begin{cases} 0, & n = 1 \\ 2f(n/2) + 1, & n \geq 2 \end{cases}$$

Solution by Repeated Substitution

$$\begin{aligned}
f(n) &= 1 + 2\,f(n/2) \\
&= 1 + 2[1 + 2f(n/4)] = 1 + 2 + 4f(n/4) \\
&= 1 + 2 + 4 + 8\,f(n/8) \\
&\vdots \\
&= 1 + 2 + 4 + \cdots + 2^{k-1} + 2^k \underbrace{f\left(n/2^k\right)}_{f(1)=0} \\
&= 1 + 2 + 4 + \cdots + 2^{k-1} \qquad \text{(Use Geometric Sum formula)} \\
&= \frac{2^k - 1}{2 - 1} = 2^k - 1 \\
&= n - 1.
\end{aligned}$$

So the number of key-comparisons is the same as when we did this problem by a simple loop.

Analysis for general *n*

The recurrence equation for the general case becomes:

$$f(n) = \begin{cases} 0, & n = 1 \\ f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) + 1, & n \geq 2 \end{cases}$$

It is easy to prove by induction that the solution is still $f(n) = n - 1$.
(We leave the induction proof to the student.)

Binary Search Algorithm

The sequential search algorithm works on an unsorted array and runs in $O(n)$ time. But if the array is sorted, the search may be done more efficiently, in $O(\log n)$ time, by a divide-and-conquer algorithm known as binary-search.

Given a **sorted** array $A[0\colon n-1]$ and a search key, the algorithm starts by comparing the search key against the **middle** element of the array, $A[m]$.

- If $KEY = A[m]$, then return m
- If $KEY < A[m]$, then recursively search the left half of the array.
- If $KEY > A[m]$, then recursively search the right half of the array.

So after one comparison, if the key is not found, then the size of the search is reduced to about $n/2$. After two comparisons, the size is reduced to about $n/4$, and so on. So in the worst-case, the algorithm makes about $\log n$ comparisons. Now, let us write the pseudocode and analyze it more carefully.

```
int BS (dtype $A[\,]$, int $Left$, int $Right$, dtype $KEY$) {
  // $Left$ is the starting index, and $Right$ is the ending index of the part to search.
  // If not found, the algorithm returns -1.
1   if ($Left > Right$) return ($-1$);      // not found
2   $m = \left\lfloor (Left + Right)/2 \right\rfloor$    // Index of the middle element
3   if ($KEY == A[m]$) return ($m$);
4   else if ($KEY < A[m]$)
5         return ($BS(A, Left, m-1, KEY)$);
6   else return ($BS(A, m+1, Right, KEY)$);
```

Let $n = Right - Left + 1 =$ Number of elements remaining in the search.
Let $f(n) =$ Worst-case number of key-comparisons to search an array of *n* elements.

Analysis (Special case when $n = 2^k$)

In lines 3 and 4 of the algorithm, it appears that there are 2 key comparisons before the recursive call. However, it is reasonable to count these as a single comparison, for the following reasoning:

- The comparisons are between the same pair of elements $(KEY, A[m])$.
- Computers normally have machine-level instructions where a single comparison is made, followed by conditional actions.

- It is also reasonable to imagine a high-level-language construct similar to a "case statement", where a single comparison is made, followed by several conditional cases.

We are now ready to formulate a recurrence equation for $f(n)$. Note that for the special case when $n = 2^k$, the maximum size of the recursive call is exactly $n/2$.

$$f(n) = \begin{cases} 1, & n = 1 \\ 1 + f\left(\frac{n}{2}\right), & n \geq 2 \end{cases}$$

Solution by repeated substitution:

$f(n) = 1 + f\left(n/2\right)$
$= 1 + 1 + f\left(n/4\right)$
$= 1 + 1 + 1 + f\left(n/8\right)$
$= 4 + f\left(n/2^4\right)$
$\vdots$
$= k + f\left(\frac{n}{2^k}\right)$
$= k + f(1)$
$= k + 1$
$= \log n + 1.$

Analysis of Binary Search for general *n*

For the general case, the size of the recursive call is at most $\lfloor n/2 \rfloor$. So,

$$f(n) = \begin{cases} 1, & n = 1 \\ 1 + f(\lfloor n/2 \rfloor), & n \geq 2 \end{cases}$$

We will prove by induction that the solution is

$$f(n) = \lfloor \log n \rfloor + 1$$

(The induction proof is almost identical to our earlier proof for Power.)

Induction Base, $n = 1$: From the recurrence, $f(1) = 1$. And the claimed solution is $f(1) = \lfloor \log 1 \rfloor + 1 = 1$. So the base is correct.

Induction step: Any integer n may be expressed as follows, for some integer k.

$$2^k \leq n < 2^{k+1}$$

This means $\lfloor \log n \rfloor = k$. And,

$$2^{k-1} \le \lfloor n/2 \rfloor < 2^k$$

Thus, $\lfloor \log \lfloor n/2 \rfloor \rfloor = k-1$. To prove the claimed solution for any $n \ge 2,$ suppose the solution is correct for all smaller values. That is,

$$f(m) = \lfloor \log m \rfloor + 1, \quad \forall\, m < n$$

In particular, for $m = \lfloor n/2 \rfloor$,

$$f(\lfloor n/2 \rfloor) = \lfloor \log \lfloor n/2 \rfloor \rfloor + 1 = (k-1) + 1 = k = \lfloor \log n \rfloor$$

Then,

$$f(n) = f(\lfloor n/2 \rfloor) + 1 = k + 1 = \lfloor \log n \rfloor + 1.$$

This completes the induction proof.

Mergesort

The insertion-sort algorithm discussed earlier has a basic *incremental* approach. Each iteration of the algorithm inserts one more element into the sorted part. This algorithm has time complexity $O(n^2)$. The Mergesort algorithm uses a divide-and-conquer strategy and runs in $O(n \log n)$ time.

Let us first review the easier problem of merging two sorted sequences. We consider a numerical example. Suppose we have two sorted sequences A and B, and we want to merge them into a sorted sequence C.

$$\begin{aligned} &A: \quad 4,5,8,10,12,15 \\ &B: \quad 2,3,9,10,11 \\ &C: \end{aligned}$$

We first compare the smallest (first) element of A, with the smallest (first) element of B. The smaller of the two is obviously the smallest element and becomes the first element in the sorted result, C.

$$\begin{aligned} &A: \quad 4,5,8,10,12,15 \\ &B: \quad 3,9,10,11 \\ &C: \quad 2 \end{aligned}$$

Now, one of the two sorted sequences (in this case, B) has one less element. And the merge process is continued the same way.

$$\begin{aligned} A&: \quad 4,5,8,10,12,15 \\ B&: \quad 9,10,11 \\ C&: \quad 2,3 \end{aligned}$$

The merge process is continued until one of the two sequences has no more elements in it, and the other sequence has one or more elements remaining.

$$\begin{aligned} A&: \quad 12,15 \\ B&: \\ C&: \quad 2,3,4,5,8,9,10,10,11 \end{aligned}$$

At this point, the remaining elements are appended at the end of the sorted result without any further comparisons.

$$\begin{aligned} A&: \\ B&: \\ C&: \quad 2,3,4,5,8,9,10,10,11,12,15 \end{aligned}$$

Let $M(m,n)$ be the worst-case number of key comparisons to merge two sorted sequences of length m and n. Then,

$$\boxed{M(m,n) = m + n - 1}$$

The reasoning is simple. With each comparison, one element is copied into the sorted result. So, after at most $m + n - 1$ comparisons, only one element will remain in one of the sorted sequences, which requires no further comparison.

What is the best-case number of key-comparisons? It is $\min(m,n)$. The best-case happens if all elements of the shorter sequence are smaller than all elements of the longer sequence.

A special case of the merge problem is when the two sorted sequences are of equal length. In this case, the worst-case number of comparisons is

$$M\left(\frac{n}{2},\frac{n}{2}\right) = n - 1.$$

The total time of the merge is $O(n)$, which mean $\le Cn$ for some constant C.

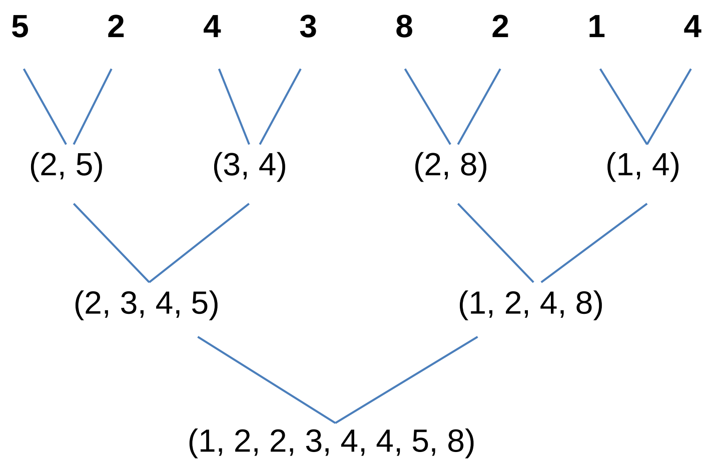
Below is the pseudocode for merging two sorted sequences $A[1{:}m]$ and $B[1{:}n]$ into the sorted result $C[1{:}m+n]$.

```
Merge (dtype $A[\,]$, int $m$, dtype $B[\,]$, int $n$, dtype $C[\,]$) {
// Inputs are sorted arrays $A[1{:}m]$ and $B[1{:}n]$. Output is sorted result $C[1{:}m+n]$.
$i = 1$;   //Index into array A
$j = 1$;   //Index into array B
$k = 1$;   //Index into array C
while ($m \ge i$ and $n \ge j$){
    if ($A[i] \le B[j]$)
        { $C[k] = A[i]$;   $i = i + 1$};
    else
        {$C[k] = B[j]$;   $j = j + 1$};
    $k = k + 1$;
    }
while ($m \ge i$)     //Empty remaining of array A
        { $C[k] = A[i]$;   $i = i + 1$; $k = k + 1$ };
while ($n \ge j$)     //Empty remaining of array B
        {$C[k] = B[j]$;   $j = j + 1$;  $k = k + 1$};
}
```

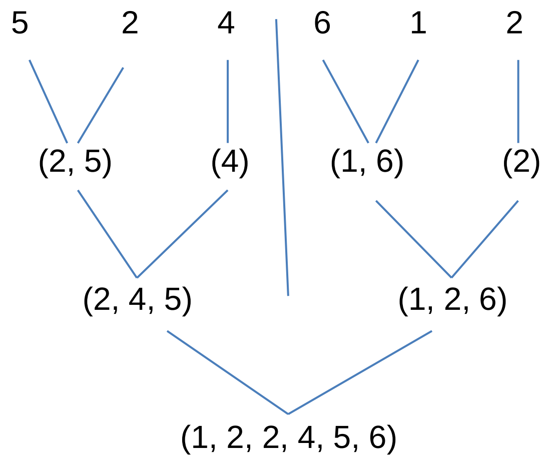
We are now ready to discuss the Mergesort algorithm, which uses a divide-and-conquer technique, and sorts a random array of n elements from scratch.

Mergesort: To sort n elements, when $n \ge 2$, do:
- Divide the array into two halves;
- Sort each half recursively;
- Merge the two sorted halves.

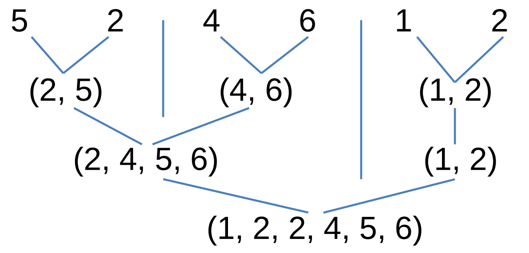
Let us consider a numerical example of Mergesort for $n = 2^3 = 8$. To sort 8 elements, they are divided into two halves, each of size 4. Then each 4 elements are divided into two halves, each of size 2. So at the bottom level, each pair of 2 is merged. At the next level, two sorted sequences of length 2 are merged into sorted sequences of length 4. At the next level, two sorted sequences of length 4 are merged into a sorted 8.



Next, consider an example of Mergesort for $n = 6$. The recursive Mergesort divides the array into two halves, each of size 3. To sort each 3, they are divided into 2 and 1.



The Mergesort algorithm may also be implemented non-recursively. At the bottom level, each pairs of 2 are sorted. Then, pairs of length 2 are merged to get sorted sequences of length 4, and so on. Below is the non-recursive implementation for the last example.



Analysis of Mergesort (Special case when $n = 2^k$)

Let $T(n)$ be the total worst-case time to sort n elements (by recursive Mergesort). The worst-case time to recursively sort each half is $T(n/2)$. And the time to merge the two sorted halves is $O(n)$, which means $\le cn$ for some constant c. Therefore,

$$T(n) \le \begin{cases} 2\,T\left(\frac{n}{2}\right) + c\,n, & n \ge 2 \\ d, & n = 1 \end{cases}$$

Solution by repeated substitution:

$T(n) \le cn + 2\,T\left(\frac{n}{2}\right)$

$\le cn + 2(c\frac{n}{2} + 2\,T\left(\frac{n}{4}\right))$

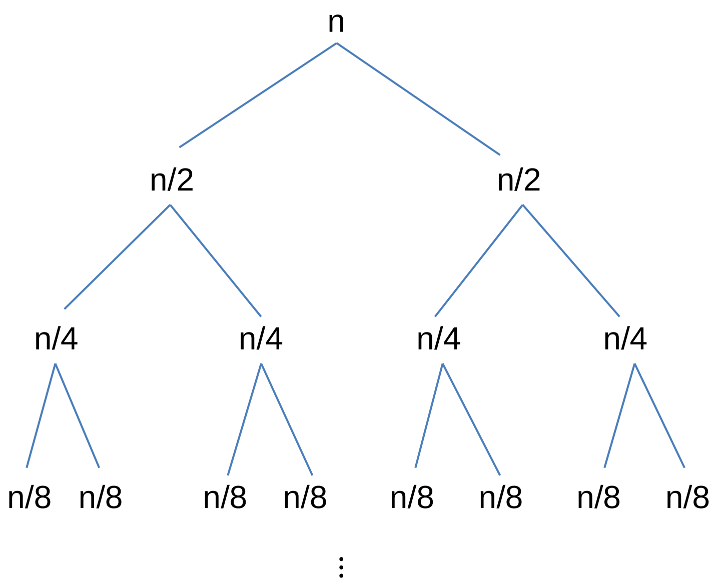
$$\le cn + cn + 4\,T\left(\frac{n}{4}\right)$$
$$\le cn + cn + cn + 8T\left(\frac{n}{8}\right)$$
$$\le 3cn + 2^3 T\left(\frac{n}{2^3}\right)$$
$$\vdots$$
$$\le kcn + 2^k T\left(\frac{n}{2^k}\right)$$
$$\le kcn + 2^k T(1)$$
$$\le kcn + d\,2^k$$
$$\le cn\log n + d\,n$$

Therefore, $T(n)$ is $O(n\log n)$.

Note: When the recurrence is $T(n) \le \cdots$, rather than strict equality, the solution simply becomes $T(n) \le \cdots$. For this reason, we often express the recurrence simply with equality, having in mind that the right side is an upper bound for $T(n)$.

Guess the solution and prove correctness by induction

To arrive at an initial guess, let us consider the merge tree, shown below. At the top level, the algorithm merges $(n/2)$ and $(n/2)$, which costs at most cn time. At the next level, to merge each $(n/4, n/4)$ pair costs $cn/2$. Since there are 2 such pairs, the total cost at this level is $2 \cdot c\,n/2$, thus a total of cn. In summary, the cost of merging at each level of tree is cn time. And there are about $\log n$ levels. (The exact number is not needed.) Therefore, the total costs is $O(n\log n)$.



$$M\left(\frac{n}{2},\frac{n}{2}\right) = Cn$$

$$2\cdot M\left(\frac{n}{4},\frac{n}{4}\right) = 2\cdot\frac{cn}{2} = cn$$

$$4\cdot M\left(\frac{n}{8},\frac{n}{8}\right) = 4\cdot\frac{cn}{4} = cn$$

We concluded in our estimate that the total time is $O(n \log n)$. Based on this, there are several possibilities for guessing the solution form.

- $T(n) \le A\, n \log n + Bn$
- $T(n) \le A\, n \log n$

We leave the induction proof to the student.

Generalization of time analysis for any integer size n

The recurrence equation for the general case may be expressed as follows.

$$T(n) = \begin{cases} T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + c\, n, & n \ge 2 \\ d, & n = 1 \end{cases}$$

It is easy to prove by induction that the solution is

$$T(n) \le cn\lceil \log n \rceil$$

The proof is left to the student as exercise.

Module 8: Classes of Recurrence Equations

Reading from the Textbook: Chapter 7 Recurrences

Introduction

In the last module, we studied several divide-and-conquer algorithms and the recurrence equations for their analysis. In this module, we derive the solution of a more general class of recurrence equations which commonly arise from divide-and-conquer algorithms. The solution form of this class of recurrences is known as the Master-Theorem. We also study the class of linear recurrences. The well-known Fibonacci sequence is an example of a linear recurrence. We will find the exact solution of this sequence and relate the solution to the famous Golden Ratio, which is claimed to appear in many places in nature!

Contents

Divide-and-Conquer Recurrences

The following recurrence is a general form of recurrences which commonly arise in divide-and-conquer algorithms. ($a, b, c, d,$ and β are constants determined by the specific algorithm, and $n = b^k$ for some integer k.)

$$T(n) = \begin{cases} a \cdot T\left({}^{n}/_{b}\right) + c \cdot n^{\beta}, & n > 1 \\ d, & n = 1 \end{cases}$$

This recurrence may correspond to a divide-and-conquer algorithm which divides a problem of size n into $'a'$ subproblems, each of size ${}^{n}/_{b}$. The term $T\left({}^{n}/_{b}\right)$ represents the time for solving each subproblem. And the term cn^{β}represents the additional work for "combining" the solutions of the subproblems to find the overall solution. For example, the recurrence for Mergesort is $T(n) = 2T\left({}^{n}/_{2}\right) + cn$. The term $T\left({}^{n}/_{2}\right)$ is the time for sorting each half of the array, and cn is the time for merging the two sorted halves.

