Chapter 2
Mathematical Preliminaries

Logarithms

\[ \log_b y = x \iff b^x = y \iff b^{\log_b y} = y \]

\[ \log nm = \log n + \log m \]

\[ \log \frac{n}{m} = \log n - \log m \]

\[ \log n^r = r \log n \]

\[ \log_a n = \frac{\log_b n}{\log_b a} \]

Summations

\[
\sum_{i=1}^{n} f(i) = f(1) + f(2) + \cdots + f(n-1) + f(n)
\]

\[
\sum_{i=1}^{n} i = \frac{n(n+1)}{2}
\]

\[
\sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6}
\]

\[
\sum_{i=1}^{n} n = n \log n
\]

\[
\sum_{i=1}^{n} \frac{a^i - 1}{a-1} \quad \text{for } a > 1
\]

The Factorial Function

Definition:

\[ n! = 1 \cdot 2 \cdot 3 \cdots \cdot (n-1) \cdot n \]

Stirling’s approximation:

\[ n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \]

or

\[ \log(n!) = O(n \log n) \]

How to approve it?

Bounds of Factorial Function

Let

\[ \log n! = \sum_{x=1}^{n} \log x \]

Then

\[ \int_x^n \log x \, dx \leq \sum_{x=1}^{n} \log x \leq \int_x^{n+1} \log x + 1 \, dx \]

which gives

\[ n \log \left( \frac{n}{e} \right) + 1 \leq \log n! \leq (n+1) \log \left( \frac{n+1}{e} \right) + 1 \]

So

\[ \left( \frac{n}{e} \right)^n \leq n! \leq \left( \frac{n+1}{e} \right)^{n+1} \]
Recurrence Relations

- For the factorial function:
  \[ n! = n \cdot (n-1)!, \quad 1! = 0! = 1 \]

- For the Fibonacci sequence:
  \[ F(n) = F(n-1) + F(n-2), \quad F(1) = F(2) = 1 \]

Recursion

- Recursion means defining something, such as a function, in terms of itself.
  - For example, if we want \( f(x) = x! \)
    - we can define \( f(x) \) as
      \[ f(x) = \begin{cases} 1 & \text{if } x < 1 \\ x \cdot f(x-1) & \text{else} \end{cases} \]

Fibonacci Numbers

- Fibonacci(0) = 0
- Fibonacci(1) = 1
- Fibonacci(n) = Fibonacci(n - 1) + Fibonacci(n - 2)
  for \( n > 1 \)

Fibonacci Performance

- Lots of duplicated values

Fibonacci sequence

- Definition of the Fibonacci sequence
  - Non-recursive:
    \[ F(n) = \frac{1 + \sqrt{5}}{2^n} - \frac{1 - \sqrt{5}}{2^n} \]
  - Recursive:
    \[ F(n) = F(n-1) + F(n-2) \]
    or:
    \[ F(n+1) = F(n) + F(n-1) \]

- Must always specify base case(s)!
  - \( F(1) = 1, F(2) = 1 \)
  - Note that some will use \( F(0) = 1, F(1) = 1 \)

Fibonacci sequence in Java

```java
long Fibonacci (int n) {
  int f1 = f2 = f3 = 1;
  for (int i=3; i <= n; i++) {
    f3 = f1+f2; f1 = f2; f2 = f3;
  }
  return f3;
}
```

```java
long Fibonacci2 (int n) {
  return (long) ((Math.pow((1.0+Math.sqrt(5.0)),n) -
                  Math.pow((1.0-Math.sqrt(5.0)),n)) /
                  (Math.sqrt(5) * Math.pow(2,n)));
}
```
Hanoi Tower - Instructions
1. Transfer all the disks from pole A to pole B.
2. You may move only ONE disk at a time.
3. A large disk may not rest on top of a smaller one at anytime and disks cannot be placed on anywhere other than poles A, B, and C.

Try this one!

And this one

Now try this one!

How to solve Tower of Hanoi of n disks?
• If n = 1, “move disk 1 from A to B”, done.
• If n > 1,
  1. Solve the Tower of Hanoi of n-1 disks, from A to C;
  2. “move disk n from A to B”
  3. Solve the Tower of Hanoi of n-1 disks, from C to B.

