1. In each situation, write a recurrence relation, including base case(s), that describes the recursive structure of the problem. You do not need to solve the recurrence.

(a) At a party with \( n \) people, everyone shakes hands once with everybody else. Let \( H(n) \) be the total number of handshakes among the \( n \) guests. Write a recurrence for \( H(n) \).

**Solution:** Since each person shakes hands with \( n - 1 \) other people, the recurrence is

\[
H(n) = H(n-1) + (n-1), \quad \text{with } H(2) = 1
\]

or equivalently,

\[
H(n) = H(n-1) + (n-1), \quad \text{with } H(1) = 0.
\]

(b) Let \( B(n) \) be the number of length \( n \) bit sequences that have no three consecutive 0’s. Write a recurrence for \( B(n) \).

**Solution:** Any bit string that has no 000 must have a 1 in at least one of the first three positions. Break up all bit strings avoiding 000 by when the first 1 occurs. That is, each bit string of length \( n \) avoiding 000 falls into exactly one of these cases:

(i) 1 followed by any bit string of length \( n - 1 \) avoiding 000.
(ii) 01 followed by any bit string of length \( n - 2 \) avoiding 000.
(iii) 001 followed by any bit string of length \( n - 3 \) avoiding 000.

Therefore, the recurrence is

\[
B(n) = B(n-1) + B(n-2) + B(n-3), \quad \text{with } B(0) = 1, B(1) = 2, B(2) = 4
\]

or equivalently,

\[
B(n) = B(n-1) + B(n-2) + B(n-3), \quad \text{with } B(1) = 2, B(2) = 4, B(3) = 7.
\]

(c) Say you are tiling a \( 2 \times n \) rectangle with L-shaped tiles of area 3 (trominoes).

To tile the rectangle is to cover it with tiles so that no tiles overlap, no tiles are hanging off the edge of the rectangle, and every space on the rectangle is covered by some tile. Let \( T(n) \) denote the number of ways to tile the rectangle. Write a recurrence for \( T(n) \).

**Solution:** First, notice that if \( n \) is not a multiple of 3, there will be no way to tile the rectangle. Now if \( n \) is a multiple of 3, then there are two ways to tile the first three columns:

The rest of the tiling is a tiling of a \( 2 \times (n - 3) \) rectangle, of which there are \( T(n - 3) \). Therefore, the recurrence is

\[
T(n) = \begin{cases} 
2T(n-3), & \text{with } T(0) = 1 \quad \text{if } n \equiv 0 \pmod{3} \\
0, & \text{else.}
\end{cases}
\]
We could instead use the base case $T(3) = 2$.
The following recurrences are also equivalent:

$$T(n) = 2T(n - 3), \text{ with } T(0) = 1, T(1) = 0, T(2) = 0.$$  

$$T(n) = 2T(n - 3), \text{ with } T(1) = 0, T(2) = 0, T(3) = 2.$$  

(d) A ternary string is like a binary string except it uses three symbols, 0, 1, and 2. For example, 12210021 is a ternary string of length 8. Let $T(n)$ be the number of ternary strings of length $n$ with the property that there is never a 2 appearing anywhere after a 0. For example, 12120110 has this property but 10120112 does not. Write a recurrence for $T(n)$.

**Solution:** This is similar to an example from class with binary strings. Any such ternary string of length $n$ starts with 0, 1, or 2:

0...

1...

2...

In the first case, since we cannot have 2 anywhere after a 0, the dots represent a binary string, that is a string of length $n - 1$ containing all 0s and 1s.

In the last two cases, the dots represent any ternary string of length $n - 1$ having the property that there is never a 2 anywhere after a 0.

For the base case, note that any ternary string of length one satisfies the required property. Therefore, our recurrence is:

$$T(1) = 3$$

$$T(n) = 2T(n - 1) + 2^{n - 1}$$

2. (a) Suppose a function $g$ is defined by the following recursive formula, where $n$ is a positive integer.

$$g(n) = 3g(n - 1), \quad g(1) = 9$$

Use the guess-and-check method to get a closed-form formula for $g(n)$. That is, guess a formula for $g(n)$ and use induction to prove that your guess is correct.