We will use repeated-substitution to find the general form of the solution.

$$\begin{aligned}
T(n) &= cn^{\beta} + a \cdot T\left({}^{n}/_{b}\right) \\
&= cn^{\beta} + a \cdot \left[c\left({}^{n}/_{b}\right)^{\beta} + aT\left({}^{n}/_{b^2}\right)\right] \\
&= cn^{\beta} + cn^{\beta}\left(\frac{a}{b^{\beta}}\right) + a^2\, T\left({}^{n}/_{b^2}\right) \\
&= cn^{\beta} + cn^{\beta}\left(\frac{a}{b^{\beta}}\right) + a^2 \cdot \left[c\left({}^{n}/_{b^2}\right)^{\beta} + aT\left({}^{n}/_{b^3}\right)\right] \\
&= cn^{\beta} + cn^{\beta}\left(\frac{a}{b^{\beta}}\right) + cn^{\beta}\left(\frac{a^2}{(b^2)^{\beta}}\right) + a^3\, T\left({}^{n}/_{b^3}\right) && \text{Use the equality } (b^2)^{\beta} = (b^{\beta})^2 \\
&= cn^{\beta} + cn^{\beta}\left(\frac{a}{b^{\beta}}\right) + cn^{\beta}\left(\frac{a^2}{(b^{\beta})^2}\right) + a^3\, T\left({}^{n}/_{b^3}\right) && \text{Factor out } cn^{\beta} \\
&= cn^{\beta}\left[1 + \frac{a}{b^{\beta}} + \left(\frac{a}{b^{\beta}}\right)^2\right] + a^3\, T\left({}^{n}/_{b^3}\right) \\
&\vdots \\
&= cn^{\beta}\left[1 + \frac{a}{b^{\beta}} + \left(\frac{a}{b^{\beta}}\right)^2 + \cdots + \left(\frac{a}{b^{\beta}}\right)^{k-1}\right] + a^k T\left({}^{n}/_{b^k}\right) && \text{Recall } n = b^k \\
&= cn^{\beta}\left[1 + \frac{a}{b^{\beta}} + \left(\frac{a}{b^{\beta}}\right)^2 + \cdots + \left(\frac{a}{b^{\beta}}\right)^{k-1}\right] + a^k T(1) && \text{Use } T(1) = d
\end{aligned}$$

And use the equality

$$a^k = a^{\log_b n} = n^{\log_b a}$$

(The reason for the equality $a^{\log_b n} = n^{\log_b a}$ is seen if we take the log of both sides, which becomes $\log_b n \log_b a$.) Let $h = \log_b a$. Then,

$$T(n) = cn^\beta \left[1 + \frac{a}{b^\beta} + \left(\frac{a}{b^\beta}\right)^2 + \cdots + \left(\frac{a}{b^\beta}\right)^{k-1}\right] + dn^h$$

We now consider two cases, depending on whether the ratio $\frac{a}{b^\beta}$ equals 1 or not.

1. The ratio $\frac{a}{b^\beta} \neq 1$, which means $a \neq b^\beta$, or $\log_b a \neq \beta$. That is, $h \neq \beta$.
   In this case, the summation inside the brackets is geometric sum. So,

$$T(n) = cn^\beta \cdot \frac{\left(\frac{a}{b^\beta}\right)^k - 1}{\left(\frac{a}{b^\beta}\right) - 1} + dn^h$$
$$= \frac{c}{\left(\frac{a}{b^\beta}\right) - 1} n^\beta \left[\left(\frac{a}{b^\beta}\right)^k - 1\right] + dn^h$$

   Recall $a^k = a^{\log_b n} = n^{\log_b a} = n^h$, and $\left(b^\beta\right)^k = (b^k)^\beta = n^\beta$. So,

$$T(n) = \frac{c}{\left(\frac{a}{b^\beta}\right) - 1} n^\beta \left[\frac{n^h}{n^\beta} - 1\right] + dn^h$$
$$= \frac{c}{\left(\frac{a}{b^\beta}\right) - 1} \left(n^h - n^\beta\right) + dn^h$$
$$= \left[\frac{c}{\left(\frac{a}{b^\beta}\right) - 1} + d\right] \cdot n^h + \left[\frac{-c}{\left(\frac{a}{b^\beta}\right) - 1}\right] \cdot n^\beta$$

   Therefore,

$$\boxed{T(n) = A \cdot n^h + B \cdot n^\beta}$$

   where *A* and *B* are the constants as derived.
   The solution is a linear sum of two terms: n^h and n^β. The first term n^h comes from the term $a \cdot T\left({}^{n}/_{b}\right)$ in the recurrence. And the second term n^β comes from $c \cdot n^\beta$ in the recurrence.
2. The ratio $\frac{a}{b^\beta} = 1$, which means $a = b^\beta$, or $\log_b a = \beta$. That is, $h = \beta$.
   In this case, the summation inside the brackets is $[1 + 1 + \cdots + 1] = k = \log_b n$.

$$T(n) = cn^\beta \log_b n + dn^h$$
$$= cn^h \log_b n + dn^h \qquad (\text{since } h = \beta)$$

   Therefore,

$$\boxed{T(n) = A \cdot n^h \log_b n + B \cdot n^h}$$

   where *A* and *B* are the constants as derived.

Let us summarize the solution we derived for the above class of divide-and-conquer recurrences.

Summary (Master Theorem): Given the recurrence equation

$$T(n) = \begin{cases} a \cdot T\left({}^{n}/_{b}\right) + c \cdot n^{\beta}, & n > 1 \\ d, & n = 1 \end{cases}$$

Let $h = \log_{\mathrm{b}} a$. The solution has the following general forms and bounds. (*A* and *B* are some constants for each case.)

$$T(n) = \begin{cases} An^{h} + Bn^{\beta} & = \Theta(n^{h}), & h > \beta \\ An^{h} + Bn^{\beta} & = \Theta\left(n^{\beta}\right), & h < \beta \\ An^{h}\log n + Bn^{h} & = \Theta(n^{h}\log n), & h = \beta \end{cases}$$

Note: This is a slightly simpler version of what is known as the ***Master Theorem*** in many textbooks on algorithms.

We derived the values of the constants *A* and *B* in terms of the given constants $a, b, c, d,$ and β. However, the values of the constants *A* and *B* need not be memorized. Normally, we may be interested only in the order of the solution. And if the exact value of the solution is desired, the constants may be easily computed as illustrated in the following examples.

Example 1: Finding Max of an array by divide-and-conquer

As discussed earlier, the recurrence for this algorithm for the case when $n = 2^k$ is:

$$f(n) = \begin{cases} 0, & n = 1 \\ 2f(n/2) + 1, & n \geq 2 \end{cases}$$

Let us apply our Master Theorem to find the solution form.

$$a = 2, b = 2, \beta = 0$$
$$h = \log_2 2 = 1$$

Since $h \neq \beta$, the solution form is

$$f(n) = An^h + Bn^\beta$$
$$= An + B$$

Finding the constants A and B

If this solution form was an initial "guess", we would apply induction to prove it correct and find the constants. But in this case, since we know the solution form is correct, we don't need to apply induction. Instead, we may use an easier method of simply plugging in two values for n.

$n = 1$:

$$f(1) = 0 \quad \text{(from the recurrence}$$
$$= A + B \quad \text{(from the solution form)}$$

$n = 2$:

$$f(2) = 2f(1) + 1 = 2 * 0 + 1 = 1$$
$$= 2A + B$$

So we have two equations:

$$\begin{cases} A + B = 0 \\ 2A + B = 1 \end{cases}$$

We find the constants: $A = 1,\ B = -1$. Therefore,

$$\boxed{f(n) = n - 1}$$

■

Example 2: Binary Search Algorithm (Special case $\boldsymbol{n = 2^k}$)

As we saw, the recurrence equation for the number of key-comparisons in this algorithm is

$$f(n) = \begin{cases} 1, & n = 1 \\ 1 + f\left(\frac{n}{2}\right), & n \geq 2 \end{cases}$$

By Master Theorem:

$$a = 1, b = 2, \beta = 0$$
$$h = \log_2 1 = 0$$

Since $h = \beta,$ the solution form is:

$$f(n) = An^h \log_2 n + Bn^h$$
$$= A \log n + B$$

To find the constants A and B, use $n = 1$ and $n = 2$.

$$f(1) = 1$$
$$= A \log 1 + B = B$$

$$f(2) = 1 + f(1) = 1 + 1 = 2$$
$$= A \log 2 + B = A + B$$

So we have two equations: $B = 1,\ A + B = 2,$ which give: $A = 1,\ B = 1.$ Therefore,

$$\boxed{f(n) = \log n + 1}$$

■

Example 3: Mergesort (Special case $n = 2^k$)

As we saw, the recurrence equation for this algorithm is

$$T(n) = \begin{cases} 2\,T\left(\frac{n}{2}\right) + c\,n, & n \geq 2 \\ d, & n = 1 \end{cases}$$

We find the solution by use of the Master Theorem:

$$a = 2, b = 2, \beta = 1$$
$$h = \log_2 2 = 1$$

Since $h = \beta$, the solution form is

$$T(n) = An^h \log_2 n + Bn^h$$
$$= An \log n + Bn$$

We find the constants A and B quickly by using two values $n = 1,\ n = 2$.

$$T(1) = d$$
$$= An \log 1 + B = B$$

$$T(2) = 2T(1) + c * 2 = 2d + 2c$$
$$= A * 2 \log 2 + B * 2 = 2A + 2B$$

So we have two equations: $B = d,\ \ A + B = c + d$.

Solving them gives: $B = d,\ A = c$. Therefore,

$$\boxed{T(n) = cn \log n + dn}$$

■

Example 4: Strassen's Matrix Multiplication Algorithm

In one of the earlier modules, we discussed a straightforward way of multiplying two matrices of size $n \times n$, which takes $O(n^3)$ time. Strassen's algorithm is an asymptotically faster and more sophisticated algorithm, which is based on the divide-and-conquer strategy. We will not discuss the details of this algorithm here, but we present the recurrence equation arising from it, and analyze its time complexity.

Let $T(n)$ be the time to multiply two $n \times n$ matrices. The algorithm uses 7 multiplications of submatrices of size $\frac{n}{2} \times \frac{n}{2}$, each with time $T(n/2)$, plus some additions and subtractions of submatrices, which take O(n^2) time.

$$T(n) = \begin{cases} 7\,T(n/2) + cn^2, & n > 1 \\ d, & n = 1 \end{cases}$$

We use the master theorem to find the solution form.

$$a = 7, b = 2, \beta = 2$$
$$h = \log_2 7 \cong 2.8$$

Therefore,

$$T(n) = An^h + Bn^\beta$$
$$= An^{\log_2 7} + Bn^2 = \Theta(n^{\log_2 7})$$

If we want the exact value of the constants, we plug in two values of n to get two equations for A and B.

$$T(1) = d$$
$$= A + B$$

$$T(2) = 7T(1) + 4c = 7d + 4c$$
$$= A2^{\log_2 7} + B2^2 = 7A + 4B$$

So we have:

$$A + B = d$$
$$7A + 4B = 7d + 4c$$

Therefore, $A = d + \frac{4}{3}c$, $B = -\frac{4}{3}c$. So, the exact solution is

$$T(n) = \left(d + \frac{4}{3}c\right) \cdot n^{\log_2 7} - \frac{4}{3}c \cdot n^2$$

■

Linear Recurrences

Let $\{F_1, F_2, \cdots, F_n\}$ be a sequence. A linear recurrence expresses F_n as a linear function of several of its predecessors $F_{n-1}, F_{n-2}, \cdots$. Let us start with some simple examples.

Example: Compound Interest

Suppose a saving bank offers yearly interest of 5% compounded annually. Suppose the initial deposit is $1000, and F_n is the accumulated amount at the end of n years. Then,

$$F_n = \begin{cases} 1000, & n = 0 \\ 1.05 * F_{n-1}, & n \geq 1 \end{cases}$$

It is easy to find the solution of this recurrence by repeated substitution.

$$\begin{aligned} F_n &= 1.05 * F_{n-1} \\ &= 1.05 * 1.05 * F_{n-2} \\ &= (1.05)^3 * F_{n-3} \\ &\vdots \\ &= (1.05)^n * F_0 \\ &= 1000 * (1.05)^n \end{aligned}$$

Example: Towers of Hanoi

We studied the algorithm for this problem earlier. The number of single moves to accomplish moving n disks is expressed by the recurrence equation

$$F_n = \begin{cases} 1, & n = 1 \\ 2F_{n-1} + 1, & n \geq 2 \end{cases}$$

We obtained the solution of this recurrence (by repeated substation) as

$$F_n = 2^n - 1$$

Example: Fibonacci Sequence

The recurrence equation for this well-known sequence is defined as

$$F_n = \begin{cases} 1, & n = 1 \\ 1, & n = 2 \\ F_{n-1} + F_{n-2}, & n \geq 3 \end{cases}$$

■

In general, a homogeneous linear recurrence of order k with constant coefficients has the following form,

$$F_n = c_1 F_{n-1} + c_2 F_{n-2} + \cdots + c_k F_{n-k}$$

together with k initial values for $F_1, F_2, \cdots, F_k$.

For first-order recurrences of the first two examples above, we saw the solution in both cases is exponential form r^n (where $r = 1.05$ and $r = 2$, respectively).

For higher-order recurrences, we start with this same form as an initial trial.

Linear Recurrence of Order 2

Let us find the solution for the following second-order linear recurrence.

$$F_n = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ 5F_{n-1} - 6F_{n-2}, & n \geq 2 \end{cases}$$

We first concentrate on the recursive line of the recurrence. (Later, we deal with the initial values for $n = 0, n = 1$.)

$$F_n - 5F_{n-1} + 6F_{n-2} = 0$$

We start with the following initial "guess", where r is a constant to be determined.

$$F_n = r^n$$

Substitute back in the recurrence:

$$\begin{aligned} &r^n - 5r^{n-1} + 6r^{n-2} = 0 \\ &r^{n-2}(r^2 - 5r + 6) = 0 \\ &r^2 - 5r + 6 = 0 \qquad \text{(Characteristic Equation)} \end{aligned}$$

The latter quadratic equation is called the characteristic equation of the recurrence. The roots of the characteristic equation may be found by either factoring out the polynomial, or using the quadratic equation formula.

$$\begin{aligned} &(r-3)(r-2) = 0 \\ &r_1 = 3, \ r_2 = 2 \end{aligned}$$

This means that each of the solution forms $F_n = 3^n \text{ and } F_n = 2^n$ satisfies the recursive line of the recurrence equation (but not the initial values for $n = 0, n = 1$).

In fact, a linear sum of the two,

$$F_n = A3^n + B2^n$$

(which is called the *complete solution*) also satisfies the recurrence equation. This is easily verified by substituting the complete solution form back in the recurrence:

$$F_n - 5F_{n-1} + 6F_{n-2} = 0$$

$$(A3^n + B2^n) - 5(A3^{n-1} + B2^{n-1}) + 6(A3^{n-1} + B2^{n-2}) = 0$$

$$A(3^n - 5 * 3^{n-1} + 6 * 3^{n-2}) + B(2^n - 5 * 2^{n-1} + 6 * 2^{n-2}) = 0$$

$$A\,3^{n-2}\left(\underbrace{3^2 - 5 * 3 + 6}_{\substack{=0 \\ \textit{since 3 is a root}}}\right) + B2^{n-2}\left(\underbrace{2^2 - 5 * 2 + 6}_{\substack{=0 \\ \textit{since 2 is a root}}}\right) = 0$$

The quantity inside each parenthesis is 0, because 3 and 2 are the two roots of the characteristic equation ($r^2 - 5r + 6 = 0$). Therefore, the complete solution satisfies the recursive line of the recurrence, as claimed earlier.

It remains to satisfy the initial conditions of the recurrence (for $n = 0$ and $n = 1$). We use these initial values to find the constants A and B.

Finding the constants A and B

The complete solution form is:

$$F_n = A3^n + B2^n$$

And the two base cases of the recurrence (also called initial values) are $F_0 = 0, F_1 = 1$. So,

$$F_0 = 0 = A + B$$
$$F_1 = 1 = 3A + 2B$$

Solving these two equations for A and B gives:

$$A = 1, \qquad B = -1$$

Therefore,

$$\boxed{F_n = 3^n - 2^n}$$

Example: Find the complete solution of the following second-order recurrence, with the initial values $F_0 = 1,\ F_1 = 7$.

$$F_n = 8F_{n-1} - 15F_{n-2}, \quad n \geq 2$$

Substitute the solution $F_n = r^n$.

$$\begin{aligned} &F_n - 8F_{n-1} + 15F_{n-2} = 0 \\ &r^n - 8r^{n-1} + 15r^{n-2} = 0 \\ &r^{n-2}(r^2 - 8r + 15) = 0 \\ &r^2 - 8r + 15 = 0 \\ &(r-5)(r-3) = 0 \\ &r_1 = 5, \qquad r_2 = 3 \end{aligned}$$

The complete solution is

$$F_n = A\,5^n + B\,3^n$$

Use the initial conditions to find A and B:

$$\begin{aligned} F_0 &= 1 = A + B \\ F_1 &= 7 = 5A + 3B \end{aligned}$$

Solve for A and B:

$$A = 2, \ \ B = -1$$

$$\boxed{F_n = 2 * 5^n - 3^n}$$

■

Example: A recurrence equation of order-3 has the following initial values

$$F_0 = 0, \qquad F_1 = 7, \qquad F_2 = 25$$

and the following characteristic equation (in factored form)

$$(r-3)(r-2)(r-1) = 0$$

(a) Find the complete solution.
(b) Work backward from the characteristic equation to find the recurrence equation.

Solution: Since the characteristic equation is already given in factored form, we know the roots are

$$r_1 = 3, \qquad r_2 = 2, \qquad r_3 = 1$$

So the complete solution is

$$F_n = A\,3^n + B\,2^n + C\,1^n$$
$$F_n = A\,3^n + B\,2^n + C$$

Use the initial conditions to find three equations for the constants.

$$F_0 = 0 = A + B + C$$
$$F_1 = 7 = 3A + 2B + C$$
$$F_2 = 25 = 9A + 4B + C$$

Solve for the constants:

$$A = 2, \qquad B = 3, \qquad C = -5$$

The exact solution is

$$\boxed{F_n = 2 * 3^n + 3 * 2^n - 5}$$

To find the original recurrence equation, we work backward from the characteristic equation.

$$(r-3)(r-2)(r-1) = 0$$
$$(r^2 - 5r + 6)(r-1) = 0$$
$$r^3 - 6r^2 + 11r - 6 = 0$$
$$r^n - 6r^{n-1} + 11r^{n-2} - 6r^{n-3} = 0$$

Therefore,

$$\boxed{F_n = 6F_{n-1} - 11F_{n-2} + 6F_{n-3}}$$

■

Repeated Roots

The linear recurrences so far had characteristic equations with distinct roots. We now consider what happens if the roots are not distinct. Consider a second-order linear recurrence, with $F_0 = 2,\ F_1 = 9,$ and

$$F_n = 6F_{n-1} - 9F_{n-2}\,, \quad n \geq 2$$
$$F_n - 6F_{n-1} + 9F_{n-2} = 0$$

Again, let us substitute the solution $F_n = r^n$. Let $P(r^n)$ denote the polynomial obtained when we substitute $F_n = r^n$ in the recurrence.

$$\begin{aligned} P(r^n) &= r^n - 6r^{n-1} + 9r^{n-2} \\ &= r^{n-2}(r^2 - 6r + 9) \\ P(r^n) &= r^{n-2}(r-3)^2 = 0 \\ (r-3)^2 &= 0 \qquad\qquad \text{Reduced Characteristic Equation} \end{aligned}$$

The characteristic equation has two roots, both $r = 3$.

(1) We already know that one solution has the form $F_n = r^n,\ r = 3$.
(2) We now claim that there is a second solution of the form:

$$F_n = n\,r^n,\ \ r = 3.$$

(The proof of this claim uses some calculus, namely the concept of derivative.)

Proof: We will provide the proof in two ways.