Hanoi ( int n, char A, char B, char C ) {
    if (n==1) cout << "move disk 1 from " << A << " to " << B << endl;
    else {
        Hanoi(n-1, A, C, B);
        cout << "move disk " << n << " from " << A << " to " << B << endl;
        Hanoi(n-1, C, B, A);
    }
}

Counting the moves:
Let f(n) be the number of moves for n disks.

f(1) = 1;
f(n) = 2f(n-1) + 1.

Let f(n) be the number of moves for n disks.

<table>
<thead>
<tr>
<th>Number of Disks</th>
<th>Number of Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>f(1) = 1</td>
</tr>
<tr>
<td>2</td>
<td>f(2) = 2*1 + 1</td>
</tr>
<tr>
<td>3</td>
<td>f(3) = 2*3 + 1</td>
</tr>
<tr>
<td>4</td>
<td>f(4) = 2*7 + 1</td>
</tr>
<tr>
<td>5</td>
<td>f(5) = 2*15 + 1</td>
</tr>
<tr>
<td>6</td>
<td>f(6) = 2*31 + 1</td>
</tr>
</tbody>
</table>

The general solution is f(n) = 2^n – 1.
Fascinating fact

So the formula for finding the number of steps it takes to transfer $n$ disks from post A to post C is:

$$2^n - 1$$

- If $n = 64$, the number of moves of single disks is $2$ to the $64$th minus $1$, or $18,446,744,073,709,551,615$ moves! If one worked day and night, making one move every second it would take slightly more than 580 billion years to accomplish the job! - far, far longer than some scientists estimate the solar system will last.

Recursive definitions

- A recursive definition is one in which something is defined in terms of itself
- Almost every algorithm that requires looping can be defined iteratively or recursively
- All recursive definitions require two parts:
  - Base case
  - Recursive step
- The recursive step is the one that is defined in terms of itself
- The recursive step must always move closer to the base case

How Recursion Works

- When a function is called, some information needs to be saved in order to return the calling module back to its original state (i.e., the state it was in before the call).
- We need to save information such as the local variables and the spot in the code to return to after the called function is finished.

How Recursion Works

- To do this we use a stack.
- Before a function is called, all relevant data is stored in a stackframe.
- This stackframe is then pushed onto the system stack.
- After the called function is finished, it simply pops the system stack to return to the original state.
- Stack is the hidden memory used by recursive functions.

Factorial Example

```c
int factorial(int n)
{
    if (n == 0) return 1;
    else return n * factorial(n-1);
}
```

There is very little overhead in calling this function, as it has only one word of local memory, for the parameter $n$. However, when we try to compute factorial(20), there will end up being 21 words of memory allocated - one for each invocation of the function.
Factorial Example

factorial(20) -- allocate 1 word of memory,
call factorial(19) -- allocate 1 word of memory,
call factorial(18) -- allocate 1 word of memory,
... 
call factorial(2) -- allocate 1 word of memory,
call factorial(1) -- allocate 1 word of memory,
call factorial(0) -- allocate 1 word of memory,
at this point 21 words of memory

... and 21 activation records have been allocated.

return 1. -- release 1 word of memory.
return 1*1. -- release 1 word of memory.
return 2*1. -- release 1 word of memory.

Tail Recursion

• Most recursive algorithms can be translated, by a fairly mechanical procedure, into iterative algorithms. Sometimes this is very straightforward - for example, most compilers detect a special form of recursion, called tail recursion, and automatically translate into iteration without your knowing. Sometimes, the translation is more involved: for example, it might require introducing an explicit stack with which to ‘fake’ the effect of recursive calls.

static int recSerS(int[] a, int target, int n) {
    if (n == -1) return -1; // target not found
    if (a[n] == target) return n; // target found
    return recSerS(a, target, n - 1);
}

The parameter n is the only that changes.
The tail-recursion removal procedure produces:

static int NonRecSerS(int[] a, int target, int n) {
    while (true) {
        if (n == -1) return -1; // target not found
        if (a[n] == target) return n; // target found
        n = n - 1;
    }
}