**Solution:** The following table suggests that a formula for $g(n)$ may be $g(n) = 3^{n+1}$.

<table>
<thead>
<tr>
<th>n</th>
<th>g(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>81</td>
</tr>
<tr>
<td>4</td>
<td>243</td>
</tr>
<tr>
<td>5</td>
<td>729</td>
</tr>
</tbody>
</table>

We can prove this guess by induction on $n$. For the base case, when $n = 1$, we have $g(1) = 9$ as given by the recurrence and $3^{1+1} = 9$, so the formula holds. Now let $n$ be a nonnegative integer. Assume that $g(n) = 3^{n+1}$ as our inductive hypothesis. Then $g(n + 1) = 3g(n)$ by the recursive formula, and so $g(n + 1) = 3g(n) = 3 \cdot 3^{n+1} = 3^{n+2}$ by applying the inductive hypothesis. Thus, our formula is correct.
(b) Suppose a function \( f \) is defined by the following recursive formula, where \( n \) is a positive integer.

\[
f(n) = f(n - 2) + 4, \quad f(1) = 1, \quad f(2) = 3
\]

Use the guess-and-check method to get a closed-form (i.e. not recursive) formula for \( f(n) \). That is, guess a formula for \( f(n) \) and use induction to prove that your guess is correct.

**Solution:** The following table suggests that a formula for \( f(n) \) may be \( f(n) = 2^n - 1 \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(n) )</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

We can prove this guess by induction on \( n \). For the base cases, when \( n = 1 \), we have \( f(1) = 1 \) as given by the recurrence and \( 2(1) - 1 = 1 \), so the formula holds. Similarly, when \( n = 2 \), we have \( f(2) = 3 \) as given by the recurrence and \( 2(2) - 1 = 3 \), so again the formula holds. Now let \( n \) be an integer greater than 1. Assume by strong induction that \( f(k) = 2k - 1 \) for any value of \( k \) with \( 1 \leq k \leq n \). Then \( f(n + 1) = f(n - 1) + 4 \) by the recursive formula, and applying the strong inductive hypothesis with \( k = n - 1 \) gives \( f(n + 1) = f(n - 1) + 4 = 2(n - 1) - 1 + 4 = 2(n + 1) - 1 \). Thus, our formula is correct.

3. (a) Suppose a function \( g \) is defined by the following recursive formula, where \( n \) is a positive integer.

\[
g(n) = 3g(n - 1), \quad g(1) = 9
\]

Use the unraveling method to get a closed-form formula for \( g(n) \). This should agree with your answer to question 2a.

**Solution:**

\[
g(n) = 3g(n - 1) \\
= 3(3g(n - 2)) \\
= 3(3(3g(n - 3))) \\
\vdots \\
= 3^k g(n - k) \\
\vdots \\
= 3^{n-1} g(1) \\
= 3^{n-1} \cdot 9 \\
= 3^{n+1}
\]

Here we are letting \( k = n - 1 \) because that is the value of \( k \) that reaches the base case \( g(1) = 9 \). As expected, the formula agrees with the answer to question 2a.

(b) Suppose a function \( d \) is defined by the following recursive formula, where \( n \) is a positive integer.

\[
d(n) = d(n - 1) + 2n + 1, \quad d(1) = 3
\]

Use the unraveling method to get a closed-form formula for \( d(n) \).
Solution:

\[ d(n) = d(n-1) + 2n + 1 \]

\[ = \left( d(n-2) + 2(n-1) + 1 \right) + 2n + 1 \]

\[ = \left( \left( d(n-3) + 2(n-2) + 1 \right) + 2(n-1) + 1 \right) + 2n + 1 \]

\[ \vdots \]

\[ = d(n-k) + 2 \left( (n-k+1) + \cdots + (n-1) + n \right) + k \]

\[ \vdots \]

\[ = d(1) + 2 \left( 2 + \cdots + (n-1) + n \right) + (n-1) \]

\[ = 3 + 2 \left( n(n+1)/2 - 1 \right) + (n-1) \]

\[ = 3 + n(n+1) - 2 + n - 1 \]

\[ = n(n+2) \]

Here we are letting \( k = n-1 \) because that is the value of \( k \) that reaches the base case \( d(1) = 3 \).

(c) Suppose a function \( g \) is defined by the following recursive formula, where \( n \) is a positive integer.

\[ g(n) = 3n \cdot g(n-1), \quad g(1) = 1 \]

Use the unraveling method to get a closed-form formula for \( g(n) \).