First Proof: One way to verify the correctness of the solution is to simply substitute it back in the recurrence equation.

$$P(nr^n) = nr^n - 6(n-1)r^{n-1} + 9(n-2)r^{n-2}$$
$$P(nr^n) = r^{n-2}[nr^2 - 6(n-1)r + 9(n-2)]$$

For $r = 3,$ it is immediately verified that $P(nr^n) = 0$.

$$P(n\,3^n) = 3^{n-2}[9n - 18(n-1) + 9(n-2)] = 0$$

Although this verifies the correctness of the solution, is still leaves us wondering as how we guessed the solution in the first place. We now provide a second proof to provide the needed insight.

Alternative Proof: Since the characteristic polynomial $P(r^n)$ has a square factor, namely $(r-3)^2$, then the derivative of the characteristic equation also has a factor $(r-3),$ thus a root $r = 3$. This is immediately seen from the reduced form.

$$\frac{d}{dr}(r-3)^2 = 2(r-3) = 0, \qquad r = 3$$

It is easy to see that the same is true about the derivative of the earlier factored form, $r^{n-2}(r-3)^2$. Thus, we know that the derivative of the original polynomial form has a root $r = 3$.

$$\frac{d}{dr}(\mathrm{P}(\mathrm{r}^{\mathrm{n}})) = \frac{d}{dr}\,(r^n - 6r^{n-1} + 9r^{n-2})$$
$$= n\,r^{n-1} - 6(n-1)\,r^{n-2} + 9(n-2)\,r^{n-3} = 0 \qquad \rightarrow \;\; r = 3$$

If we multiply both sides of the above equation by $r,$ it will still have a root $r = 3$.

$$n\,r^n - 6(n-1)\,r^{n-1} + 9(n-2)\,r^{n-2} = 0 \qquad \rightarrow \;\; r = 3$$

But this latter equation is exactly what we get by directly substituting the second solution form $F_n = n\,r^n$ into the recurrence equation. That is,

$$P(nr^n) = n\,r^n - 6(n-1)\,r^{n-1} + 9(n-2)r^{n-2} = 0$$

Since we showed that this equation has a root $r = 3,$ it means that

$$F_n = n\,r^n, \quad r = 3$$

is a solution as claimed. ■

Having proved the second solution, we obtain the complete solution as a linear sum of the two solutions, as before:

$$F_n = A\,3^n + Bn\,3^n$$

The constants are found by using the initial conditions.

$$F_0 = 2 = A$$
$$F_1 = 9 = 3A + 3B$$

So, $A = 2,\; B = 1$.

$$\boxed{F_n = 2 *\; 3^n + n\,3^n = (n+2)\,3^n}$$

We will provide some more examples of repeated roots.

Example: Find the complete solution of the following second-order linear recurrence, with the initial conditions $F_1 = 16,\ F_2 = 44$.

$$F_n = 4F_{n-1} - 4F_{n-2}$$

Solution: Substitute the solution $F_n = r^n$.

$$\begin{aligned}&F_n - 4F_{n-1} + 4F_{n-2} = 0\\&r^n - 4\,r^{n-1} + 4\,r^{n-2} = 0\\&r^{n-2}(r^2 - 4r + 4) = 0\\&r^{n-2}(r-2)^2 = 0\\&\text{Roots: } r_1 = r_2 = 2\end{aligned}$$

Since there are repeated roots, the complete solution has the form:

$$F_n = A\,2^n + Bn\,2^n$$

The constants are found by using the initial conditions.

$$\begin{aligned}F_1 &= 2A + 2B = 16\\F_2 &= 4A + 8B = 44\end{aligned}$$

So, $A = 5,\ B = 3$.

$$\boxed{F_n = 5\,2^n + 3n\,2^n}$$

■

Example: Find the general solution form of a linear recurrence of order 3, which has the following factored characteristic equation.

$$(r-5)^2(r-2) = 0$$

Solution: The roots are: $r_1 = r_2 = 5,\ \ r_3 = 2$.

The complete solution has the form:

$$F_n = A\,5^n + Bn\,5^n + C\,2^n$$

The constants (A, B, C) may be computed by using the initial conditions (three base cases), not provided here.

■

Fibonacci Numbers

Let us now look at the Fibonacci sequence, a second-order linear recurrence, with many applications in computer science and other fields. The sequence is defined as $F_0 = 0,\ F_1 = 1,$ and

$$F_n = F_{n-1} + F_{n-2}, \quad n \geq 2$$

Below are the tabulated values up to $n = 15$.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F_n	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610

To find the solution, we substitute $F_n = r^n$.

$$\begin{aligned}
&F_n - F_{n-1} - F_{n-2} = 0\\
&r^n - r^{n-1} - r^{n-2} = 0\\
&r^{n-2}(r^2 - r - 1) = 0\\
&r^2 - r - 1 = 0
\end{aligned}$$

The quadratic equation formula is used to find the roots. [Recall the general formula: given the equation $ar^2 + br + c = 0,$ the roots are $r = \frac{-b \mp \sqrt{(b^2 - 4ac)}}{2}$.]

$$r_1 = \frac{1+\sqrt{5}}{2} \cong 1.618033989, \qquad r_2 = \frac{1-\sqrt{5}}{2} \cong -0.618033989$$

The complete solution has the form:

$$F_n = A\, r_1^n + B\, r_2^n$$

The Initial conditions (base cases) are used to find the constants.

$$\begin{aligned}
&F_0 = 0 = A + B\\
&F_1 = 1 = A\ \frac{1+\sqrt{5}}{2} + B\ \frac{1-\sqrt{5}}{2}
\end{aligned}$$

The constants are: $A = {}^{1}/_{\sqrt{5}}\ ,\ B = {}^{-1}/_{\sqrt{5}}$, and the exact solution is:

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

It is interesting to note that although F_n is a purely integer function, the closed-form expression for it is so complex-looking!

Approximation of Fibonacci Numbers

Observe that $r_2 \cong -0.618$ is a negative fraction and for large n, the term r_2^n becomes very negligible. (For $n \geq 10$, the absolute value of $r_2^n < 0.008131.$) So, for large n,

$$F_n \cong \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n = \frac{1}{\sqrt{5}}\,(1.618)^n$$

The following table shows how close this approximation is to the actual value of F_n. Also shown is the ratio F_n/F_{n-1}, which gets very close to 1.618 for large n.

n	F_n	$\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$	$\frac{1}{\sqrt{5}}\,(1.618)^n$	Ratio F_n/F_{n-1}
1	1	0.72361	0.72359	
2	1	1.17082	1.17077	1.00000
3	2	1.89443	1.89431	2.00000
4	3	3.06525	3.06499	1.50000
5	5	4.95967	4.95915	1.66667
6	8	8.02492	8.02391	1.60000
7	13	12.98460	12.98269	1.62500
8	21	21.00952	21.00599	1.61538
9	34	33.99412	33.98769	1.61905
10	55	55.00364	54.99208	1.61765
11	89	88.99775	88.97719	1.61818
12	144	144.00139	143.96509	1.61798
13	233	232.99914	232.93552	1.61806
14	377	377.00053	376.88967	1.61803
15	610	609.99967	609.80749	1.61804
16	987	987.00020	986.66852	1.61803
17	1,597	1,596.99987	1,596.42967	1.61803
18	2,584	2,584.00008	2,583.02321	1.61803
19	4,181	4,180.99995	4,179.33156	1.61803
20	6,765	6,765.00003	6,762.15846	1.61803
21	10,946	10,945.99998	10,941.17238	1.61803
22	17,711	17,711.00001	17,702.81692	1.61803
23	28,657	28,656.99999	28,643.15777	1.61803
24	46,368	46,368.00000	46,344.62928	1.61803
25	75,025	75,025.00000	74,985.61017	1.61803

Applications

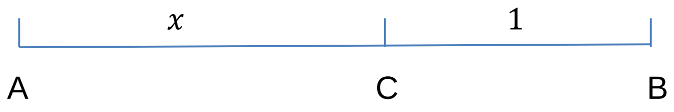
Fibonacci numbers find many applications in computer science. Below are some examples:

- Fibonacci search tree
- Analysis of AVL Tree, which is a balanced search tree. This analysis uses a sequence very close to Fibonacci numbers.
- Time complexity analysis of Euclid's Algorithm for the greatest-common-divisor.

Next, we relate Fibonacci numbers to the famous Golden Ratio, which is claimed to appear throughout nature!

The Golden Ratio

The great ancient mathematician and the founder of Euclidean Geometry, Euclid of Alexandria, apparently introduced a number around 300 B.C. which later became known as the Golden Ratio. This number was introduced by Euclid as a division of a line segment AB into two segments (AC and CB) in such a way that the ratio of the greater segment over the smaller one is the same as the ratio of the whole line segment over the greater one. (Euclid called it "extreme and mean ratio".)



Let x be the length of the larger segment, and 1 be the length of the shorter segment, thus $(x+1)$ the total length. Then, the division must be such that

$$\frac{x}{1} = \frac{x+1}{x}$$

This produces the quadratic equation

$$x^2 - x - 1 = 0$$

This equation is exactly the same as the characteristic equation for Fibonacci sequence. The larger positive root is known as the Golden Ratio, which is the ratio of the larger segment to the smaller one.

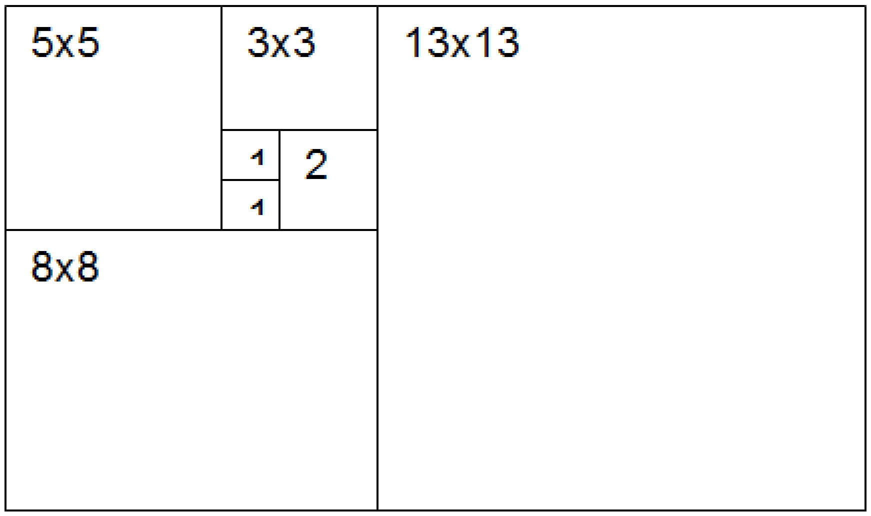
$$\boxed{Golden\ Ratio = \ x = \frac{1+\sqrt{5}}{2} \cong 1.618}$$

Examples in Nature

There is a rich body of evidence that Fibonacci numbers and the Golden Ratio show up many places in nature. Some of this material is mathematically sound and quite convincing. Some of the other examples quoted in the literature are more subjective and open to one's perception.

An interesting source is the book *The Golden Ratio*, by Mario Livio. (Broadway Books, Random House, Inc. 2002, ISBN 0-7679-0816-3) In the introductory parts, the author tries hard to excite the reader, and does not get to the meat of the matter very quickly. But, once you get passed this introductory material, you will find many intriguing examples in this book.

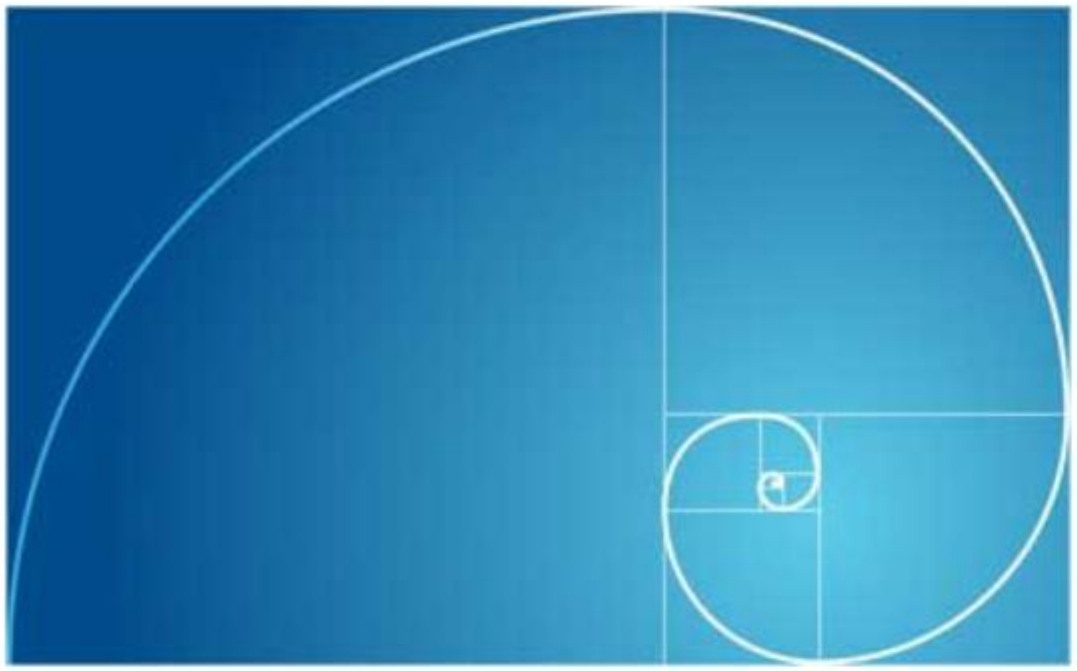
One common example is how squares with sizes that follow Fibonacci numbers $(1,1,2,3,5,8,13,\cdots)$ fit perfectly into spiraling shells. Below is a geometrical arrangement of such squares. Start at the top smallest square of size 1×1 at the center, and move in counterclockwise direction (down, right, up, left, $\cdots$) to get to the next square of size 1×1, then 2×2, then 3×3, then 5, 8, 13, and so on.



A spiraling sea shell with such squares is shown below, copied from the website:

http://science.howstuffworks.com/math-concepts/fibonacci-nature1.htm

The article by Robert Lamb on this website contains numerous examples of where Fibonacci numbers and the Golden Ratio are found in nature. The article starts with an introductory paragraph that many such observations on numbers may be plain coincidence and "numerological superstitions". But the article quickly moves on to say that Fibonacci numbers appear often enough in nature to reflect “some naturally occurring patterns.”



The golden ratio is expressed in spiraling shells. In the above illustration, areas of the shell's growth are mapped out in squares. If the two smallest squares have a width and height of 1, then the box to their left has measurements of 2. The other boxes measure 3, 5, 8 and 13.
©iStockphoto.com/Janne Ahvo

Another example is a pine cone shown below (copied from our Discrete Math textbook by Richard Johnsonbaugh). The picture claims there are 13 clockwise spirals (marked with white thread) and 8 counterclockwise spirals (marked by dark thread). I find this type of observation somewhat subjective. (If I were told that there are 14 clockwise spirals and 7 counterclockwise spirals, rather than 13 and 8, I would stare long enough until I either got dizzy or I became convinced!)

Similar observations have been made about orange, cauliflower, pineapple, and many other things in nature.

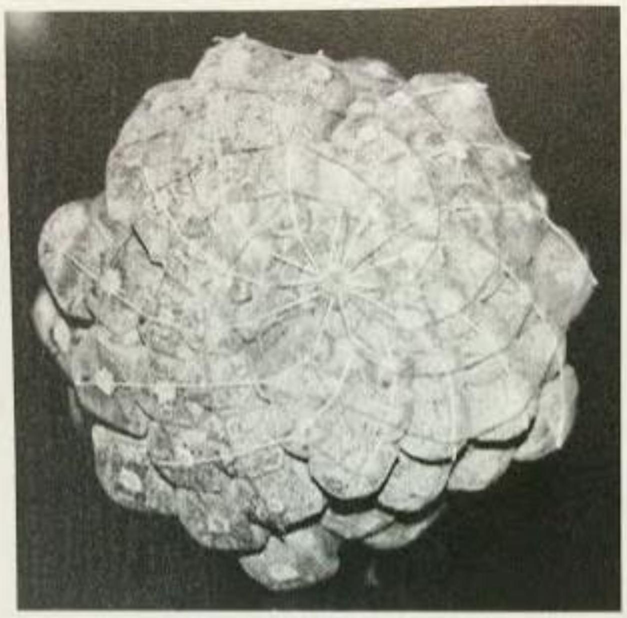


Figure 4.4.1 A pine cone. There are 13 clockwise spirals (marked with white thread) and 8 counterclockwise spirals (marked with dark thread). [*Photo by the author; pine cone courtesy of André Berthiaume and Sigrid (Anne) Settle.*]

In closing, I would like to leave you with the following intriguing thought. First observe that the specific values in Fibonacci sequence entirely depend on the two starting values (base cases), which are arbitrarily defined. Let us redefine the sequence with a slightly different base values, as shown below. (The value for $n = 2$ has been changed from 1 to 3.) The recursive definition for this new sequence, G_n, remains the same.

$$G_n = \begin{cases} 1, & n = 1 \\ 3, & n = 2 \\ G_{n-1} + G_{n-2}, & n \geq 3 \end{cases}$$

This recurrence has the same roots as before, and the Golden Ratio is still one root. But the numbers which are generated have different values. The first 10 numbers in this sequence are shown below, along with Fibonacci numbers.

n	1	2	3	4	5	6	7	8	9	10
F_n	1	1	2	3	5	8	13	21	34	55
G_n	1	3	4	7	11	18	29	47	76	123

Note that F_n and G_n are of the same order. In fact, it is easy to prove by induction that

$$F_n \leq G_n \leq 3F_n, \qquad \forall n.$$

Will nature alter its growth of sea shells, pine cones, orange, and cauliflower to match the pattern 4, 7, 11 in this sequence (rather than the current pattern of $3,5,8$)? Probably not, since the base values in our sequence are not divinely inspired!

Module 9: Counting Methods, Permutations, and Combinations

Reading from the Textbook: Chapter 6 Counting Methods

Introduction

The counting methods studied in this module are useful in many areas of computer science, including hashing techniques and probability theory. This material is commonly known as combinatorics. We start by two basic principles called multiplication principle and addition principle, and then study permutations and combinations. We use these concepts to derive binomial coefficients and Pascal's triangle. In the next module, we will see an important application of combinatorics, namely discrete probability.

Contents

Basic Principles

Multiplication Principle

Consider the example of a diner menu. Suppose the menu for “dinner special” consists of

1. A choice of 3 possible appetizers
2. A choice of 4 main entrées
3. A choice of 5 available desserts

The total number of possible orders becomes

$$3 * 4 * 5 = 60$$

This may also be expressed as the Cartesian product of sets. Let A be the set of appetizers, E the set of entrees, and D the set of desserts. Then, the number of possible orders is

$$|A \times E \times D| = |A| * |E| * |D|.$$

This is an example of the multiplication principle.

Multiplication Principle: Suppose a process consists of k independent steps:

Step 1: Choose one out of n_1 possible ways;
Step 2: Choose one out of n_2 possible ways;
⋮
Step k: Choose one out of n_k possible ways.

Then, the total number of possible choices is

$$n_1 * n_2 * \cdots * n_k$$

Note that the number of choices available in each step is independent of the choices made in previous steps.

Example: Let us compute the total number of n-bit integers. Each bit has two possible values {0, 1}. So the total number is

$$\underbrace{2 * 2 * 2 * \cdots * 2}_{n \text{ times}} = 2^n$$

(An alternative proof is by induction.)