Recursion pros

• Easy to program
• Easy to understand

Recursion cons

• The main disadvantage of programming recursively is that, while it makes it easier to write simple and elegant programs, it also makes it easier to write inefficient ones.
• The stack is the hidden memory usage if it is not tail-recursion.
• When we use recursion to solve problems we are interested exclusively with correctness, and not at all with efficiency. Consequently, our simple, elegant recursive algorithms may be inherently inefficient.
Recursion cons

• Consider the recursive Fibonacci generator
• How many recursive calls does it make?
  – F(1): 1
  – F(2): 1
  – F(3): 3
  – F(4): 5
  – F(5): 9
  – F(10): 109
  – F(20): 13,529
  – F(30): 1,664,079
  – F(40): 204,658,309
  – F(50): 25,172,538,049
  – F(100): 708,449,696,358,523,830,149 ≈ 7 × 10^20

  ▷ At 1 billion recursive calls per second (generous), this would take over 22,000 years.
  ▷ But that would also take well over 10^12 Gb of memory!

How many binary strings of length n that do not contain the pattern 11?

• n = 1: 1 string (1, the empty string)
• n = 2: 2 strings (0 and 1)
• n = 3: 5 strings (000, 001, 010, 100, 101)
• n = 4: 8 strings (0000, 0001, 0010, 0100, 1000, 0101, 1001, 1010)
• Any pattern?
• A Fibonacci sequence!

How many binary strings of length n that do not contain the pattern 11?

• n = 2: 3 strings (00, 01, 10)
• n = 3: 5 strings (000, 001, 010, 100, 101)
• n = 4: 8 strings (0000, 0001, 0010, 0100, 1000, 0101, 1001, 1010)
  ▷ The strings of n=4 can be divided into two classes:
  ▷ X = { 0000, 0001, 0010, 0100, 0101 } and
  ▷ Y = { 1000, 1001, 1010 }

  ▷ X can be obtained from n = 3: adding a leading 0
  ▷ Y can be obtained from n = 2: adding leading 10.

A Puzzle

• There are seven steps from the first floor to the second floor. You can go one, two, or three steps in one stride. How many different ways you can climb seven steps?

  Let f(n) be the number of different ways you climb a n-step stair.

  f(1) = 1 // one stride of 1 step
  f(2) = 2 // either a stride of 2 steps or 2 strides of 1 step
  f(3) = 4 // either a stride of 3 steps, or 1 stride of 2 steps followed by f(1), or 1 stride of 1 step ...

  f(n) = f(n-3) + f(n-2) + f(n-1) // either 1 stride of 3 steps followed by f(n-3), ...

Proof methods

• We will discuss ten proof methods:
  1. Direct proofs
  2. Indirect proofs
  3. Proof by contradiction
  4. Proof by cases
  5. Proofs of equivalence
  6. Counterexamples
Direct proofs

• Consider an implication: \( p \rightarrow q \)
  – If \( p \) is false, then the implication is always true
  – Thus, show that if \( p \) is true, then \( q \) is true

• To perform a direct proof, assume that \( p \) is true, and show that \( q \) must therefore be true

Direct proof example

• Show that the square of an even number is an even number
  – Rephrased: if \( n \) is even, then \( n^2 \) is even
  
  • Assume \( n \) is even
  – Thus, \( n = 2k \), for some \( k \) (definition of even numbers)
  – \( n^2 = (2k)^2 = 4k^2 = 2(2k^2) \)
  – As \( n^2 \) is 2 times an integer, \( n^2 \) is thus even

Indirect proofs

• Consider an implication: \( p \rightarrow q \)
  – It’s contrapositive is \( \neg q \rightarrow \neg p \)
  – Is logically equivalent to the original implication!
  – If the antecedent \( (\neg q) \) is false, then the contrapositive is always true
  – Thus, show that if \( \neg q \) is true, then \( \neg p \) is true

• To perform an indirect proof, do a direct proof on the contrapositive

Indirect proof example

• If \( n^2 \) is an odd integer then \( n \) is an odd integer

  • Prove the contrapositive: If \( n \) is an even integer, then \( n^2 \) is an even integer

  • Proof: \( n=2k \) for some integer \( k \) (definition of even numbers)
  – \( n^2 = (2k)^2 = 4k^2 = 2(2k^2) \)
  – Since \( n^2 \) is 2 times an integer, it is even

Which to use

• When do you use a direct proof versus an indirect proof?