Solution:

\[ g(n) = 3n \cdot g(n-1) \]

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

\[ = 3n(3(n-1)(3(n-2)g(n-3))) \]

\[ \vdots \]

\[ = 3^k n \cdot (n-1) \cdot \cdots \cdot (n-k+1) \cdot g(n-k) \]

\[ \vdots \]

\[ = 3^{n-1} \cdot n! \cdot g(1) \]

\[ = 3^{n-1} \cdot n! \]

(d) Suppose a function \( g \) is defined by the following recursive formula, where \( n \) is a positive integer.

\[ g(n) = 4 + g(n-2), \quad g(1) = 1, \quad g(2) = 2 \]

Use the unraveling method to get a closed-form formula for \( g(n) \).

Solution: We show \( g(n) = 2n - 1 \).

\[ g(n) = 4 + g(n-2) \]

\[ = 2 \cdot 4 + g(n-4) \]

\[ \vdots \]

\[ = 4 \cdot k + g(n-2k) \]
Now, let’s consider the two cases, \( n \) is even or \( n \) is odd. When \( n \) is even, \( n = 2l \) for some positive integer \( l \). Then, we will subtract 2 from \( n \) \( l-1 \) times, this yields
\[
g(n) = 4l + g(2) = 4\left(\frac{n}{2} - 1\right) + 3 = 2n - 1
\]
. When \( n = 2l + 1 \), we will subtract 2 from \( n \) \( \frac{n-1}{2} \) times to get
\[
g(n) = 4\left(\frac{n-1}{2}\right) + 1 = 2n - 1.
\]

4. The following algorithm (Rosen pg. 363) is a recursive version of linear search, which has access to a global list of distinct integers \( a_1, a_2, \ldots, a_n \).

\textbf{procedure} search\((i, j, x : i, j, x \text{ integers, } 1 \leq i \leq j \leq n)\)

1. \textbf{if} \( a_i = x \) \textbf{then}
2. \hspace{1em} \textbf{return} \( i \)
3. \textbf{else if} \( i = j \) \textbf{then}
4. \hspace{1em} \textbf{return} \( 0 \)
5. \textbf{else}
6. \hspace{1em} \textbf{return} search\((i + 1, j, x)\)

(a) Prove that this algorithm correctly solves the searching problem when called with parameters \( i = 1 \) and \( j = n \). That is, prove that it returns the location of the target value \( x \) in the list, and returns 0 if the target is not present in the list.

\textbf{Solution:} Base Case: When \( n = 1 \), the algorithm checks if \( x = a_1 \) then returns 1 if so and 0 otherwise.

IH: Assume the algorithm returns the position of \( x \) if it is in an array of size \( n = k - 1 \).

IS: For an array of size \( n = k \), \( j = k \) and we start at \( i = 1 \). This yields a simple check to see if \( x = a_1 \). If so, we return 1, otherwise we recurse on the array \( a_2, \ldots, a_k \) which will be correct by the IH. Notice, that the induction is “backwards” since we are moving from left to right. That is okay since we induct on the size of the array.

Picture: \( a_1 a_2 \ldots a_k \). Is \( x = a_1 \)? If so, return 1. Else, search\((2, k, x)\). That is, we search on the list \( a_2 \ldots a_k \).

(b) Let \( T(n) \) be the running time of this algorithm. Write a recurrence relation that \( T(n) \) satisfies.

\textbf{Solution:} We start with a list of size \( n \) and with each recursive call, we search a list of size one smaller than before. All the other work in the algorithm is just basic operations, which take a constant amount of time. In the base case, which is for a list of size 1, we also do a constant amount of work in lines 1 through 4. Therefore, the recurrence is
\[
T(n) = T(n - 1) + c, \text{ with } T(1) = d,
\]
where \( c \) and \( d \) are constants. Note that we cannot determine exactly how much time (in seconds or minutes) these basic operations take, so \( c \) and \( d \) must be arbitrary constants.

(c) Solve the recurrence found in part (b) and write the solution in \( \Theta \) notation.

\textbf{Solution:} We will use the unravel method since we don’t have particular constants to guess a pattern.
\[ T(n) = T(n-1) + c \]
\[ = T(n-2) + c + c \]
\[ = T(n-3) + c + c + c \]
\[ \vdots \]
\[ = T(n-k) + ck \]
\[ \vdots \]
\[ = T(1) + c(n-1) \]
\[ = d + c(n-1) \]
\[ = cn + (d - c) \]

Here we are letting \( k = n - 1 \) because that is the value of \( k \) that reaches the base case \( T(1) = d \). In \( \Theta \) notation, we have \( \Theta(cn + (d - c)) = \Theta(n) \) since \( c \) and \( d \) are constants. Therefore, this algorithm is linear in the size of the list we are searching.