■

Example: What is the number of 8-bit integers which do not start with 11?

Since the number does not start with 11, there are 3 possible values for the leftmost two bits: 00, 01, 10. Therefore, the total number is

$$3 * 2^6 = 3 * 64 = 192$$

■

Example: What is the number of subsets of an n-element set $X = \{1,2,3,\cdots,n\}$?

A subset is formed by a process with n steps. In step i, $1 \leq i \leq n$, decide if element i is to be included in the subset or not. So, the total number of subsets is

$$\underbrace{2 * 2 * 2 * \cdots * 2}_{n \text{ times}} = 2^n$$

This problem may also be related to the number of n-bit integers. Each subset may be represented by an n-bit integer, where bit i equals 1 if and only if element i is in the subset. So the total number of subsets equals the number of n-bit integers.

■

Example: A college consists of the following three departments.

- Computer Science (10 faculty)
- Computer Technology (6 faculty)
- Computer Arts (4 faculty)

The college wants to select one faculty from each department to form a committee. How many ways are there to form such a committee? By the multiplication principle, the number is: $\boxed{10 * 6 * 4 = 240}$

■

Example: Suppose we roll a pair of 6-sided dice, one black and one white.
(a) What is the total number of outcomes?
(b) How many of the outcomes have one or both dice equal 1?

Solution:
(a) By the multiplication principle, the number of outcomes is: $\boxed{6 * 6 = 36}$
(b) One way to do this is to first find the number of outcomes with neither dice = 1. Each dice has 5 possible values. So the total is $5 * 5 = 25$. Therefore, the number of outcomes with one or both dice equal 1 is: $\boxed{36 - 25 - 11}$

■

Addition Principle

Consider rolling a pair of 6-sided dice. Each dice has 6 possible values, so by the multiplication principle, there are a total of 36 outcomes. Let us find the number of ways the sum of the two dice will be either 6 or 7. We consider each case separately.

- Sum $= 6$: There are 5 possible outcomes with this sum, as listed below.

$D_1 D_2$
1 5
2 4
3 3
4 2
5 1

- Sum $= 7$: There are 6 possible outcomes with this sum, as listed below.

$D_1 D_2$
1 6
2 5
3 4
4 3
5 2
6 1

Since the two sets of outcomes are disjoint, then the total number of possible outcomes is sum of the two:

$$5 + 6 = 11$$

This is an example of the addition principle.

Addition Principle: Given a number of sets $S_1,\ S_2, \cdots, S_k$ with the number of elements in them $n_1,\ n_2, \cdots, n_k$ respectively. Suppose the sets are pairwise disjoint. (That is, no pair of sets have any common elements.) The number of elements in the union of the sets is the sum of elements in each set. That is,

$$|S_1 \cup S_2 \cup \cdots \cup S_k| = |S_1| + |S_2| + \cdots + |S_k| = n_1 + n_2 + \cdots + n_k$$

The next two examples make use of both the addition principle and the multiplication principle.

Example: College of Computing has three academic departments:

- CS with 10 faculty members
- IS with 7 faculty members
- IT with 4 faculty members

(Each faculty is a member of only one department.) Each year, the college selects two faculty members to receive Teaching Awards. What is the number of ways for the college to make its selection if the college decides that one faculty must be from CS, and the other from either IS or IT?

Solution: We may view the selection as a 2-step process:

(a) Select one faculty from CS. There are 10 possible choices.
(b) Select one faculty from either IS or IT. By the addition principle, the number of choices is $7 + 4 = 11$.

So, by the multiplication principle, the total number of possible choices is $10 * 11 = 110$.

Let CS, IS, and IT denote the sets of faculty from the respective departments. Then the number of choices may also be expressed as the cardinality (size) of the Cartesian product

$$|CS \times (IS \cup IT)| = |(CS \times IS) \cup (CS \times IT)|$$

■

Example: In the above example, suppose the college decides that the awards must be made to two faculty from two different departments, with no other restriction. What is the number of choices?

Solution: Now there are three possible ways to choose the two departments:

(a) CS and IS: By the multiplication principle, the number of choices is $10 * 7 = 70$.
(b) CS and IT: The number of choices is $10 * 4 = 40$.
(c) IS and IT: The number of choices is $7 * 4 = 28$.

So, by the addition principle, the total number of choices is $70 + 40 + 28 = 138$.

In terms of set operations (Cartesian product) this number may be expressed as

$$|(CS \times IS) \cup (CS \times IT) \cup (IS \times IT)|$$

■

Inclusion-Exclusion Principle

We now consider how the addition principle is revised if the sets are not disjoint. Let us start with an example.

Example: What is the number 4-bit binary integers that either start with 0 or end with 0, or both? The count is computed as follows.

(a) Let A be the set of integers that start with 0. Then, $|A| = 2^3 = 8$
(b) Let B be the set of integers that end with 0. Then, $|B| = 2^3 = 8$.
(c) Let $C = A \cap B$ be the set of integers that start and end with 0, $|C| = 2^2 = 4$.

So the total number of integers that start with 0 or end with 0, or both, is $8 + 8 - 4 = 12$. That is,

$$|A \cup B| = |A| + |B| - |A \cap B| = 8 + 8 - 4 = 12$$

The reasoning is simple. Set A includes those integers that start and end with 0. And set B also includes those integers that start and end with 0. So $|A| + |B|$ counts those integers in the intersection twice, thus we subtract it once to get the correct count.

(Another way to find the above count is to first find the count of those integers that start and end with 1, which is $2^2 = 4$. And we know the total number of 4-bit integers is $2^4 = 16$. Therefore, the desired number is $16 - 4 = 12$.)

■

Inclusion-Exclusion Principle: Given two sets A and B which may not be disjoint. The number of elements in their union is

$$|A \cup B| = |A| + |B| - |A \cap B|$$

The above principle may be generalized to more than two sets in an obvious way. Below is the formula for 3 sets.

$$\begin{aligned}&|A \cup B \cup C| = |A| + |B| + |C|\\&-|A \cap B| - |A \cap C| - |B \cap C|\\&+|A \cap B \cap C|\end{aligned}$$

Again, the reasoning is simple. Those elements that are in the intersection of only two sets, but not in the intersection of all three, are counted twice in line 1, and subtracted once in line 2, so they are counted correctly. But those that are in the intersection of all three are counted three time in line 1, discounted (subtracted) three time in line 2, thus they still need to be counted once in line 3.

Permutations

Definition: A permutation of n distinct elements is an ordering of the n elements.

Let us find the number of permutations of n distinct elements $\{1,2,3,\cdots,n\}$. There are n positions, 1 through n.

1	Choose one of n possible elements for position 1.
2	Choose one of the remaining $n-1$ elements for position 2.
3	Choose one of the remaining $n-2$ elements for position 3.
$\vdots$	$\vdots$
$n-1$	Choose one of the remaining 2 elements for position $n-1$.
n	Place the only remaining element in position n.

By the multiplication principle, the total number of permutations is

$$\boxed{n*(n-1)*(n-2)*\cdots*2*1=n!}$$

Example: Enumerate all permutation of $\{A,B,C\}$.
There are $3!=3*2*1=6$ possible orderings.

$$ABC$$
$$ACB$$
$$BAC$$
$$BCA$$
$$CAB$$
$$CBA$$

■

Example: How many permutations of $\{A,B,C,D\}$ have 'AB' as a substring?
Since $'AB'$ must stay together, the substring is treated as one object. So, we need to find all permutations of 3 objects$\{AB,C,D\}$. The number of permutations is $3!=3*2=6$. Below is an enumeration of them.

$$\boldsymbol{AB}CD$$
$$\boldsymbol{AB}DC$$
$$C\boldsymbol{AB}D$$
$$D\boldsymbol{AB}C$$
$$CD\boldsymbol{AB}$$
$$DC\boldsymbol{AB}$$

■

Example: A girl volleyball team has four players: Genie, Ruth, Sally, and Susan. The coach wants to assign them to four distinct positions: Setter, Spiker, Server, and Bumper. How many ways are there for the assignment?

The number of possible assignments is

$$4! = 4 * 3 * 2 * 1 = 24$$

■

Example: Four people (Bill, Jill, Liam, and Pat) need to be seated around a round table. How many ways are there to order them? (You may consider the ordering in clockwise direction, starting with one particular person, say Bill.)

We may seat one particular person, say Bill, and then consider all orderings of the remaining 3 (in clockwise direction). Thus the number is:

$$3! = 3 * 2 * 1 = 6$$

Suppose Bill is seated as a reference. Then, in clockwise direction, to the left of Bill, one of the remaining 3 people may be seated, and then one of the remaining 2. Finally, the last remaining person will be seated in clockwise direction (to the right of Bill).

■

Permutations of Subsets

Next, we consider permutations of an r-element subset of n distinct elements, rather than permutations of all elements. That is, we pick a subset of r elements and consider all orderings of each subset.

Definition: An r-permutation of n distinct elements is an ordering of an r-element subset of n distinct elements. In other words, an r-permutation of n distinct elements is an **ordered subset** of size r.

To find all r-permutations of n distinct elements:

(a) Find all subsets of size r.
(b) For each r-element subset, find all $(r!)$ permutations of it.

The number of r-permutations of n distinct elements is denoted as $P(n, r)$.

Example: List all 2-permutations of 4 distinct elements $\{A, B, C, D\}$.

AB BA
AC CA
AD DA
BC CB
BD DB
CD DC

The total number is $4 * 3 = 12$, because any of the 4 elements may be selected for position 1, and then any of the remaining 3 elements may be selected for position 2.

■

In the above listing, all orderings of a given subset are grouped together in a sub-box. For example, the subset $\{A, B\}$ generates 2 orderings: AB and BA. In general, each subset of size r generates $(r!)$ orderings.

The number of r-permutations of n distinct elements is

$$P(n,r) = n\,(n-1)(n-2)\cdots(n-r+1) = \frac{n!}{(n-r)!}$$

Proof:

- For position 1, select any of the n distinct elements.
- For position 2, select any of the remaining $n-1$ elements.

  ⋮
- For position r, select any of the remaining $n-r+1$ elements.

By the multiplication principle, the total number is

$$n\,(n-1)(n-2)\cdots(n-r+1).$$

Example: A student-club of 12 persons wants to select a president, vice-president, and a secretary. What is the number of ways to do the selection? The number is

$$P(12,3) = 12 * 11 * 10 = 1320.$$

■

Example: A volleyball team has 4 players: Rebecca, Rachael, Sally, and Sue. The coach wants to assign one player for Setter and another for Spiker. What is the number of ways to make the assignment? Give the number and then list them.

The number of possible assignments is $P(4,2) = 4 * 3 = 12$. Below is the listing.

Setter	**Spiker**
Rebecca	Rachael
Rebecca	Sally
Rebecca	Sue
Rachael	Rebecca
Rachael	Sally
Rachael	Sue
Sally	Rebecca
Sally	Rachael
Sally	Sue
Sue	Rebecca
Sue	Rachael
Sue	Sally

Combinations

So far, we considered an r-permutation of n distinct elements, which is an **ordered** selection of size r. Next we consider the selection of **unordered subsets**, called c**ombination**.

Example: Enumerate all 2-combinations (i.e., all unordered subsets of size 2) out of 4 distinct elements $\{A, B, C, D\}$.

$\{A, B\}$
$\{A, C\}$
$\{A, D\}$
$\{B, C\}$
$\{B, D\}$
$\{C, D\}$

Definition: An r-combination of a set of n distinct elements is an **unordered** selection of a subset of r elements out of the set. The selection may also be called "combination of r out of n" or "choose r out of n".

The number of r-combinations of n distinct elements is denoted as $C(n, r)$ or $\binom{n}{r}$.

There is a simple relation between $P(n,r)$ and $C(n,r)$. For $P(n,r)$, each subset of size r generates $r!$ possible orderings. For $C(n,r)$, each subset of size r must be counted only once. So we get the following formula.

$$\boxed{C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!\,(n-r)!}}$$

Note there is symmetry in this count, so $C(n,r) = C(n,n-r)$. For example,

$$C(7,2) = C(7,5) = \frac{7!}{2!\,5!}$$

This should make an intuitive sense:

- $C(7,2)$ chooses 2 out of 7 to be included in the subset.
- $C(7,5)$ chooses 5 out of 7 to be excluded, thus leaving 2 for the subset.

Example: How many 8-bit integers have exactly four 1's and four 0's? One example is 11010001.

We need to choose a subset of 4 out of 8 for 1's. The number is

$$C(8,4) = \frac{8!}{4!\,(8-4)!} = \frac{8*7*6*5}{4*3*2*1} = \frac{1680}{24} = 70$$

■

Example: A committee of 7 people wants to select a chair, a vice-chair, and 2 secretaries. What is the total number of ways?

(a) Choose 1 out of 7 for chair. The number of ways is $C(7,1) = 7$.
(b) Choose 1 out of the remaining 6 for vice-chair. The number of ways is $C(6,1) = 6$.

Alternatively, steps (a) and (b) may be combined into one step:
Choose an ***ordered subset*** of 2 out of 7 for chair and vice-chair. The number is

$$P(7,2) = 7*6 = 42$$

The selection of 2 must be ordered because the two positions (chair and vice-chair) are different.

(c) Choose 2 out of the remaining 5 for secretaries. This is an ***unordered subset*** of 2 out of 5, and the number is

$$C(5,2) = \frac{5!}{2!\,3!} = \frac{5*4}{2} = 10$$

By the multiplication principle, the total number of ways is

$$P(7,2) * C(5,2) = 7 * 6 * 10 = 420.$$

Alternatively, we could start by picking two secretaries out of 7 (unordered), and then pick a chair and a vice-chair out of the remaining 5 (ordered). The total number becomes the same.

$$C(7,2) * P(5,2) = \frac{7*6}{2} * (5*4) = 21 * 20 = 420$$

The important thing to remember is that the selection of chair and vice-chair is ordered, because the two positions are different, but the selection of two secretaries is unordered because the two secretaries have equal status.

■

Example (Poker Hands): A deck of cards consists of 52 cards, with 4 *suits* (hearts, diamonds, clubs, spades) and 13 *kinds* (ace, king, queen, jack, 10, 9, 8, 7, 6, 5, 4, 3, 2). A poker hand is a drawing of 5 cards out of 52.

(a) What is the total number of possible poker hands?
(b) What is the number of poker hands with 3 cards out of one kind, and 2 cards out of another kind (3,2)? This is called a *full-house*. (An example is 3 aces and 2 kings.)
(c) What is the number of poker hands with 2 cards out of one kind, 2 cards out of another kind, and one card out of a third kind (2,2,1)? This is called *two-pairs.* An example is (2 aces, 2 queens, and 1 jack).

Solution:
(a) The total number of poker hands is

$$C(52,5) = \frac{52!}{5!\,(47!)} = \frac{52*51*50*49*48}{5*4*3*2} = 2{,}598{,}960$$

(b) The number of distinct full-house hands (3,2) is computed as follows.

(i) Pick one kind out of 13, and choose 3 cards out of that kind.

$$C(13,1) * C(4,3) = 13 * 4 = 52$$

(ii) Pick a second kind out of the remaining 12 kinds, and choose 2 cards from it.

$$C(12,1) * C(4,2) = 12 * \frac{4!}{2!\,(4-2)!} = 12 * \frac{4*3}{2} = 12 * 6 = 72$$

By the multiplication principle, the total number is:

$$52 * 72 = 3744$$

An alternative way of computing steps (i) and (ii) above is to pick an *ordered subset* of 2 kinds, pick 3 out of the first kind, and pick 2 out of the second kind. The total count becomes the same:

$$P(13,2) * C(4,3) * C(4,2) = 13 * 12 * 4 * 6 = 3744$$

(c) The number of distinct two-pair hands (2,2,1) is computed as follows.

(i) Choose an *unordered subset* of 2 kinds, and pick a pair out of each kind.

$$C(13,2) * C(4,2) * C(4,2) = \frac{13 * 12}{2} * \frac{4 * 3}{2} * \frac{4 * 3}{2} = 78 * 6 * 6 = 2808$$

(ii) Choose a third kind and pick one card out of that kind.

$$C(11,1) * C(4,1) = 11 * 4 = 44$$

By the multiplication principle, the total number is: $2808 * 44 = 123{,}552$.

Binomial Coefficients

An interesting application of combinations $C(n,r)$ is to derive the coefficients for the binomial expansion $(a+b)^n$. For example, consider the formula for $n = 3$.

$$(a+b)^3 = (a+b)(a+b)(a+b) = a^3 + 3a^2b + 3ab^2 + b^3$$

There are 3 factors $(a+b)$ to be multiplied. From each factor, either a or b is selected, then multiplied together to get a product term, and the product terms are added to get the final result. Since there are 2 choices from each factor, the total number of product terms is $2 * 2 * 2 = 8$. The following table shows all product terms, one per row.

	Select from $(a+b)$	Select from $(a+b)$	Select from $(a+b)$	Product
1	a	a	a	a^3
2	a	a	b	a^2b
3	a	b	a	a^2b
4	b	a	a	a^2b
5	a	b	b	ab^2
6	b	a	b	ab^2
7	b	b	a	ab^2
8	b	b	b	b^3

To get the product term a^3, all 3 factors must select a. (This means no factor selects b.) The number of ways for this selection is $C(3,0) = C(3,3) = 1$. (Note $0! = 1$.) One row gives the product a^3, and in the final result, the coefficient for a^3 is 1.

To get the product term a^2b, we must select a from 2 of the 3 factors. (This means select b from 1 of the 3 factors.) The number of ways for this selection is $C(3,1) = 3$. Three rows in the table (rows 2, 3, 4) give the product a^2b. Therefore, after adding the product terms, we get $3a^2b$.

Similarly, for the product term ab^2, we need to select a from 1 out of the 3 factors. (This means select b from 2 out of the 3 factors.) The number of ways is $C(3,2) = 3$. Three rows in the table (rows 5, 6, 7) give the product term ab^2. So after adding the product terms, we get $3ab^2$.

Finally, to get the product term b^3, all 3 factors must select b. The number of ways for this selection is $C(3,3) = 1$. So the coefficient for b^3 is 1. In summary,

$$\begin{aligned}(a+b)^3 &= C(3,0)*a^3 + C(3,1)*a^2b + C(3,2)*ab^2 + C(3,3)*b^3\\ &= 1*a^3 + 3*a^2b + 3*ab^2 + 1*b^3\end{aligned}$$

Note the coefficients are symmetrical. For example, the coefficients for a^2b and ab^2 are both equal to 3 because $C(3,1) = C(3,2) = 3$.

As another example, let us compute the coefficients for $(a+b)^4$.

$$\begin{aligned}(a+b)^4 &= C(4,0)*a^4 + C(4,1)*a^3b + C(4,2)*a^2b^2 + C(4,3)*ab^3 + C(4,4)*b^4\\ &= a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4\end{aligned}$$

In general,

$$\boxed{\begin{aligned}(a+b)^n &= C(n,0)a^n + C(n,1)a^{n-1}b + C(n,2)a^{n-2}b^2 + \cdots + C(n,n)b^n\\ &= \sum_{r=0}^{n} C(n,r)a^{n-r}b^r\end{aligned}}$$

Recursive Formulation of Combination

The formula for $C(n,r) = \frac{n!}{r!(n-r)!}$ may be formulated recursively in an interesting way. $C(n,r)$ is the number of ways to select an unordered subset of size r out of a set of n distinct elements $\{1,2,3,\cdots,n\}$. We consider two cases depending on whether element 1 is selected to be included in the subset or not. Let $0 < r < n$.