• If it’s not clear from the problem, try direct first, then indirect second
  – If indirect fails, try other proofs

Example of which to use

• Prove that if \( n \) is an integer and \( n^3+5 \) is odd, then \( n \) is even

  • Via direct proof
  – \( n^3+5 = 2k+1 \) for some integer \( k \) (definition of odd numbers)
  – \( n^3 = 2k-4 \)
  – \( n = \sqrt[3]{2k-4} \)

  • So direct proof didn’t work out. Next up: indirect proof
Example of which to use

• Prove that if \( n \) is an integer and \( n^3 + 5 \) is odd, then \( n \) is even

• Via indirect proof
  – Contrapositive: If \( n \) is odd, then \( n^3 + 5 \) is even
  – Assume \( n \) is odd, and show that \( n^3 + 5 \) is even
  – \( n = 2k + 1 \) for some integer \( k \) (definition of odd numbers)
  – \( n^3 + 5 = (2k+1)^3 + 5 = 8k^3 + 12k^2 + 6k + 6 = 2(4k^3 + 6k^2 + 3k + 3) \)
  – As \( 2(4k^3 + 6k^2 + 3k + 3) \) is 2 times an integer, it is even

Proof by contradiction

• Given a statement \( p \), assume it is false
  – Assume \( \neg p \)

• Prove that \( \neg p \) cannot occur
  – A contradiction exists

• Given a statement of the form \( p \rightarrow q \)
  – To assume it’s false, you only have to consider the case where \( p \) is true and \( q \) is false

Proof by contradiction example 1

• Theorem (by Euclid): There are infinitely many prime numbers.

• Proof. Assume there are a finite number of primes

• List them as follows: \( p_1, p_2, \ldots, p_n \).

• Consider the number \( q = p_1p_2 \ldots p_n + 1 \)
  – This number is not divisible by any of the listed primes
    • If we divided \( p_i \) into \( q \), there would result a remainder of 1
  – We must conclude that \( q \) is a prime number, not among the primes listed above
    • This contradicts our assumption that all primes are in the list \( p_1, p_2, \ldots, p_n \).

Proof by cases

• Show a statement is true by showing all possible cases are true

• Thus, you are showing a statement of the form:
  \( (p_1 \lor p_2 \lor \ldots \lor p_n) \rightarrow q \)

is true by showing that:

\[ [(p_1 \lor p_2 \lor \ldots \lor p_n) \rightarrow q] \leftrightarrow [(p_1 \rightarrow q) \land (p_2 \rightarrow q) \land \ldots \land (p_n \rightarrow q)] \]

Proof by cases example

• Prove that \( \left\lfloor \frac{a}{b} \right\rfloor \) where \( b \neq 0 \)

• Cases:
  – Case 1: \( a \geq 0 \) and \( b > 0 \)
    • Then \( \left\lfloor \frac{a}{b} \right\rfloor = a \), \( \left\lfloor \frac{b}{b} \right\rfloor = 1 \), \( \left\lfloor \frac{a}{b} \right\rfloor = \frac{a}{b} \)
  – Case 2: \( a \geq 0 \) and \( b < 0 \)
    • Then \( \left\lfloor \frac{a}{b} \right\rfloor = a \), \( \left\lfloor \frac{b}{b} \right\rfloor = -1 \), \( \left\lfloor \frac{a}{b} \right\rfloor = \frac{a}{b} \)
  – Case 3: \( a < 0 \) and \( b > 0 \)
    • Then \( \left\lfloor \frac{a}{b} \right\rfloor = -a \), \( \left\lfloor \frac{b}{b} \right\rfloor = 1 \), \( \left\lfloor \frac{a}{b} \right\rfloor = \frac{a}{b} \)
  – Case 4: \( a < 0 \) and \( b < 0 \)
    • Then \( \left\lfloor \frac{a}{b} \right\rfloor = -a \), \( \left\lfloor \frac{b}{b} \right\rfloor = -1 \), \( \left\lfloor \frac{a}{b} \right\rfloor = \frac{a}{b} \)