(a) If element 1 is in the subset, then we need to recursively select $r-1$ elements out of the remaining $n-1$ elements. The number of ways for this selection is
$$C(n-1, r-1)$$
(b) If element 1 is not in the subset, then we need to recursively select all r elements out of the remaining $n-1$ elements. The number of ways for this selection is
$$C(n-1, r)$$

These two sets of selections are disjoint. So, we apply the addition principle to get the total number for $C(n,r)$.

$$\boxed{C(n,r) = \begin{cases} C(n-1,r-1) + C(n-1,r), & 0 < r < n \\ 1, & r = 0 \text{ or } r = n \end{cases}}$$

The actual computation of $C(n,r)$ must not be done recursively, because recursion would introduce terrible inefficiency. (There is a great deal of overlap in recursive computation of the two terms $C(n-1,r-1)$ and $C(n-1,r)$.) The recursive formulation is used to compute $C(n,r)$ in a bottom-up approach: First compute $C(1,r)$ for all r, then compute $C(2,r)$ for all r, then $C(3,r)$, and so on. Below is a simple nested-loop to perform the bottom-up computation and store the results in an array $C[1:n, 0:n]$.

```
$C[1,0] = C[1,1] = 1;$
for $m = 2$ to $n$ {
  $C[m,0] = C[m,m] = 1;$
  for $r = 1$ to $m-1$
      $C[m,r] = C[m-1,r-1] + C[m-1,r];$
}
```

The following table illustrates the computation for $n = 6$. First, the row $C[2,r]$ is computed, then the row $C[3,r]$, then $C[4,r]$, then $C[5,r]$, and finally the row $C[6,r]$. For example, once row 5 has been computed, row 6 is computed by using the values from row 5. One such computation is highlighted: $C[6,3] = C[5,2] + C[5,3] = 10 + 10 = 20$.

	$r=0$	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$
$n=1$	1	1					
$n=2$	1	2	1				
$n=3$	1	3	3	1			
$n=4$	1	4	6	4	1		
$n=5$	1	5	10	10	5	1	
$n=6$	1	6	15	20	15	6	1

Note that the values in row n, which are $C[n,0], C[n,1], \cdots, C[n,n]$, are the binomial coefficients for $(a+b)^n$.

Pascal's Triangle

The above table may be drawn in the form of triangle, known as *Pascal's triangle*. The boundary values in each row of the triangle are all 1, and each interior value is computed as sum of the two values directly above it. For example, the 20 at the base is computed as 10+10, as highlighted.

$$\begin{array}{c}
1 \\
1 \quad 1 \\
1 \quad 2 \quad 1 \\
1 \quad 3 \quad 3 \quad 1 \\
1 \quad 4 \quad 6 \quad 4 \quad 1 \\
1 \quad 5 \quad 10 \quad 10 \quad 5 \quad 1 \\
1 \quad 6 \quad 15 \quad 20 \quad 15 \quad 6 \quad 1
\end{array}$$

Generalized Permutations and Combinations

We now discuss several generalizations of permutations/combinations, including the case when the elements are **not distinct**, and the case when we make selections with repetitions allowed.

Partitioning a set into a number of disjoint subsets

The combination $C(n,r)$ is the number of ways to choose an unordered subset of r elements out of n distinct elements. This problem may be generalized into partitioning a set of n distinct elements into a number of disjoint subsets.

Example: How many distinct strings are generated by all permutations of the following string of 8 characters, which consists of 2 A's, 3 B's, 2 C's, and 1 D?

$$AABBBCCD$$

If the 8 characters were all distinct, the number of distinct permutations would be 8!. But in this case, the number will be much smaller because there are several groups of identical letters. Given any ordering of the string, if we reorder the letters within each group, it will not produce a new string.

One way to find the number of distinct permutation is in terms of $C(n,r)$, as follows. Let the positions in the string be numbered 1 to 8.

(a) Pick 2 out of 8 positions for letter A. The number of ways is: $C(8,2)$.
(b) Pick 3 out of the remaining 6 positions for letter B. The number of ways is $C(6,3)$.
(c) Pick 2 out of the remaining 3 positions for letter C. The number of ways is $C(3,2)$.
(d) Use the remaining single position for letter D. The number of ways is 1.

By the multiplication principle, the total number is

$$C(8,2) * C(6,3) * C(3,2) = \frac{8!}{2!\,6!} * \frac{6!}{3!\,3!} * \frac{3!}{2!\,1!} = \frac{8!}{2!\,3!\,2!\,1!}$$

After cancelling the highlighted terms, the reduced result has 8! in the numerator, corresponding to the length of the string, and a factorial in the denominator for each group size (2,3,2,1).

■

There is also a way to derive the above formula directly. The number of permutations for a string of n *distinct* characters is $n!$. But if the characters are not distinct, then for each group of r identical characters, there are $r!$ permutations of the letters within that group which must be counted as only one. Thus, $n!$ must be divided by $r!$ for each group of r identical characters.

General Formula: Suppose we want to partition a set of n distinct elements into a number of disjoint subsets of sizes $n_1, n_2, \cdots, n_k$, where $n = \sum_{i=1}^{k} n_i$. The number of ways to do the partitioning is

$$\boxed{\frac{n!}{n_1!\,n_2!\cdots n_k!}}$$

Example: How many distinct strings are produced by all permutations of the following string?

$$MISSISSIPPI$$

There are 4 groups of identical letters: $1\ M, 4\ I, 4\ S,\ 2\ P$. The number of distinct strings is

$$\frac{11!}{1!\,4!\,4!\,2!} = 34{,}650$$

Selection with Repetitions (Introducing Walls/Dividers)

Given a set of n distinct objects $\{1,2,\cdots,n\}$, let us find the number of ways to pick an unordered selection of r elements out of the n objects, when **repetitions are allowed**. (That is, each object may be selected several times.) There is no restriction on the value of r compared to n.

Theorem: The number of ways to make an r-selection of n distinct objects (with repetitions allowed) is

$$\boxed{C(r+n-1,r) = C(r+n-1,n-1)}$$

Basically, **we need to assign a number to each object** to specify how many times that object is selected. Before explaining how the counts are incorporated, let us consider a specific small example with

$$n = 3, r = 4.$$

This means we have three distinct objects $\{1,2,3\}$ and we want to make a selection 4 times. According to the above formula, the number of ways to make the selection is

$$C(n+r-1,n-1) = C(6,2) = \frac{6*5}{2} = 15.$$

And in this case, since the problem is small enough, we can enumerate all cases as follows. (This, of course, is not proof of the above formula in general.)

Object	1	2	3
Number of times selected	0	0	4
	0	1	3
	0	2	2
	0	3	1
	0	4	0
	1	0	3
	1	1	2
	1	2	1
	1	3	0
	2	0	2
	2	1	1
	2	2	0
	3	0	1
	3	1	0
	4	0	0

This method of enumeration is not practical in general. So we need a way to prove the above formula with an eloquent combinatorial argument.

Proof of the Formula (n distinct objects, and r selection)

$$\boxed{C(r+n-1,r)=C(r+n-1,n-1)}$$

We will use a line-up of

$$r+n-1 \quad \text{Positions}$$

This line-up will consist of an arrangement of

r Tokens

$n-1$ Objects (Each object is also called a **divider**)

(They may appear in any order.)

Interpretation of the line-up: The number of tokens lined-up before each object will denote the number of times that object is selected.

Let us consider the above example again ($n=3, r=4$). The particular case highlighted in yellow in the table selects object 1 twice. object 2 once, and object 3 once. The line-up of tokens (shown as 0) and objects (shown as 1) for this case is shown below.

Position	1	2	3	4	5	6
	Token	Token	Divider 1 (Object 1)	Token	Divider 2 (Object 2)	Token
	0	0	1	0	1	0

In this line up,

- There are two tokens before the first divider, so object 1 is selected twice.
- There is one token before the second divider, so object 2 is selected once.
- There is one token **after** the second (and last) divider, so object 3 is selected once.

The proof of the general formula is now apparent. The number of positions is $r+n-1$. We need to choose $n-1$ positions to place the dividers. Therefore, the number of ways is as claimed before,

$$C(r+n-1,n-1)$$

■

Example: What is the number of ways to place 8 identical golf balls into three distinct holes (A, B, C)?

In terms of the above theorem, this is an 8-selection of 3 distinct objects $(n = 3, r = 8.)$ Use a line-up of 8 balls and 2 dividers, thus a total of 10 slots. Find the number of ways to choose 2 slots out of 10 for the dividers. The number of ways is:

$$C(n+r-1, n-1) = C(10,2) = \frac{10 * 9}{2} = 45$$

Example: How many ways are there to divide 8 identical slices of pizza among 3 friends, with no leftover, in each of the following cases?

(a) No restriction on how many slices each person gets
(b) Each person gets at least two slices
(c) Each person gets at least two slices and at most 3 slices

Solution:

(a) One way to compute the number of ways is by enumerating all possible number of slices for each person. This may be done by a nested summation, where the first person has a slices, the second person has b slices, and the third person has the leftover number $(8 - a - b)$.

$$\sum_{a=0}^{8}\sum_{b=0}^{8-a}(1) = \sum_{a=0}^{8}(8 - a + 1) = 9 + 8 + \cdots 1 = 9 * \frac{10}{2} = 45$$

This method is not easy to generalize to a larger number of friends.
A more general and eloquent approach is to use "dividers" as explained in the above theorem. In terms of the theorem, this is an 8-selection of 3 distinct elements. Since there are $n = 3$ friends, we use $n - 1 = 2$ dividers. We use a line-up of $8 + 2 = 10$ positions, where each position can hold either a slice of pizza (shown as O) or a divider (shown as |). All slices of pizza positioned before the first divider goes to the first person (A), all slices between the two dividers goes to person B, and all slices after the second divider goes to the third person (C).
In the following illustration, the two dividers are in positions 3 and 7. So 2 slices goes to person A, 3 slices goes to person B, and 3 slices to person C.

Position	1	2	3	4	5	6	7	8	9	10
Pitzza/Divider	O	O	|	O	O	O	|	O	O	O

So we need the number of ways to pick 2 positions out of 10 for dividers, which is

$$C(10,2) = \frac{10 * 9}{2} = 45$$

(b) After giving each person 2 slices, there are 2 slices remaining to be divided among three friends with no restriction. Again use 2 dividers and 2 slices. So the problem is the number of ways to pick 2 out of 4 positions for the dividers, which is

$$\boxed{C(4,2) = 6}$$

Below is an explicit listing of all 6 possibilities for dividing the remaining 2 slices between 3 friends, with no restriction on how many slices each person gets.

A	B	C
0	0	2
0	1	1
0	2	0
1	0	1
1	1	0
2	0	0

(c) Give each person a minimum of 2 slices. Then, there are 2 slices remaining to be divided among 3 friends, with each person getting at most one. So pick 2 out of 3 friends to get one slice each. The number of ways is: $C(3,2) = 3$.

■

Example: What is the number of 3-digit decimal integers with the sum of the 3 digits equal 8? One example is 215.

Solution: We use the idea of dividers used in the previous problem, in conjunction with the "unary representation" of a decimal number. For example, 215 is represented as:

$$11\_1\_11111$$

A decimal digit of 2, for example, is shown as 11 (which is two 1's), and dividers are used in between digits.

Position	1	2	3	4	5	6	7	8	9	10
1 or divider	1	1	_	1	_	1	1	1	1	1

So the problem becomes a line-up of 10 positions (8 1's plus 2 dividers). And we need the number of ways to pick 2 out of 10 positions for the two dividers.

$$C(10,2) = \frac{10 * 9}{2 * 1} = 45.$$

■

Module 10: Introduction to Discrete Probability

Reading from the Textbook: Chapter 6 Counting Methods

Introduction

We present a brief and informal treatment of discrete probability in this module. The counting methods of the previous module are applied for this development. We start with the basic sum rule and product rule, and then study conditional probability. Finally, we apply the counting methods and discrete probability for games such as poker and lottery.

Contents

Basic Definitions

An **experiment** is a process that produces an **outcome**. For example, an experiment may be the roll of a six-sided die, which has 6 possible outcomes.

The set of possible outcomes is called **sample space.**

An **event** is an outcome or a set of outcomes. For example, the event that the die is ≥ 5 consists of two outcomes $\{5, 6\}$.

For a roll of a die, if we assume that all 6 possible outcomes are equally likely, then they each have a probability of 1/6, and the sum of the probabilities for all possible outcomes equals to 1.

If E is an event from a finite sample space S, and if all outcomes have equal probability, then the probability of event E is the ratio of the number of outcomes in event E over number of outcomes in sample space S.

$$\boxed{P(E) = \frac{|E|}{|S|}}$$

For example, consider the event that a roll of a die produces a face ≥ 5. Since $E = \{5,6\}$, and $S = \{1,2,3,4,5,6\}$, this event has the probability $2/6$.

Another immediate rule applies to complements:

$$\boxed{P(E) + P(\bar{E}) = 1}$$

Two events E_1 and E_2 are **disjoint**, or **mutually exclusive**, if $E_1 \cap E_2 = \emptyset$.

Two events E_1 and E_2 are **independent** if the outcome on one event does not change the probability for the other event.

Given two sets of events E_1 and E_2,

- $(E_1 \cup E_2)$ may also be viewed as the event (E_1 **or** E_2),
- $(E_1 \cap E_2)$ may also be viewed as the event (E_1 **and** E_2).

Sum Rule

Recall the addition principle for disjoint sets E_1 and E_2, which is $|E_1 \cup E_2| = |E_1| + |E_2|$. We may express a similar sum-of-probability rule for two disjoint events.

Sum Rule: If two events E_1 and E_2 are disjoint (also called mutually exclusive), then

$$P(E_1 \cup E_2) = P(E_1) + P(E_2)$$

Note: $(E_1 \cup E_2)$ may also be interpreted as $(E_1 \text{ or } E_2)$.

Example: Consider the roll of a pair of 6-sided dice. What is the probability that the sum is 6 or 7? The sample space consists of $6 \times 6 = 36$ outcomes. Let E_1 be the event that the sum equals 6, and $E_2$the event that the sum equals 7.

$$E_1 = \{(1,5), (2,4), (3,3), (4,2), (5,1)\},$$

$$E_2 = \{(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)\}$$

So, $P(E_1) = 5/36$ and $P(E_2) = 6/36$. Therefore,

$$P(E_1 \cup E_2) = \frac{5}{36} + \frac{6}{36} = \frac{11}{36}$$

■

If the two events are not disjoint, then following the addition principle, the following obvious rule applies.

$$\boxed{P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)}$$

Example: A pair of dice is rolled. What is the probability that one die is 5? (This includes the case when both are 5.)

$$P(D_1 = 5 \vee D_2 = 5) = P(D_1 = 5) + P(D_2 = 5) - P(D_1 = 5 \wedge D_2 = 5) = \frac{1}{6} + \frac{1}{6} - \frac{1}{36} = \frac{11}{36}$$

Another way to compute this probability is as follows. First observe that the probability that neither die is 5 becomes $25/36$, because each die has 5 possible values. So,

$$P(D_1 = 5 \vee D_2 = 5) = 1 - P(D_1 \neq 5 \wedge D_2 \neq 5) = 1 - \frac{25}{36} = \frac{11}{36}$$

This probability may also be seen by enumerating the entire sample space, as below. Note that (5,5) is removed in the second set of 6, because it is repeated, thus leaving 11 outcomes in the sample space.

D_1D_2
51
52
53
54
55
56
15
25
35
45
~~55~~
65

■

Example: A coin is flipped 4 times. What is the probability of exactly two heads (and two tails)? The sample space has $2^4 = 16$ possible outcomes. The number of outcomes with exactly two heads equals the number of subsets of size 2 out of 4, which is $C(4,2) = 6$. So the probability of exactly 2 heads is

$$\frac{C(4,2)}{16} = \frac{6}{16} = \frac{3}{8}$$

We may also list all 16 outcomes to see that 6 of them have exactly two heads.

Row	$F_1F_2F_3F_4$	Number of Heads
0	T T T T	0
1	T T T H	1
2	T T H T	1
3	T T H H	2
4	T H T T	1
5	T H T H	2
6	T H H T	2
7	T H H H	3
8	H T T T	1
9	H T T H	2
10	H T H T	2
11	H T H H	3
12	H H T T	2
13	H H T H	3
14	H H H T	3
15	H H H H	4

Let us compare the above probability with the probability of exactly k heads, for different values of k.

k	$C(4,k)$	Probability of k heads
0	1	1/16
1	4	4/16
2	6	6/16
3	4	4/16
4	1	1/16

The probability is the highest for $k = 2$, as expected. That is, if we flip a coin n times, the probability of exactly k heads is the highest for $k = n/2$. This means that it is most likely that half of the outcomes are head and half tail, which basically follows from the definition of probability. ■

Product Rule

Consider a random flip of a coin. It has two outcomes: head and tail, each with the probability 1/2. Now consider two flips of a coin. The outcome of the first flip does not change the probability for the second flip. So we say the two events are independent.

What is the probability that both flips are head? There are 4 possible outcomes in the sample space. (Each flip has 2 possible outcomes. So, by the multiplication principle, the number of outcomes for two flips is $2 \times 2 = 4$.) The 4 outcomes have equal probability 1/4. So, the probability that a head occurs on both flips is 1/4.

Flip1	Flip2
H	H
H	T
T	H
T	T

Another way is by using the product rule for independent events. Let H_1 be the event that a head occurs on the first flip, and H_2 the event that a head occurs on the second flip.

$$P(H_1 \; and \; H_2) = P(H_1) * P(H_2) = \frac{1}{2} * \frac{1}{2} = \frac{1}{4}$$

Let us state the rule formally.

Product Rule: Given two independent events E_1 and E_2,

$$P(E_1 \cap E_2) = P(E_1) * P(E_2)$$

Note: $E_1 \cap E_2$ may also be interpreted as $E_1 and\ E_2$.

Example: Consider an experiment where we keep flipping a coin until we get a head.

(a) What is the probability that the first head occurs on the 3rd flip?
(b) What is the probability that the first head occurs on the k^{th} flip?
(c) What is the **expected** number of flips before the first head occurs?

Solution:

(a) Probability of getting the first head on the 3rd flip is found by using the product rule.

$$P(T_1 \wedge T_2 \wedge H_3) = P(T_1) * P(T_2) * P(H_3) = \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{8}.$$

(b) Probability of getting the first head on the k^{th} flip is similarly found by the product rule.

$$P(\text{first head occurs on } k^{th} \text{ flip}) = P(T_1 \wedge T_2 \wedge \cdots \wedge T_{k-1} \wedge H_k) = \left(\frac{1}{2}\right)^k = \frac{1}{2^k}$$

(c) The expected number of flips before getting the first head is

$$\sum_{k=1}^{\infty} k * P(\text{first head occurs on } k^{th} \text{ flip}) = \sum_{k=1}^{\infty} \frac{k}{2^k}$$

The above summation has a value of 2. To prove this sum value, let us first consider the sum for the finite series. We leave it to the student to prove by induction the following formula.

$$S(n) = \sum_{k=1}^{n} \frac{k}{2^k} = 2 - \frac{n+2}{2^n}$$

Then, it is immediate that

$$\lim_{n \to \infty} S(n) = 2.$$

This result (expected value of 2) also follows directly from the definition of probability. Since the probability of a head on a single flip is 1/2, then it follows that on the average, it takes two flips before getting the first head.

■

Conditional Probability

We now consider the case when the outcome of one event effects the probability of another event. That is, when the events are not independent. As a very simple example, suppose a pair of dice is rolled. The probability that the sum is 11 is $2/36$, since there are two outcomes {56, 65} with the sum $= 11$. But suppose we already know the first die is 6. This knowledge changes the probability to $1/6$ because the second die has to be 5, which is one of six possible outcomes. The following example will further motivate the concept of conditional probability, and the formula for it.

Example: A pair of dice is rolled. Suppose we know that at least one die is 6. Compute the probability that the sum of the two is ≥ 9, **given that** one or both die is 6. This is called **conditional probability,** and it is written as:

$$P(sum \geq 9 \mid one\ die\ is\ 6).$$

(The vertical line | is read as "given that".)

Solution: The original unconditional sample space, S, has 36 outcomes. The sum value for each outcome is shown below. Since we know that one die is 6, the sample space becomes limited to only the last row and last column, as shaded, which has 11 outcomes. Let us call this the conditional sample space S'.

	$D_2 = 1$	$D_2 = 2$	$D_2 = 3$	$D_2 = 4$	$D_2 = 5$	$D_2 = 6$
$D_1 = 1$	2	3	4	5	6	7
$D_1 = 2$	3	4	5	6	7	8
$D_1 = 3$	4	5	6	7	8	9
$D_1 = 4$	5	6	7	8	9	10
$D_1 = 5$	6	7	8	9	10	11
$D_1 = 6$	7	8	9	10	11	12

There are 7 outcomes (out of 11) in sample space S' with the sum ≥ 9, as highlighted in blue. Therefore, the conditional probability of this event is the ratio $7/11$.

$$P(sum \geq 9 \mid \text{one die is } 6) = \frac{7}{11}$$

Now, observe that the 7 highlighted outcomes in the original unconditional sample space S constitute the event

$$(sum \geq 9) \cap (\text{one die is } 6)$$

This event has the probability of 7/36. Therefore,

$$P(sum \geq 9 \mid \text{one die is } 6) = \frac{P((sum \geq 9) \cap (\text{one die is } 6))}{P(\text{one die is } 6)} = \frac{7/36}{11/36} = \frac{7}{11}$$

■

The above example motivates the formula for the general case.

Conditional Probability: The probability of event A, given that event B has occurred, is

$$P(A\,|B) = \frac{P(A \cap B)}{P(B)}$$

Therefore,

$$\begin{aligned} P(A \cap B) &= P(A\,|B) * P(B) \\ &= P(B \mid A) * P(A) \end{aligned}$$

If events A and B are independent, then $P(A\,|B) = P(A),$ and the above formula becomes

$$P(A \cap B) = P(A) * P(B)$$

which is the product rule stated earlier for independent events.

Example: A pair of dice is rolled. Compute the probability of each event.

(a) Sum ≥ 8.
(b) One die is 6. (This includes the case when both are 6.)
(c) Sum ≥ 8 and one die is 6.
(d) Sum ≥ 8, given that one die is 6. (Use the formula for conditional probability)
(e) One die is 6, given that Sum ≥ 8. (Use the formula for conditional probability)

Solution:

(a) Sum ≥ 8: The following table shows all 36 outcomes in the sample space.

	$D_2 = 1$	$D_2 = 2$	$D_2 = 3$	$D_2 = 4$	$D_2 = 5$	$D_2 = 6$
$D_1 = 1$	2	3	4	5	6	7
$D_1 = 2$	3	4	5	6	7	8
$D_1 = 3$	4	5	6	7	8	9
$D_1 = 4$	5	6	7	8	9	10
$D_1 = 5$	6	7	8	9	10	11
$D_1 = 6$	7	8	9	10	11	12

There are 15 outcomes with the sum ≥ 8, as shaded in pink. (There are 5 outcomes with the sum equal 8, four outcomes with the sum 9, three outcomes with sum 10, two outcomes with sum 11, and 1 with sum 12.) So,

$$P(Sum \geq 8) = \frac{5+4+3+2+1}{36} = \frac{15}{36}$$

(b) One die is 6: There are 11 outcomes in this event (the last row and last column).

$$P(\text{one die is } 6) = \frac{11}{36}$$

(c) Sum ≥ 8 and one die is 6: There are 9 outcomes in this event, as highlighted in blue.

$$P(\text{Sum } \geq 8 \text{ and one die is } 6) = \frac{9}{36}$$

(d) Sum ≥ 8, given that one die is 6.

$$P(\text{Sum } \geq 8 \mid \text{one die is } 6) = \frac{P(\text{Sum } \geq 8 \text{ and one die is } 6)}{P(\text{one die is } 6)} = \frac{9/36}{11/36} = \frac{9}{11}$$

(e) One die is 6, given that Sum ≥ 8.

$$P(\text{One die is } 6 \mid \text{Sum } \geq 8) = \frac{P(\text{Sum } \geq 8 \text{ and one die is } 6)}{P(Sum \geq 8)} = \frac{9/36}{15/36} = \frac{9}{15}$$

■

Example (Conditional probability of full house): A poker hand is a drawing of 5 cards out of a deck of 52, and *full-house* is a poker hand with 3 cards out of one kind, and 2 cards out of another kind. Compute the probability of each of the following.

(a) A full-house (unconditional case)
(b) A full-house, given that the first two face-up cards are 2 aces. (The two aces may be any two suits. Or, we may assume two specific suits such as hearts and diamonds.)
(c) A full-house, given that the first two cards are (ace of hearts & king of diamonds).

Solution:

(a) The size of sample space for all poker hands is

$$C(52,5) = \frac{52!}{5!\,(47!)} = \frac{52*51*50*49*48}{5*4*3*2*1} = 2{,}598{,}960$$

The number of possible full-house hands is

$$P(13,2) * C(4,3) * C(4,2) = 13*12*4*6 = 3744$$

Therefore,

$$P(\text{full house}) = \frac{3744}{2{,}598{,}960} \cong \frac{1}{694}$$

(It is common to express the probability as a ratio of 1 over the nearest integer.)

(b) Now let us compute the conditional probability. The size of the sample space for the conditional case is the number of ways to pick 3 more cards out of 50.

$$C(50,3) = \frac{50*49*48}{3*2*1} = 19{,}600$$

After the initial 2 aces, there are two ways to make a full-house:

- Get one more ace, and two cards out of another kind. The number of ways is
$$C(2,1) * C(12,1) * C(4,2) = 2*12*6 = 144$$
- Get 3 cards out of another kind. The number of ways is
$$C(12,1) * C(4,3) = 12*4 = 48$$

By the addition principle, the total number of ways is: $144 + 48 = 192$. So, the conditional probability becomes:

$$P(\text{full house} \mid \text{first 2 aces}) = \frac{192}{19600} \cong \frac{1}{102}$$

Using the Formula for Conditional-Probability

Now, let us find the above conditional probability by using the formula. First, we assume the first two aces may be any two suits.

$$P(\text{full house} \mid \text{first 2 aces}) = \frac{P(\text{first 2 aces} \cap \text{full house})}{P(\text{first 2 aces})}$$

$$P(\text{first 2 aces}) = \frac{C(4{,}2)}{C(52{,}2)}$$

$$P(\text{first 2 aces } \cap \text{full house}) = \frac{C(4{,}2) * 192}{C(52{,}2) * C50{,}3)}$$

The numerator in the latter formula is the number of ways to start with two aces of any two suits, times the number of ways to finish with a full house after the initial 2 aces. And the denominator is the size of the sample space because the initial 2 cards and the latter 3 cards are two separate groups. Therefore,

$$P(\text{full house} \mid \text{first 2 aces}) = \frac{[C(4{,}2) * 192]/[C(52{,}2) * C(50{,}3)]}{C(4{,}2)/C(52{,}2)} = \frac{192}{C(50{,}3)} \cong \frac{1}{102}$$

Now, suppose we assume the two aces are two specific suits such as hearts and diamonds. Then the term $C(4{,}2)$ in both places in the above computation is replaced by one, since there is one way of getting two aces of specific suits. But the final conditional probability still comes out the same. This shows that the suits of the initial two aces do not matter.

(c) A full-house, given that the first two cards are (ace of hearts, king of diamonds). There are two ways to make a full-house after these two cards:

- Get two more aces and one more king, or
- Get two more kings and one more ace.

The number of ways for each of these is: $C(3{,}2) * C(3{,}1) = 3 * 3 = 9$. By the addition principle the total number of ways is: $9 + 9 = 18$. So, the conditional probability is

$$P(\text{full house} \mid \text{first 2 are ace of hearts \& king of diamonds}) = \frac{18}{C(50{,}3)} \cong \frac{1}{1089}$$

It is interesting to compare these three probabilities. The unconditional probability of a full-house is $1/694$. In case (b), when we know the first two cards are two aces, it becomes much more likely to end up with a full-house. (The conditional probability becomes $1/102$.) But in the last case, when the first two cards are an ace & king, it is much less likely to get a full-house, as the probability becomes $1/1089$. ■

Example (Detecting the HIV Virus): This example is an interesting application of conditional probability formula.

Suppose 10% of patients in a particular clinic have the HIV virus. A test is used to detect the HIV virus in patients. The result of this test is *positive* for 98% of patients who have the HIV virus. Unfortunately, this test is also positive on about 1% of patients who do not have the HIV virus. What is the probability that a patient has the HIV virus if the test is positive?

Solution: Let H denote the set of patients who have the HIV virus (and $\overline{H}$ those who do not have the virus). And let T denote the set of patients who test positive. Then

$$P(H) = 0.10, \qquad P(\overline{H}) = 0.90, \qquad P(T|H) = 0.98, \qquad P(T|\overline{H}) = 0.01$$

And the probability that a patient has the HIV virus if he/she tests positive is $P(H|T)$. First, use the conditional probability formula to compute

$$P(T \cap H) = P(T|H) * P(H) = 0.98 * 0.10 = 0.098$$
$$P(T \cap \overline{H}) = P(T|\overline{H}) * P(\overline{H}) = 0.01 * 0.90 = 0.009$$

Now, $T = (T \cap H) \cup (T \cap \overline{H})$. Since $(T \cap H)$ and $(T \cap \overline{H})$ are mutually exclusive (disjoint),

$$\begin{aligned} P(T) &= P\big((T \cap H) \cup (T \cap \overline{H})\big) \\ &= P(T \cap H) + P(T \cap \overline{H}) \\ &= 0.098 + 0.009 = 0.107 \end{aligned}$$

Now compute $P(H|T)$.

$$P(H|T) = \frac{P(H \cap T)}{P(T)} = \frac{0.098}{0.107} = 0.9159$$

■

The above example is a special case of Bayes' Theorem, as stated below.

Bayes' Theorem: Let: $C_1, C_2, \cdots, C_n$ be n mutually exclusive classes, and let F be a "feature set" used to classify each object into one of the classes. (Each object is classified into the class which has the highest conditional probability.)

$$P(C_j|F) = \frac{P(F \cap C_j)}{\sum_{i=1}^{n} P(F \cap C_i)} = \frac{P(F|C_j)\, P(C_j)}{\sum_{i=1}^{n} P(F|C_i)\, P(C_i)}$$

Proof: Since every object is classified into one class, $U = (C_1 \cup C_2 \cup \cdots \cup C_n)$. Thus, $F = F \cap U = F \cap (C_1 \cup C_2 \cup \cdots \cup C_n) = (F \cap C_1) \cup (F \cap C_2) \cup \cdots \cup (F \cap C_n)$. So, the above denominator becomes $P(F)$, and the equation becomes the correct formula for conditional probability.

Example: For the above example of HIV virus detection, find the probability that a patient has the HIV virus if he/she does not test positive, $P(H|\bar{T})$.

Solution:

$$P(H|\bar{T}) = \frac{P(\bar{T} \cap H)}{P(\bar{T})} = \frac{P(\bar{T} \cap H)}{P(\bar{T} \cap H) + P(\bar{T} \cap \bar{H})} = \frac{P(\bar{T}|H)P(H)}{P(\bar{T}|H)P(H) + P(\bar{T}|\bar{H})P(\bar{H})}$$

$$= \frac{(1-0.98)(0.1)}{(1-0.98)(0.1) + (1-0.01)(0.90)} = \frac{0.002}{0.002+0.891} = \frac{0.002}{0.893} = 0.00224$$

■

Example (Black Friday Sale): The day after Thanksgiving, Best Buy has a TV sale, and has two salespersons working, Mary and Larry. Mary makes 75% of total sale, and Larry 25%. On the following Monday, 30% of all sale made by Mary are returned, and 10% of sale made by Larry are returned. (This is summarized in the following table.)

	Mary	Larry
Percent Sale	75	25
Percent Returns	30	10

Let M denote the event that Mary made the sale, and L the event that Larry made the sale. And let R denote the event that a sale item is returned. Find the probability that Mary made the sale on any returned item, $P(M|R)$.

Solution:

$$P(M) = 0.75, \qquad P(L) = 0.25, \qquad P(R|M) = 0.30, \qquad P(R|L) = 0.10$$

Recall $R = R \cap U = R \cap (M \cup L) = (R \cap M) \cup (R \cap L)$. So,

$$P(R) = P(R \cap M) + P(R \cap L)$$

Now,

$$P(M|R) = \frac{P(R \cap M)}{P(R)} = \frac{P(R|M)P(M)}{P(R|M)P(M) + P(R|L)P(L)}$$

$$= \frac{0.30 * 0.75}{0.30 * 0.75 + 0.10 * 0.25} = \frac{0.225}{0.225 + 0.025} = \frac{0.225}{0.250} = 0.90$$

■

Lottery

New Jersey Mega Millions Lottery is played as follows. Each ticket consists of two sets of numbers: (A picture is attached.)

- Upper Set: Numbers 1 through 75.
  You pick 5 distinct numbers in this group. These numbers are called White Balls.
- Lower Set: Numbers 1 through 15.
  You pick 1 number (called Mega Ball) in this group.

On the day of the drawing, the winning numbers are announced (5 White-Ball numbers, and one Mega Ball number). The prizes are listed below, together with the odds of winning each prize.

Match	**Prize**	**Odds 1:x**
Match 5 white-balls, and Mega Ball	Jackpot	258,890,850
Match 5 white balls	$1,000,000	18,492,204
Match 4 White Balls, and Mega Ball	$5000	739,688
Match 4 White Balls	$500	52,835
Match 3 White Balls, and Mega Ball	$50	10,720
Match 3 White Balls	$5	766
Match 2 White Balls, and Mega Ball	$5	473
Match 1 White Balls, and Mega Ball	$2	56
Match 0 White Balls, and Mega Ball	$1	21

We will show the computation for a few of these and leave the rest for the student as exercise.

Sample Space: The size of the sample space is the total number of distinct tickets.

$$U = C(75,5) * C(15,1) = \frac{75 * 74 * 73 * 72 * 71}{5 * 4 * 3 * 2 * 1} * 15 = 258{,}890{,}850$$

Match 5 white-balls and Mega Ball (Jackpot):

To win the jackpot, your white-ball numbers must match all 5 of the winning white-balls, and your mega-ball number must match the winning mega-ball. There is only one possible way for this to happen.

$$W = C(5,5) * C(1,1) = 1$$

So the probability of winning the jackpot is:

$$P = \frac{W}{U} = \frac{1}{258{,}890{,}850}$$

Match 3 White Balls, and Mega Ball:

The drawing announces 5 winning white balls, and the remaining $(75 - 5)$ are non-winning white-ball numbers. Similarly, there is one winning mega ball, and the remaining $(15 - 1)$ non-winning mega ball numbers. To win this prize category, your ticket must have 3 out of 5 winning white balls, and 2 out of 70 non-winning white balls. And your mega-ball number must match the winning mega-ball. The number of ways is

$$W = C(5,3) * C(70,2) * C(1,1) = \frac{5 * 4}{2 * 1} * \frac{70 * 69}{2 * 1} * 1 = 10 * 2415 = 24{,}150$$

The probability is

$$P = \frac{W}{U} = \frac{24{,}150}{258{,}890{,}850} \cong \frac{1}{10{,}720}$$

Match 3 White Balls:

To win this prize, your white balls must match 3 out of 5 winning white-balls, and 2 out of 70 non-winning white-balls. And your mega-ball must be 1 of the 14 non-winning mega-balls. Therefore,

$$W = C(5,3) * C(70,2) * C(14,1) = 10 * 2415 * 14 = 338{,}100$$

$$P = \frac{W}{U} = \frac{338{,}100}{258{,}890{,}850} \cong \frac{1}{766}$$

■

Poker Hands

We have looked at some of poker hands in our earlier examples. Here, we list all poker hands in ranked order. We show the computation of the probabilities for a select few, and leave the rest for the student as exercise.

A poker hand is a random drawing of 5 cards out of a deck of 52 cards (with no jokers). There are 4 *suits,* and 13 *ordered kinds* (ace, king, queen, jack, 10, 9, 8, 7, 6, 5, 4, 3, 2). An ace may be counted as either the highest (above king) or lowest (as 1).

Below is the list of all poker hands in ranked order. Each hand is defined so that it does not include any of the higher hands. For example, a *straight* poker hand is five cards in sequence, but not all in the same suite. (Otherwise, it would become straight flush.)
For the list of poker hands with illustrations, please see:

https://en.wikipedia.org/wiki/List_of_poker_hands

	Poker Hand	**# Ways (W)**	**Prob. (P)**
1	Straight Flush Five cards of same suit and in consecutive order	40	$1/64974$
2	Four-of-a-Kind (4,1) Four of one kind, and 1 of another kind	624	$1/4165$
3	Full House (3,2) 3 of one kind, and 2 of another kind	3,744	$1/694$
4	Flush Five cards in the same suit, but not all in sequence	5,108	$1/509$
5	Straight Five cards all in sequence, but not all in same suit	10,200	$1/255$
6	Three-of-a-Kind (3,1,1) 3 of one kind, 1 of another kind, and 1 of a third kind	54,912	$1/47$
7	Two-Pairs (2,2,1) 2 of one kind, 2 of another kind, and 1 of a third kind	123,552	$1/21$
8	One Pair (2,1,1,1) Two of one kind, and one from each of 3 other kinds	1,098,240	$1/2.36$
9	High Card (None of the above hands) Five different kinds, not all same suit, and not all in order	1,302,540	$1/1.99$
	Total	2,598,960	1.00

The number of ways for each poker hand is listed in the table, and the sum of these values equals the total sample size, $C(52,5)$. This provides a good check that the computations are done correctly. But the probabilities have been mostly truncated to 1/x, where x is the nearest integer. For this reason, sum of the probabilities cannot be used as a check. (They add up to $1.00 \cdots$.)

The sample space size is

$$U = C(52,5) = \frac{52 * 51 * 50 * 49 * 48}{5 * 4 * 3 * 2 * 1} = 2{,}598{,}960$$

Below is the computation for a few of the hands. (Earlier we showed the computation for full-house and two-pairs.)