The thing about proof by cases

• Make sure you get ALL the cases
  – The biggest mistake is to leave out some of the cases

• Don’t have extra cases
  – We could have 9 cases in the last example
    • Positive numbers
    • Negative numbers
    • Zero
  – Those additional cases wouldn’t have added anything to the proof
Proofs of equivalences

- This is showing the definition of a bi-conditional
- Given a statement of the form “p if and only if q”
  - Show it is true by showing \((p \rightarrow q) \land (q \rightarrow p)\) is true

Proofs of equivalence example

- Show that \(m^2 = n^2\) if and only if \(m = n\) or \(m = -n\)
  - Rephrased: \((m^2 = n^2) \leftrightarrow [(m = n) \lor (m = -n)]\)
- Need to prove two parts:
  - \([(m = n) \lor (m = -n)] \rightarrow (m^2 = n^2)\)
    - Proof by cases:
      - Case 1: \(m = n\) \rightarrow \((m^2 = n^2)\)
        - \(m^2 = n^2\) and \((n)^2 = n^2\), so this case is proven
      - Case 2: \(m = -n\) \rightarrow \((m^2 = n^2)\)
        - \((-n)^2 = n^2\), so this case is proven
  - \((m^2 = n^2) \rightarrow [(m = n) \lor (m = -n)]\)
    - Subtract \(n^2\) from both sides to get \(m^2 - n^2 = 0\)
    - Factor to get \((m + n)(m - n) = 0\)
    - Since that equals zero, one of the factors must be zero
    - Thus, \(m + n = 0\) (which means \(m = -n\)) or \(m - n = 0\) (which means \(m = n\))

Counterexamples

- Given a universally quantified statement, find a single example which it is not true
- Note that this is DISPROVING a UNIVERSAL statement by a counterexample
- \(\forall x \neg R(x)\), where \(R(x)\) means “x has red hair”
  - Find one person (in the domain) who has red hair
- Every positive integer is the square of another integer
  - The square root of 5 is 2.236, which is not an integer

A note on counterexamples

- You can DISPROVE something by showing a single counter example
  - You are finding an example to show that something is not true
- You cannot PROVE something by example
- Example: prove or disprove that all numbers are even
  - Proof by contradiction: 1 is not even
  - (Invalid) proof by example: 2 is even

Permutations vs. Combinations

- Both are ways to count the possibilities
- The difference between them is whether order matters or not
- Consider a poker hand:
  - A*, 5#, 7*, 10, K
- Is that the same hand as:
  - K, 10, 7*, 5#, A*
- Does the order the cards are handed out matter?
  - If yes, then we are dealing with permutations
  - If no, then we are dealing with combinations

Permutations

- A permutation is an ordered arrangement of the elements of some set \(S\)
  - Let \(S = \{a, b, c\}\)
  - \(a, b, c\) is a permutation of \(S\)
  - \(b, c, a\) is a different permutation of \(S\)
- An \(r\)-permutation is an ordered arrangement of \(r\) elements of the set
  - A*, 5#, 7*, 10, K is a 5-permutation of the set of cards
- The notation for the number of \(r\)-permutations: \(P(n, r)\)
  - The poker hand is one of \(P(52, 5)\) permutations
Permutations

- Number of poker hands (5 cards):
  - \( P(52,5) = \frac{52!}{(52-5)!} = 311,875,200 \)
- Number of (initial) blackjack hands (2 cards):
  - \( P(52,2) = \frac{52!}{(52-2)!} = 2,652 \)
- \( r \)-permutation notation: \( P(n,r) \)
  - The poker hand is one of \( P(52,5) \) permutations
  \[
P(n,r) = n(n-1)(n-2)...(n-r+1)
  = \frac{n!}{(n-r)!}
  = \prod_{i=r}^{n} i
  \]

Permutations vs. \( r \)-permutations

- \( r \)-permutations: Choosing an ordered 5 card hand is \( P(52,5) \)
  - When people say “permutations”, they almost always mean \( r \)-permutations
    • But the name can refer to both
- Permutations: Choosing an order for all 52 cards is \( P(52,52) = 52! \)
  - Thus, \( P(n,n) = n! \)

Sample question

- How many permutations of \{a, b, c, d, e, f, g\} end with a?
  - Note that the set has 7 elements
- The last character must be a
  - The rest can be in any order
- Thus, we want a 6-permutation on the set \{b, c, d, e, f, g\}
  - \( P(6,6) = 6! = 720 \)
- Why is it not \( P(7,6) \)?