Straight Flush (SF):

Since five cards must be in sequence, there is 10 possible values (kinds) for the highest card in the hand {ace, king, queen, jack, 10, 9, 8, 7, 6, 5}. For example, if the high card is jack, the hand is (jack, 10, 9, 8, 7). If the high card is 5, the hand is (5, 4, 3, 2, ace). But the high card cannot be any of {4, 3 , 2}. And the five cards must be all the same suit, so there are 4 possible suits. Thus, the total number of ways W and the probability P for a straight flush is

$$W(SF) = C(10,1) * C(4,1) = 10 * 4 = 40$$

$$P(SF) = \frac{W}{U} = \frac{40}{2{,}598{,}960} \cong \frac{1}{64{,}974}$$

Flush:

Five cards must be the same suit, and five different kinds but not in sequence. This means the five cards can be any 5 kinds, except for the 10 that makes straight. So,

$$W(flush) = C(4,1) * [C(13,5) - 10] = 4 * \left[\frac{13 * 12 * 11 * 10 * 9}{5 * 4 * 3 * 2 * 1} - 10\right] = 5108$$

$$P(flush) = \frac{5108}{2{,}598{,}960} \cong \frac{1}{509}$$

One Pair:

Pick 1 out of 13 kinds, and choose a pair out of it. Then make an *unordered* selection of 3 other kinds, and pick 1 card out of each kind.

$$W(1\,pair) = C(13,1) * C(4,2) * C(12,3) * C(4,1) * C(4,1) * C(4,1)$$

$$= 13 * 6 * \frac{12 * 11 * 10}{3 * 2 * 1} * 4 * 4 * 4 = 1{,}098{,}240$$

$$P(1\,pair) = \frac{1{,}098{,}240}{2{,}598{,}960} \cong \frac{1}{2.36}$$

We leave the computation of the remaining hands to the student as an exercise.

■

Module 11: Introduction to Number Theory and Cryptography

Introduction

The Internet is in ever-increasing use for a variety of applications that involve sensitive data and demand secure communication. In this module, we study some of the basic number theoretic results and then apply it to introduce cryptography.

Contents

1. Prime Numbers

Definitions:

- Given positive integers a and b, we say a ***divides*** b, or a is a ***divisor*** of b, and denote it as
$$a|b$$
if there is a positive integer q such that $b = qa$. We also say b is ***divisible*** by a.

- Given positive integers $a, b,$ and d. If $d|a$ and $d|b$, we say d is a ***common divisor*** of a and b.

Theorem 1: If $d|a$ and $d|b$, then for any integers s and t,

$$d \mid (sa + tb)$$

We call $(sa + tb)$ a ***linear sum***, or ***linear combination***, of a and b.

Proof: Since $d|a$ and $d|b$, then $a = pd$ and $b = qd$ for some integers p and q. So

$$sa + tb = spd + tqd = (sp + tq)d$$

Since $(sp + tq)$ is an integer, this means $d \mid (sa + tb)$. ■

Definition: A positive integer greater than 1 is ***prime*** if it is divisible only by 1 and itself.

For example, the first few prime numbers are: 2, 3, 5, 7, 11, 13, 17, 19, 23, and 29.

An integer greater than 1 that is not prime is called ***composite***. For example, 514250 is a composite number and it may be written as product of its prime factors.

$$514250 = 2 * 5^3 * 11^2 * 17$$

A positive integer $n > 1$ has a ***unique prime factorization***

$$n = p_1^{t_1}\, p_2^{t_2} \cdots p_k^{t_k}$$

for some integer k, where $p_1 < p_2 < \cdots < p_k$ are prime factors (divisors) and $t_1, t_2, \cdots, t_k$ are positive integer exponents.

If integer n is not too large, then it is fairly simple to find its prime factors.

Lemma 1: A composite integer $n > 1$, has a prime factor $p \le \sqrt{n}$.
Proof: Suppose to the contrary, the smallest prime factor is $p > \sqrt{n}$. Then, $n = pq$, for some integer $q < \sqrt{n}$. This means n has a prime factor $< \sqrt{n}$, which is a contradiction. Therefore, the smallest prime factor is $p \le \sqrt{n}$. ■

The following algorithm determines if an integer $n > 1$ is composite, and if so finds the smallest prime factor of n.

```
int Factor (int $n$) {
for $p = 1$ to $\sqrt{n}$
  if ($n \bmod p == 0$) return $p$;        \\ prime factor
return 0 }                                 \\ n is prime
```

Note that the first integer found by the algorithm is the smallest prime factor. (If p is composite, the algorithm would find its prime factors before it gets to p.)

The time complexity of the above algorithm is $O(\sqrt{n})$. The algorithm is efficient if n is not very large. But if n is very large, then it becomes very difficult to find the factorization.

Later, we study a cryptosystem known as RSA. This system is based on an extremely large composite n, relying on the fact that it would be nearly impossible to "break the code" by decomposing n. Suppose n is about 100 decimal digits. Then,

$$\sqrt{n} \cong 10^{50}$$

To appreciate how large this exponential time becomes, suppose an iteration of the loop in the above algorithm takes 10 nanoseconds. Then, the running time of the algorithm is 10^{42} seconds. The number of seconds in one year is $365 * 24 * 3600 \cong 3.15 * 10^7$. So the running time becomes about $3 * 10^{34}$ years! In other words, exponential time means forever!

2. Greatest Common Divisors (GCD)

The ***greatest common divisor*** of two positive integers a and b, denoted as gcd(a, b), is the largest number that divides both of them. Note that $\gcd(a, 0) = a$. Integers a and b are said to be ***relatively prime*** if they have no common divisor > 1, so gcd$(a, b) = 1$.

If the prime factorization is known, then finding GCD is trivial. For example, suppose

$$\begin{aligned} a &= 6825 = 3 * 5^2 * 7 * 13 \\ b &= 90 = 2 * 3^2 * 5 \end{aligned}$$

Then, gcd$(a, b) = 15 = 3 * 5$. For very large numbers, however, finding prime factorization is not a feasible approach, and we will see an alternative method for finding the GCD, known as Euclid's algorithm.

The GCD is useful for reducing a ratio to an equivalent form so that the numerator and denominator become relatively prime. For example, for the above numbers

$$\frac{b}{a} = \frac{90}{6825} = \frac{90/15}{6825/15} = \frac{6}{455}$$

The GCD plays a very important role in finding the inverse of an integer and in RSA cryptosystem.

Next, we study an ancient efficient algorithm, Euclidean Algorithm, for computing the GCD and finding the integers s and t so that $\gcd(a, b) = sa + tb$.

3. Euclidean GCD Algorithm

This ancient algorithm is based on the following observation.

Lemma 2: Let a and b be two positive integers, where $a > b$. Then,

$$\gcd(a, b) = \gcd(b, a \ mod \ b).$$

Proof:

(1) From Theorem 1, we know that a common divisor of (a, b) also divides

$$r = a \bmod b = a - qb.$$

So, every common divisor of (a, b) is also a common divisor of (b, r).

(2) The converse is also true. That is, $a = r + qb$. So, a is a linear sum of b and r. Thus, by Theorem 1, every common divisor of (b, r) is a common divisor of (a, b).

Therefore, $\gcd(a, b) = \gcd(b, r)$. ■

Euclid's algorithm applies this rule repeatedly to reduce the size of the numbers until the smaller number becomes 0. An iterative version of the algorithm follows.

```
int GCD(int $a$, int $b$){ // Assume $a > b \geq 0$
while ($b > 0$) {
   $r = a \bmod b$;
   $a = b$;  $b = r$;
    }
return $a$  // When $b = 0$;  $\gcd(a, 0) = a$;
}
```

Below is a numerical illustration of the algorithm, which finds $\gcd(7650, 285) = 15$.

Table 1: Illustration of Euclid's algorithm

Iteration	a	b	$r = a \bmod b$
1	7650	285	240
2	285	240	45
3	240	45	15
4	45	15	0
5	15	0	

Theorem 2: Let a and b be positive integers, $a > b$. There exist integers s and t so that

$$\gcd(a, b) = sa + tb$$

Furthermore, $\gcd(a, b)$ is the ***smallest*** positive integer which is a linear combination of a and b.

Proof: Since each iteration of the algorithm performs a mod operation, which is a linear combination of (a, b), the final GCD is also a linear combination of the inputs (a, b). Next, we prove that $\gcd(a, b)$ is the smallest positive linear combination of (a, b). Suppose to the contrary, there is a smaller integer d',

$$0 < d' < \gcd(a, b)$$

where $d' = s'a + t'b$ for some integers s', t'. Since d' is a linear combination of (a, b), by Theorem1, $\gcd(a, b)$ divides d'. Thus, $\gcd(a, b) \leq d'$, which is a contradiction. ■

It is easy to modify the iterative Euclid's algorithm to compute the integers (s, t), so that $\gcd(a, b) = sa + tb$. Let (a, b) be the value of the inputs at the beginning. Let a_i, b_i denote the variables at the start of iteration i, and let $q_i = \lfloor a_i/b_i \rfloor$ and $r_i = a_i - q_i b_i = a_i \bmod b_i$. The algorithm will maintain two variables (s^a, t^a) associated with a_i, which show the value of a_i in terms of the original inputs (a, b). That is, $a_i = s^a * a + t^a * b$. Similarly, the algorithm maintains variables (s^b, t^b) associated with b_i, and variables (s^r, t^r) associated with r_i. Then, when iteration i performs the mod operation, it updates the variables r_i and (s^r, t^r) accordingly. That is,

$$(s^r, t^r) = (s^a, t^a) - q * (s^b, t^b)$$

Below is an illustration of this implementation for the same pair of inputs a, b. (The pseudo-code is left to the student as an exercise.)

Table 2: Computation of integers s, t by the iterative Euclid's algorithm

Iteration	a	b	$r = a \bmod b$
1	7650 $(1,0)$	285 $(0,1)$	$240 = a_1 - 26b_1$ $(1, -26)$
2	285 $(0,1)$	240 $(1, -26)$	$45 = a_2 - b_2$ $(-1, 27)$
3	240 $(1, -26)$	45 $(-1, 27)$	$15 = a_3 - 5b_3$ $(6, -161)$
4	45 $(-1, 27)$	15 $(6, -161)$	0

The algorithm returns $\gcd = 6$, $s = 6, t = -161$. That is,

$$\gcd(7650, 285) = 15 = 6 * 7650 - 161 * 285.$$

Next, we present a recursive version of the GCD algorithm, which also computes the integers (s, t).

```
int RGCD(int $a$, int $b$, int $s$, int $t$)
{ // Inputs: $a, b$. It is assumed that $a > b \geq 0$;
  // Outputs:  Returned parameters $s, t$;  and GCD returned as the value of the
      function,
if $(b == 0)$ $\{ s = 1;\ t = 0;\ \text{return } a \}$;
$q = \lfloor a/b \rfloor$;
$G =$ RGCD($b,\ a - qb,\ s_2,\ t_2$);
// Now GCD = $s_2 b + t_2(a - qb) = t_2 a + (s_2 - t_2 q)\, b$
$s = t_2;\ \ t = s_2 - t_2 q$;
return $(G)$;  }
```

Analysis of Euclid's Algorithm

Lemma 3: Let a be the larger of the two inputs to GCD algorithm. The number of iterations (thus the number of mod operations) is at most $2\log_2 a$.

Proof: We show that a will reduce by at least a factor of 2 after one or two iterations. Let a_i, b_i denote the value of the variables at the beginning of iteration i of the algorithm, and let r_i be the mod value computed during this iteration. There are two cases:

1. $b_i \le a_i/2$: Then after one iteration, $a_{i+1} = b_i \le a_i/2$.
2. $b_i > a_i/2$: Then $r_i = a_i - b_i < a_i/2$. So, $a_{i+2} = b_{i+1} = r_i < a_i/2$.

So the size of the larger input a is reduced by at least a factor of 2 after at most two iterations. Therefore, the number of iterations is at most $2\log_2 a$.

Finally, we may also express the running time in terms of the smaller input b. Since after one iteration, $a_{i+1} = b_i$, then the number of iterations is at most 1+$2\log_2 b$. ■

Next, we present an interesting alternative analysis in terms of Fibonacci sequence, which is defined as $F_1 = F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$, $n \ge 3$. This alternative analysis improves the worst-case number of iterations to $1.44\log_2 a$.

Lemma 4: If the larger input to GCD algorithm is $a < F_n$ for some integer n, then the number of mod operations of the algorithm is at most $n-3$.

Proof: This is proved by (strong) induction on n. For base cases, we use $n = 3, n = 4$.

- For $n = 3$, the larger input is $a < F_3 = 2$, so $a \le 1$ and clearly no mod operation is needed, which is $n - 3 = 0$.
- For $n = 4$, the larger input is $a < F_4 = 3$, so $a \le 2$. In this case, at most one mod operation is needed, which is $n - 3 = 1$. So the two base cases are correct.

Now suppose $a < F_n$ for some $n \ge 5$. Let a_i, b_i denote the variables at the beginning of iteration i of the algorithm, and let r_i be the mod computed in this iteration. Initially, $a_1 < F_n$. We consider two cases:

1. If $b_1 < F_{n-1}$: Then, after one iteration, the larger number is $a_2 = b_1 < F_{n-1}$.
2. If $b_1 \ge F_{n-1}$: Then, $r_1 = a_1 - b_1 < F_n - F_{n-1} = F_{n-2}$. And r_1 becomes the larger input after two iterations. So, $a_3 = b_2 = r_1 < F_{n-2}$.

In simple words, if initially the larger input is $< F_n$, then

- after one iteration the larger input will be $< F_{n-1}$, or
- after two iterations the larger input will be $< F_{n-2}$.

Therefore, by induction, the total number of mod operations is at most $n-3$. ■

Corollary 1: Let a be the larger input to GCD algorithm. The number of mod operations is at most $(1.4405)\log_2 a$.

Proof: Suppose the larger input a is $F_{n-1} \le a < F_n$ for some integer n. The above lemma established the number of mod operations is at most $n-3$. And, it is easy to prove by induction that

$$F_n \ge (1.618)^{n-2}, \qquad \forall n \ge 2$$

(This induction proof is left to the reader.) Therefore, $a \ge F_{n-1} \ge (1.618)^{n-3}$. Taking the log of both sides yields the worst-case number of mod operations:

$$n-3 \le \log_{1.618} a = (\log_2 a)(\log_{1.618} 2) < (1.4405)\log_2 a.$$ ■

It is interesting to note that if initially $a = F_n$ and $b = F_{n-1}$ (the exact equality) then the number of iterations will be exactly $n-2$. Below is an example for $a = F_8$ and $b = F_7$. Note that iteration 6 is the last mod operation.

Table 3: The working of Euclid's algorithm with inputs $a = F_8, b = F_7$.

Iteration	a	b	$r = a \bmod b$
1	$F_8 = 21$	$F_7 = 13$	$F_6 = 8$
2	$F_7 = 13$	8	5
3	$F_6 = 8$	5	3
4	$F_5 = 5$	3	2
5	$F_4 = 3$	2	1
6	$F_3 = 2$	1	0
	1	0	

4. Modulo Arithmetic

The following lemma is useful for handling a series of multiplications, modulo n.

Lemma 5: Let $x, y, z,$ and n be positive integers.

(1) $(x * y) \bmod n = \big((x \bmod n) * (y \bmod n)\big) \bmod n$
(2) $(x * y * z) \bmod n = \big((x * y) \bmod n\big) * z) \bmod n$

Proof: We prove only (1), since (2) basically follows from (1). Let $x \bmod n = d_1$ and $y \bmod n = d_2$. This means $x = d_1 + r_1 n,$ and $y = d_2 + r_2 n,$ for some integers r_1, r_2. Then,

$$(x * y) \bmod n = (d_1 + r_1 n)(\, d_2 + r_2 n) \bmod n$$
$$= \big((d_1 d_2) + (d_1 r_2 + r_1 d_2)n + r_1 r_2 n^2\big) \bmod n = (d_1 d_2) \bmod n.$$

■

Suppose we want to compute a series of multiplications $(x_1 * x_2 * x_3 * \cdots * x_p)\ mod\ n$.
If we first perform the entire series of multiplications and then perform the mod operation at the end, the intermediate results may get too large. Instead, we perform a mod operation after each multiplication, to prevent the intermediate results from becoming too large: $x_1 * x_2 \bmod n\ *\ x_3\ \bmod n * \cdots * x_p \bmod n$
(The above lemma provides the justification for reordering the mod operations.)

5. Inverse of Integers Modulo n

Definition: Let x and n be two positive integers. The ***inverse*** of x modulo n, if it exists, is a positive integer $s < n$ such that

$$x * s \bmod n = 1.$$

The inverse of x is denoted as x^{-1}. ■

For example, inverse of 5 mod 13 is 8, since $5 * 8 \bmod 13 = 40 \bmod 13 = 1$. The inverse does not always exist. For example, inverse of 6 mod 15 does not exist. (This may be easily verified by testing all integers 1 to 14.) The following theorem proves the necessary and sufficient condition for the inverse to exist.

Theorem 3: Let x and n be two positive integers. The inverse of x modulo n exists, and is unique, if and only if $\gcd(x, n) = 1$.

Proof:
1. Suppose $\gcd(x, n) = 1$. By Theorem 2, there are integers s, t such that
$$sx + tn = 1$$
This means $sx \bmod n = 1$, which means $(s \bmod n)\ x \bmod n = 1$. Therefore,
$$x^{-1} = s \bmod n.$$
2. Suppose the inverse of x mod n exists. Let s be the inverse. Then, $sx \bmod n = 1$. Thus, there is an integer t such that $sx + tn = 1$. By Theorem 2,
$$\gcd(x, n) = sx + tn = 1.$$

The uniqueness may be established by a simple proof by contradiction. Suppose an integer x has inverses s and s', where $s < s' < n$. Then,

$$xs\ mod\ n = xs' mod\ n = 1,$$
$$x(s' - s) = rn, \quad \text{for some integer } r.$$

Since x is prime relative to n, and $s' - s < n$, the latter is true only if $r = 0, s' = s$. ■

Computation of Inverse using GCD

The above theorem suggests how to compute the inverse efficiently by computing GCD. To compute the inverse of an integer x mod n, we do:

Find $\gcd(x, n)$, and integers (s, t) so that $\gcd(x, n) = sx + tn$.
If $\gcd(x, n) = 1$, then $x^{-1} = s \bmod n$;
Else, the inverse does not exist.

The table below shows the inverse of integers modulo 13. Since 13 is prime, all integers 1 to 12 are prime relative to 13 and thus have inverse. For example, to compute inverse of 6, we use GCD to find $\gcd(13, 6) = 1 = 1 * 13 - 2 * 6$. Thus, $6^{-1} = -2 \bmod 13 = 11$.

i	1	2	3	4	5	6	7	8	9	10	11	12
$i^{-1}\bmod 13$	1	7	9	10	8	11	2	5	3	4	6	12

The next table shows the inverse of integers 1 to 9 modulo 10. The only integers which are prime relative to 10 are {1, 3, 7, 9} and thus have inverse. Integers {2, 4, 5, 6, 8} are not prime relative to 10 and so they do not have inverse.

i	1	2	3	4	5	6	7	8	9
$\gcd(10, i)$	1	2	1	2	5	2	1	2	1
$i^{-1} \bmod 10$	1	-	7	-	-	-	3	-	9

The inverse plays a critical role in secure communications. In such a communication, the sender of a message encodes the message by applying some transformation so that an unauthorized third party will not be able to read it. Then, the receiver applies the inverse transformation to recover the original message.