\( r \)-permutations example

- How many ways are there for 5 people in this class to give presentations?
  - There are 27 students in the class
    - \( P(27,5) = 27*26*25*24*23 = 9,687,600 \)
    - Note that the order they go in does matter in this example!

Permutation formula proof

- There are \( n \) ways to choose the first element
  - \( n-1 \) ways to choose the second
  - \( n-2 \) ways to choose the third
  - …
  - \( n-r+1 \) ways to choose the \( r \)th element
- By the product rule, that gives us: \( P(n,r) = n(n-1)(n-2)...(n-r+1) \)
Combinations

- What if order doesn’t matter?
- In poker, the following two hands are equivalent:
  - A♦, 5♥, 7♣, 10♠, K♠
  - K♠, 10♠, 7♣, 5♥, A♦

- The number of $r$-combinations of a set with $n$ elements, where $n$ is non-negative and $0 \leq r \leq n$ is:
  \[ C(n,r) = \frac{n!}{r!(n-r)!} \]

Note on combinations

- An alternative (and more common) way to denote an $r$-combination:
  \[ C(n,r) = \binom{n}{r} \]
- I’ll use $C(n,r)$ whenever possible, as it is easier to write in PowerPoint

Combinations example

- How many different poker hands are there (5 cards)?
  \[ C(52,5) = \frac{52!}{2!(52-5)!} = \frac{52!}{2!47!} = 2,598,960 \]
- How many different (initial) blackjack hands are there?
  \[ C(52,2) = \frac{52!}{2!(52-2)!} = \frac{52!}{2!50!} = \frac{52 \times 51}{2 \times 1} = 1,326 \]

Combination formula proof

- Let $C(52,5)$ be the number of ways to generate unordered poker hands
- The number of ordered poker hands is $P(52,5) = 311,875,200$
- The number of ways to order a single poker hand is $P(5,5) = 5! = 120$
- The total number of unordered poker hands is the total number of ordered hands divided by the number of ways to order each hand
- Thus, $C(52,5) = P(52,5)/P(5,5)$

- Let $C(n,r)$ be the number of ways to generate unordered combinations
- The number of ordered combinations (i.e. $r$-permutations) is $P(n,r)$
- The number of ways to order a single one of those $r$-permutations $P(r,r)$
- The total number of unordered combinations is the total number of ordered combinations (i.e. $r$-permutations) divided by the number of ways to order each combination
- Thus, $C(n,r) = P(n,r)/P(r,r)$
Combinations

\[ C(n, r) = \frac{P(n, r)}{P(r, r)} = \frac{n!}{(n-r)!} \]

Recursive Definition:
- \( C(n, r) = 0 \) if \( n < r \)
- \( C(n, 1) = n \) if \( n > 0 \)
- \( C(n, r) = C(n-1, r) + C(n-1, r-1) \) if \( n > 0, r > 1 \)

Binomial Coefficients

\[ (x+y)^n = C(n,0)x^n + C(n,1)x^{n-1}y + \ldots + C(n,i)x^{n-i}y^i + \ldots + C(n,n)y^n \]

Proof: Induction on \( n \)

Claim: \( (x+y)^n = C(n,0)x^n + C(n,1)x^{n-1}y + \ldots + C(n,n)y^n \)

Lemma 1: \( C(n, i) = C(n, n-i) \)

Lemma 2: \( C(n, i) = C(n-1, i) + C(n-1, i-1) \)

Bit strings

- How many bit strings of length 10 contain:
  a) exactly four 1’s?
    - Find the positions of the four 1’s
    - Does the order of these positions matter?
      - No!
      - Positions 2, 3, 5, 7 is the same as positions 7, 5, 3, 2
    - Thus, the answer is \( C(10,4) = 210 \)
  b) at most four 1’s?
    - There can be 0, 1, 2, 3, or 4 occurrences of 1
    - Thus, the answer is: \( \sum_{i=0}^{4} C(10,i) \)
    - \( = 1 + 10 + 45 + 120 + 210 \)
    - \( = 386 \)
  c) at least four 1’s?
    - There can be 4, 5, 6, 7, 8, 9, or 10 occurrences of 1
    - Thus, the answer is: \( \sum_{i=4}^{10} C(10,i) \)
    - \( = 210 + 252 + 210 + 120 + 45 + 10 + 1 \)
    - \( = 847 \)
  d) an equal number of 1’s and 0’s?
    - Thus, there must be five 0’s and five 1’s
    - Find the positions of the five 1’s
    - Thus, the answer is \( C(10,5) = 252 \)

Circular seatings

- How many ways are there to sit 6 people around a circular table, where seatings are considered to be the same if they can be obtained from each other by rotating the table?
  - First, place the first person in the north-most chair
    - Only one possibility
  - Then place the other 5 people
    - There are \( P(5,5) = 5! = 120 \) ways to do that
  - By the product rule, we get \( 1*120 = 120 \)
  - Alternative means to answer this:
    - There are \( P(6,6) = 720 \) ways to seat the 6 people around the table
    - For each seating, there are 6 "rotations" of the seating
    - Thus, the final answer is \( 720/6 = 120 \)

Horse races

- How many ways are there for 4 horses to finish if ties are allowed?
  - Note that order does matter!
  - Solution by cases
    - No ties
      - The number of permutations is \( P(4,4) = 4! = 24 \)
    - Two horses tie
      - There are \( C(4,2) = 6 \) ways to choose the two horses that tie
      - There are \( P(3,3) = 6 \) ways for the "groups" to finish
      - A "group" is either a single horse or the two tying horses
        - By the product rule, there are \( 6*6 = 36 \) possibilities for this case
    - Two groups of two horses tie
      - There are \( C(4,2) = 6 \) ways to choose the two winning horses
      - The other two horses tie for second place
        - By the product rule, there are \( 2*2 = 4 \) possibilities for this case
    - Three horses tie with each other
      - There are \( C(4,3) = 4 \) ways to choose the two horses that tie
      - There are \( P(2,2) = 2 \) ways for the "groups" to finish
        - By the product rule, there are \( 4*2 = 8 \) possibilities for this case
    - All four horses tie
      - There is only one combination for this
        - By the sum rule, the total is \( 24+36+6+8+1 = 75 \)
How to enumerate all subsets?

- A set of n elements has $2^n$ subsets
- An n-bit integer has $2^n$ values, each value corresponds to one subset.
- Can we use the values of an n-bit integer for printing out each subset?

```java
class EnumerateDemo {
    public static void main(String[] args) {
        enumerateSubsets(10);
    }
    public static void enumerateSubsets (int n) {
        // Pre: n < 32
        for (int x = 0; x < (1 << n); x++) {
            System.out.println("{ ");
            for (int j = 1; j <= n; j++) if (x & (1 << (j-1)) != 0)
                System.out.println(j + ", ");
            System.out.println("} 
");
        }
    }
}
```

How to enumerate all permutations of n numbers?

- Let $N$ be the set \{1, 2, 3, ..., n\} and 0 < k < n.
- Any subset $X$ of $N, |X| = k$, is a k-combination of $N$.
- There are C(k, n) = n!(n-k)! such combinations.
- We may use the idea in enumerate5Combinations for general k.

```java
public static void enumerateCombinations (int k, int n) {
    int x[] = new int[100];    // k <= 100
    for (int j = 0; j < k; j++) x[j] = j+1;
    while (true) {
        printCombination(x, k);
        if (nextCombination(x, k, n) == false) break;
    }
}
```

```java
public static boolean nextCombination (int x[], int k, int n) {
    for (int j = k-1; j >= 0; j--) if (x[j] <= (n - k + j)) {
        x[j]++;
        for (int i = 1; i < k - j;  i++) x[i+j] = x[j]+i;
        return true; }
    return false;
}
```

How to enumerate k-combinations of n numbers?