A Naïve Communication Protocol

Consider the following naïve scheme for secure communications. Suppose a sender and a receiver privately agree on a secret pair of integers (s, n), where $\gcd(s, n) = 1$. The receiver computes the integer $s^{-1} \bmod n$, and saves it in a secure place. Then:

1. The sender, who wants to transmit a message as an integer $M < n$, *encodes*, or *encrypts*, the message by computing $E = (M * s) \bmod n$, and transmits E.

2. The receiver *decodes* the received data by computing $(E * s^{-1}) \bmod n$. This produces the original message because
$$(M * s \bmod n) * s^{-1} \bmod n = (M * s * s^{-1}) \bmod n = M.$$

However, this scheme has some problems. For example, how do the sender and receiver communicate the pair of integers (s, n) in a secure way?! Later, we will see a more sophisticated and secure communication scheme which uses the idea of inverse but in conjunction with exponentiation modulo n.

6. Exponentiation Modulo n

We now describe an efficient algorithm to compute exponentiation modulo n. Suppose we have integers $x, p, \text{and } n$, and we want to compute $x^p \bmod n$. In an earlier module, we discussed how to perform exponentiation by repeated squaring. Here, we extend the algorithm to add the mod computation in a straightforward way.

```
Int Power $(\text{int } x, \text{int } p, \text{int } n)$ {  // Compute $x^p \bmod n$.  Assume $p \geq 1$.
if $(p == 1)$ return $(x \bmod n)$;
$T =$ Power $(x, \lfloor p/2 \rfloor, n)$;
$T = (T * T) \bmod n$;
if $(p \bmod 2 == 1)$
    $T = (T * x) \bmod n$;
return $T$; }
```

To analyze the running time of the algorithm, let $f(p)$ be the worst-case number of multiplications. Then,
$$f(p) = \begin{cases} 0, & p = 1 \\ f\left(\left\lfloor \frac{p}{2} \right\rfloor\right) + 2, & p > 1 \end{cases}$$

It is easy to prove by induction that the solution is $f(p) = 2\lfloor \log p \rfloor$.

Example: Below is computation of $x^p \ mod \ n$, for $x = 79,\ p = 5143,\ n = 4927$.

Power p	Square	Mod n	$* x$	Mod n
5143	2,560,000	2887	228,073	1431
2571	5,121,169	2016	159,264	1600
1285	22,401,289	3147	248,613	2263
642	10,316,944	4733	4,733	4733
321	6,105,841	1288	101,752	3212
160	9,339,136	2471	2,471	2471
80	2,047,761	3056	3,056	3056
40	10,909,809	1431	1,431	1431
20	19,430,464	3303	3,303	3303
10	4,064,256	4408	4,408	4408
5	1,726,596	2146	169,534	2016
2	6,241	1314	1,314	1314
1				79

7. Number-Theoretic Background

Let $Z_n = \{0, 1, \cdots, n-1\}$.

Lemma 6: Given positive integers x and n, where $\gcd(x, n) = 1$. The sequence

$$(x * i \bmod n \mid i = 1, 2, \cdots, n-1)$$

is a permutation of $(1, 2, \cdots, n-1)$.

Proof: We show that for any pair of positive integers i and j, where $1 \le i < j \le n-1$,

$$x * i \bmod n \neq x * j \bmod n$$

That is, for any integer r,

$$x\,(j-i) \neq rn.$$

The latter is true since x is prime relative to n, and $0 < (j-i) < n$. ■

Below is an example for $n = 8,\ x = 3$.

i	1	2	3	4	5	6	7
$3i \bmod 8$	3	6	1	4	7	2	5

A Word of Caution About Modulo Arithmetic: Consider the following equality, where x and i are two positive integers in Z_n. Can we cancel the factor i from both sides and conclude $x = 1$?

$$x * i \bmod n = i$$

Not always! In the above example, $x = 3,\ i = 4, n = 8$. Here, $x \neq 1$, but

$$x * 4 \bmod 8 = 4$$

(General rule: Multiplying both sides by a term mod n is valid. Dividing both sides by a term is not valid.)

However, if integer i has an inverse mod n, then we can multiply both sides by the inverse i^{-1}, and as a result cancel i from both sides:

$$\begin{aligned}
&x * i \bmod n = i\\
&\big((x * i \bmod n) * i^{-1}\big) \bmod n = (i * i^{-1}) \bmod n\\
&(x * i * i^{-1}) \bmod n = 1\\
&x = 1
\end{aligned}$$

We are now ready for our important theorem.

Theorem 4 (Fermat's Little Theorem): Let p be a prime integer, and let x be a positive integer with $\gcd(x, p) = 1$. (Since p is prime, this condition means $x \neq rp$ for any integer r.) Then,

$$\boxed{x^{p-1} \bmod p = 1}$$

Proof: Let us form the following product of $p - 1$ terms.

$$F = (x * 1 \bmod p)(x * 2 \bmod p) \cdots (x * (p-1) \bmod p) \bmod p$$

From Lemma 6, we know the sequence $(x * i \bmod p \mid i = 1, 2, \ \cdots, p-1)$ is a permutation of $(1, 2, \cdots, p-1)$. Therefore, the product of the terms in the sequence results in $(p-1)!$. That is,

$$F = (p-1)! \bmod p$$

But we may also rearrange the product terms to separate all x factors, and thus produce

$$F = x^{p-1} * (\Pi_{\mathrm{i=1:p-1}} i) \bmod \mathrm{p} = x^{p-1}(p-1)! \bmod p$$

Therefore,

$$x^{p-1}(p-1)! \bmod p = (p-1)! \bmod p$$

Since p is prime, all integers $(1, 2, \cdots, p-1)$ have inverse mod p. So we may multiply both sides by $(p-1)^{-1}(p-2)^{-1} \cdots 1^{-1}$. This will result in cancelling the term $(p-1)!$ from both sides. Therefore,

$$x^{p-1} \bmod p = 1$$

■

The following table illustrates the computation of $x^i \bmod p, 1 \le i \le p-1$, for $x = 3, p = 7$. Observe that $3^6 \bmod 7 = 1$.

i	1	2	3	4	5	6
$3^i \bmod 7$	3	2	6	4	5	1

Next we generalize the above theorem to the case when p may not be prime.

Generalization of Fermat's Little Theorem (Euler's Theorem)

Definitions: Let $\phi(n)$ denote the number of integers in $Z_n = \{0, 1, \cdots, n-1\}$ which are prime relative to n. And let $Z_n^* = \{u_1, u_2, \cdots, u_{\phi(n)}\}$ be the set of elements in Z_n which are prime relative to n. ■

For the case when n is product of two primes p and q, $n = pq$, it is easy to compute $\phi(n)$. First, we count the integers in Z_n which are *not* prime relative to n. There are q integers in Z_n which are a multiple of p, namely $\{0, p, 2p, \cdots, (q-1)p\}$. Similarly, there are p integers which are a multiple of q. And integer 0 is common to both, so there are $p + q - 1$ integers not prime relative to n. Since $n = pq$, then

$$\phi(n) = pq - p - q + 1 = (p-1)(q-1)$$

Example: For $Z_{15} = \{0, 1, \cdots, 14\}$, $n = 15 = 3 * 5$, and $\phi(n) = (3-1)(5-1) = 8$. There are 8 integers in Z_{15} which are prime relative to $n = 15$. (The integers are highlighted in yellow below.) Thus, $Z_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$. ■

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

The set of elements in Z_n^* satisfy two important properties, proved below.

Lemma 7:

1. Every integer $i \in Z_n^*$ has an inverse mod n, and the inverse is in Z_n^*.
2. For every pair of integers i and j in Z_n^*, their product $(i * j \bmod n)$ is also in Z_n^*. (This property is called *closure under multiplication.*)

Proof:

1. Every integer $i \in Z_n^*$ has an inverse because $\gcd(i,n) = 1$. And i^{-1} has an inverse. That is, $(i^{-1})^{-1} = i$. And this requires $\gcd(i^{-1}, n) = 1$, which means $i^{-1} \in Z_n^*$. For example, the inverse of all integers in Z_{15}^* are shown below.

$i \in Z_{15}^*$	1	2	4	7	8	11	13	14
$i^{-1} \bmod 15$	1	8	4	13	2	11	7	14

2. For every pair of integers i and j in Z_n^*, their product $(i * j \bmod n)$ is also in Z_n^*. The reason is obvious: Since i and j are both prime relative to n, their product $(i * j)$ is also prime relative to n, and therefore $(i * j \bmod n)$ is prime relative to n. For example, for integers 4 and 11 in Z_{15}^*, their product is $(4 * 11 \bmod 15 = 14)$, which is in Z_{15}^*. ■

Next, let us state a generalization of Lemma 6.

Lemma 8: Let $Z_n^* = \{u_1, u_2, \cdots, u_{\phi(n)}\}$. Let x be any integer in Z_n^*. The sequence

$$(x * u_i \bmod n \mid i = 1, 2, \cdots, \phi(n))$$

is a permutation of $(u_1, u_2, \cdots, u_{\phi(n)})$.

Proof: The proof is very similar to the proof of Lemma 6. We need to show that for any pair u_i and u_j, where $u_i < u_j$,

$$x * u_i \bmod n \neq x * u_j \bmod n$$

That is, for any integer r,

$$x(u_j - u_i) \neq rn$$

This is true since x and n are relatively prime, and $0 < u_j - u_i < n$. ■

Below is an illustration for Z_{15}^*, and $x = 7$.

$u_i \in Z_{15}^*$	1	2	4	7	8	11	13	14
$7u_i \bmod 15$	7	14	13	4	11	2	1	8

We are now ready for a generalization of Fermat's Little Theorem.

Theorem 5 (Euler's Theorem): Let n be a positive integer, and $Z_n^* = \{u_1, u_2, \cdots, u_{\phi(n)}\}$ be the set of elements in Z_n^* which are prime relative to n. And let x be an integer in Z_n^*. Then,

$$\boxed{x^{\phi(n)} \bmod n = 1}$$

Proof: The proof is similar to the proof of Fermat's Little Theorem. Let's form the following product.

$$F = (\Pi_{i=1:\phi(n)}\ (xu_i \bmod n)) \bmod n$$

On one hand, from Lemma 8, we know the sequence $(xu_i \bmod n \mid i = 1, 2, \cdots, \phi(n))$ is a permutation of $(u_1, u_2, \cdots, u_{\phi(n)})$. So, the product of the terms in the sequence becomes

$$F = (\mathrm{u}_1 * \mathrm{u}_2 * \cdots * \mathrm{u}_{\phi(\mathrm{n})}) \bmod n$$

On the other hand, we may rearrange the product terms to separate all x factors, and produce

$$F = x^{\phi(n)} * (\mathrm{u}_1 * \mathrm{u}_2 * \cdots * \mathrm{u}_{\phi(\mathrm{n})}) \bmod n$$

Therefore,

$$x^{\phi(n)} * (\mathrm{u}_1 * \mathrm{u}_2 * \cdots * \mathrm{u}_{\phi(\mathrm{n})}) \bmod n = (\mathrm{u}_1 * \mathrm{u}_2 * \cdots * \mathrm{u}_{\phi(\mathrm{n})}) \bmod n$$

Since every u_i term has an inverse mod n, we multiply both sides by the inverse terms. This cancels all u_i terms from both sides and gives: $x^{\phi(n)} \bmod n = 1$. ■

Note that if n is prime, all integers in $(1, 2, \cdots, n-1)$ are prime relative to n, and $\phi(n) = n - 1$. In this case, Euler's Theorem reduces to Fermat's Little Theorem.

Computing Inverse by Exponentiation: Euler's Theorem gives another way of computing the inverse of an integer x mod n, where $x \in Z_n^*$,. Namely,

$$\boxed{x^{-1} = x^{\phi(n)-1} \bmod n}$$

Example: Let $x = 7 \in Z_{15}^*$. Since $n = 15 = 3 * 5$, $\phi(n) = 8$. Then

$$x^{-1} = x^{\phi(n)-1} \bmod 15 = 7^7 \bmod 15 = 13.$$

■

Corollary 2: Let n be a positive integer, and $Z_n^* = \{u_1, u_2, \cdots, u_{\phi(n)}\}$ be the set of elements prime relative to n. And let x be an integer in Z_n^*. Then, for any integer r,

$$\boxed{x^{r\phi(n)+1} \bmod n = x}$$

Proof: By Euler's Theorem, $x^{\phi(n)} \bmod n = 1$. Raise both sides to the power r to get: $x^{r\phi(n)} \bmod n = 1$. Then multiply both sides by x. ■

Generalization of Euler's Theorem

Euler's Theorem requires $x \in Z_n^*$, so x is prime relative to n. Our next theorem removes this restriction, so x may be any integer.

Theorem 6: Let $n = pq$ be product of two primes p and q, and $\phi(n) = (p-1)(q-1)$. Let x be any positive integer, $x < n$. Then, for any integer r,

$$\boxed{x^{r\phi(n)+1} \bmod n = x}$$

Proof: If $\gcd(x, n) = 1$, the theorem is true by the above corollary. Now suppose x is not prime relative to n. Since $x < n$, then x must be either a multiple of p or a multiple of of q, but not both. Suppose $x = tp$, for some positive integer $t < q$. Since x is prime relative to q, then by Fermat's Little Theorem,

$$x^{q-1} \bmod q = 1.$$

Raise both sides to the power of $r(p-1)$ to get

$$x^{r(p-1)(q-1)} \bmod q = 1.$$

Thus, for some integer k,

$$x^{r\phi(n)} = kq + 1$$

Now, multiply both sides by $x = tp$.

$$x^{r\phi(n)+1} = kt(pq) + x$$
$$x^{r\phi(n)+1} = kt\,n + x$$
$$x^{r\phi(n)+1} \bmod n = x.$$

■

Example: For $n = 15 = 3 * 5$, $\phi(n) = 8$, and $x = 5$,

$$x^{\phi(n)+1} \bmod 15 = 5^9 \bmod 15 = 5.$$

■

Example: Let $n = 15 = 3 * 5,\ \ \phi(n) = 2 * 4 = 8$. The following table shows a row $(x^i \ mod\ n \mid i = 1,2,\cdots,9)$ for every positive integer $x < n$. The following observations are made from the table:

- For every x, column 9 confirms Theorem 6, namely $x^{\phi(n)+1} mod\ n = x$.
- For those rows where $x \in Z_n^*$ (highlighted in yellow), column 8 shows a value 1 thus confirming Euler's Theorem. That is, $x^{\phi(n)} mod\ n = 1$. And for these rows, column 7 confirms $x^{\phi(n)-1} mod\ n = x^{-1}$. ■

i	1	2	3	4	5	6	7	$8 = \phi(n)$	$9 = \phi(n) + 1$
$1^i mod\ n$	1	1	1	1	1	1	1	1	1
$2^i mod\ n$	2	4	8	1	2	4	8	1	2
$3^i mod\ n$	3	9	12	6	3	9	12	6	3
$4^i mod\ n$	4	1	4	1	4	1	4	1	4
$5^i mod\ n$	5	10	5	10	5	10	5	10	5
$6^i mod\ n$	6	6	6	6	6	6	6	6	6
$7^i mod\ n$	7	4	13	1	7	4	13	1	7
$8^i mod\ n$	8	4	2	1	8	4	2	1	8
$9^i mod\ n$	9	6	9	6	9	6	9	6	9
$10^i mod\ n$	10	10	10	10	10	10	10	10	10
$11^i mod\ n$	11	1	11	1	11	1	11	1	11
$12^i mod\ n$	12	9	3	6	12	9	3	6	12
$13^i mod\ n$	13	4	7	1	13	4	7	1	13
$14^i mod\ n$	14	1	14	1	14	1	14	1	14

8. The RSA Public-Key Cryptosystem

We now have the theoretical background for an understanding of the cryptosystem. The RSA public-key cryptosystem is named after its inventors: Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. It enables many senders to securely communicate with one receiver.

An example we use is the commerce communication between many clients or customers of a bank (called public) and the bank. All the clients have the same ***public keys*** they use to encode their messages. And the bank (recipient) holds a secret ***private key*** used to decode the messages. Furthermore, the public keys work in such a way that although all the senders (public) share the same public keys, they still will not be able to decipher each other's messages (even if they try to eavesdrop) because they lack the private keys needed to unlock the encoded messages.

Here is how the RSA cryptosystem works. First, the bank (receiver) privately picks two very large prime numbers p and $q,$ and computes

$$n = pq, \quad \phi(n) = (p-1)(q-1).$$

The bank also picks a small number s which is **prime relative to $\boldsymbol{\phi(n)}$**, and privately computes

$$t = s^{-1} \bmod \phi(n).$$

Notice the inverse is mod $\phi(n),$ not mod n. Thus, $st = r\phi(n) + 1$ for some integer r.

In practice, a small prime number may be chosen for $s,$ but it must be done carefully to make sure that $\gcd(s, \phi(n)) = 1$. For example, suppose $n = 41 * 67,$ thus $\phi(n) = 40 * 66 = 2640$. Then, for example, $s = 11$ would not work since it is not prime relative to 2640.

The bank announces the two integers $s \text{ and } n$ as ***public keys*** to all of its clients. The integer $t = s^{-1}$ is kept secret by the bank, and is called the ***private key***.

The communication between the clients and the bank proceeds as follows.

1. The client (sender), who wants to transmit an information in the form of a large integer $x < n,$ uses the public keys (s, n) and computes
$$y = x^s \bmod n$$
The value y, called the **encrypted** message, is transmitted over the Internet.
2. The bank (receiver) takes the data $y,$ and uses its private key $t,$ together with the public key $n,$ to decrypt the message by computing
$$z = y^t \bmod n$$

By Theorem 6, we know the result z is indeed the original message x.

$$z = y^t \bmod n = (x^s \bmod n)^t \bmod n = x^{st} \bmod n = x^{r\phi(n)+1} \bmod n = x.$$

Example: Let us pick a small example for illustration. We choose two prime numbers $p = 5,\ q = 13$.

$$n = p * q = 5 * 13 = 65$$
$$\phi(n) = (p-1)(q-1) = 4 * 12 = 48$$

And let us pick a small prime number s, while making sure that $\gcd(s, \phi(n)) = 1$.

$$s = 11$$

Then,

$$t = s^{-1}\ mod\ 48 = (-13)\ mod\ 48 = 35$$

So, the public keys for all senders are: $s = 11,\ n = 65$. And the receiver keeps the private key $t = 35$, and the public key $n = 65$.

Suppose a sender wants to send a message (number) $x = 53$. The message x is encrypted using the public keys (s, n).

$$y = x^s\ mod\ n = 53^{11}\ mod\ 65 = 27$$

The computed value y is transmitted. The receiver takes the data y and decrypts it, using the private key $t = 35$ and public key n.

$$z = y^t\ mod\ n = 27^{35}\ mod\ 65 = 53$$

And the result z is the original message x. ■