- Let $N$ be the set \{1, 2, 3, ..., n\} and 0 < k < n.
- Any subset $X$ of $N, |X| = k$, is a k-combination of $N$.
- There are C(k, n) = n!(n-k)! such combinations.
- How to enumerate all of them? E.g., if n = 9, k = 5, how can we enumerate them from \{1, 2, 3, 4, 5\} to \{5, 6, 7, 8, 9\}?
- If k is fixed, we can use k-nested loops:

```java
public static void enumerate5Combinations (int n) {
    // Pre: k = 5
    for (int x1 = 1; x1 <= (n - 4); x1++)
        for (int x2 = x1+1; x2 <= (n - 3); x2++)
            for (int x3 = x2+1; x3 <= (n - 2); x3++)
                for (int x4 = x3+1; x4 <= (n - 1); x4++)
                    for (int x5 = x4+1; x5 <= n; x5++) {
                        System.out.println("{ ");
                        System.out.println(x1+" , " +x2+" , " +x3+" , " +x4+" , " +x5);
                        System.out.println("} 
");
                    }
}
```

```java
public static void printCombination(int x[], int k) {
    int i;
    System.out.println("{ ");
    for (i = 0; i < k; i++)
        System.out.println(x[i] + " , ");
    System.out.println("} 
");
}
```

```java
public static boolean nextCombination (int x[], int k, int n) {
    for (int j = k-1; j >= 0; j--) if (x[j] <= (n - k + j)) {
        x[j]++;
        for (int i = 1; i < k - j;  i++) x[i+j] = x[j]+i;
        return true; }
    return false;
}
```
How to enumerate all permutations of n numbers?

- Let \( N \) be the set \{ 1, 2, 3, ..., n \}.
- There are \( n! \) different permutations on \( N \).
- How to enumerate all of them? E.g., if \( n = 5 \), how can we enumerate them from \{ 1, 2, 3, 4, 5 \} to \{ 5, 4, 3, 2, 1 \}?
- If \( n \) is fixed, we can use \( n \)-nested loops:

```java
public static void enumerate5Permutations () {
    // Pre: n = 5
    for (int x1 = 1; x1 <= 5; x1++)
        for (int x2 = 1; x2 <= 5; x2++) if (x1 != x2)
            for (int x3 = 1; x3 <= 5; x3++) if (x3 != x1 && x3 != x2)
                for (int x4 = 1; x4 <= 5; x4++) if (x4 != x1 && x4 != x2 && x4 != x3)
                    for (int x5 = 1; x5 <= 5; x5++)
                        if (x5 != x1 && x5 != x2 && x5 != x3 && x5 != x4) {
                            System.out.println("{ " + x1 + ", " + x2 + ", " + x3 + ", " + x4 + ", " + x5 + "}
                        }
    }
```

How to enumerate \( k \)-combinations of \( n \) numbers?

- Let \( N \) be the set \{ 1, 2, 3, ..., n \} and \( 0 < k < n \).
- Any subset \( X \) of \( N \), \(|X| = k\), is a \( k \)-combination of \( N \).
- There are \( C(k, n) = \frac{n!}{k!(n-k)!} \) such combinations.
- How to enumerate all of them? E.g., if \( n = 9, k = 5 \), how can we enumerate them from \{ 1, 2, 3, 4, 5 \} to \{ 5, 6, 7, 8, 9 \}?
- If \( k \) is fixed, we can use \( k \)-nested loops:

```java
public static void enumerate5Combinations (int n) {
    // Pre: k = 5
    for (int x1 = 1; x1 <= (n - 4); x1++)
        for (int x2 = x1+1; x2 <= (n - 3); x2++)
            for (int x3 = x2+1; x3 <= (n - 2); x3++)
                for (int x4 = x3+1; x4 <= (n - 1); x4++)
                    for (int x5 = x4+1; x5 <= n; x5++) {
                        System.out.println("{ " + x1 + ", " + x2 + ", " + x3 + ", " + x4 + ", " + x5 + "}
                    }
    }
```