5. The following algorithm determines whether a word is a palindrome, that is, if the word is the same read left to right as right to left. An example of a palindrome is *raccoar*.

**procedure** Palindrome\((s_1s_2s_3\ldots s_n)\)

1. if \( n = 0 \) or \( n = 1 \) then return true
2. if \( s_1 = s_n \) then return Palindrome\((s_2\ldots s_{n-1})\)
3. else return false

**Note:** Writing \( s_1s_2s_3\ldots s_n \) denotes a string of length \( n \) whose characters are \( s_1, s_2, s_3, \) etc. These characters are being concatenated (not multiplied) to form a string.

(a) Prove that this algorithm is correct, i.e., that it returns true if and only if \( s_1s_2s_3\ldots s_n \) is a palindrome.

**Solution:**

Base Case: When \( n = 1 \) and \( n = 0 \), the string is a palindrome by definition and the algorithm is correct.

IH: Assume the algorithm is correct on inputs of size 0, 1, . . . , \( k - 1 \).

IS: Here we show the algorithm is correct on input of size \( k \). If \( a_1 \neq a_k \) then the string is not a palindrome and we return false. Otherwise, we recurse on the string \( a_2\ldots a_{k-1} \) and the algorithm is correct by the IH.

Picture: input = \( a_1\ldots a_k \). If, \( a_1 = a_k \), we repeat on the substring \( \ldots \). Else, there is no way this string is a palindrome and we return false.

(b) Let \( C(n) \) be the number of times this algorithm compares two letters \( s_i \) and \( s_j \) for some \( i, j \). Write a recurrence relation that \( C(n) \) satisfies.

**Solution:** With each recursive call, one comparison is done in line 2 (between the first and last letter of the word), and then Palindrome is called again with an input having length two smaller than before. No comparisons are done in the base cases in line 1. Therefore, the recurrence is

\[ C(n) = C(n - 2) + 1, \text{ with } C(0) = 0, C(1) = 0. \]

(c) Solve the recurrence found in part (b) and write the solution in \( \Theta \) notation.

**Solution:** To find an exact formula for \( C(n) \) we will have to consider whether \( n \) is even or odd, since the recursion expresses \( C(n) \) in terms of \( C(n - 2) \). This means that when \( n \) is even, we will
eventually hit the base case for \( n = 0 \) and when \( n \) is odd, we will eventually hit the base case for \( n = 1 \). Unraveling the recurrence gives

\[
C(n) = C(n - 2) + 1 \\
= C(n - 4) + 2 \\
= C(n - 6) + 3 \\
\vdots \\
= C(n - k) + \frac{k}{2} \\
\vdots \\
= \begin{cases} 
C(0) + \frac{n}{2} & \text{if } n \text{ is even} \quad \text{(letting } k=n) \\
C(1) + \frac{n-1}{2} & \text{if } n \text{ is odd} \quad \text{(letting } k=n-1) \\
\end{cases}
\]

\[
= \begin{cases} 
\frac{n}{2} & \text{if } n \text{ is even} \\
\frac{n-1}{2} & \text{if } n \text{ is odd} \\
\end{cases}
\]

\[
= \lfloor \frac{n}{2} \rfloor
\]

In \( \Theta \) notation, this is \( \Theta(n) \), or linear time in the number of letters in the input word.

6. Here we look at the ratio \( f(5n)/f(n) \).

(a) \( f(5n)/f(n) = \frac{5^{5n}}{5^n} = 5 \)

(b) \( \frac{4(5n)^3}{4n^3} = 125 \)

(c) \( \frac{5^n}{3^n} = \left( \frac{243}{3} \right)^n = 81^n \)

(d) \( \frac{5n}{5n-2} = \frac{5n}{5n-2} \)

(e) \( \frac{5n}{n!} = 5n \cdot (5n - 1) \cdot \cdots \cdot (n + 1) \)

(f) \( \frac{(5n)^n}{n^n} = \frac{5^n \cdot n^n}{n^n} = 5^n n^4_n \)

7. Prove the following \( f(n) \in O(g(n)) \) by providing witnesses, \( k, C \) such that \( f(n) \leq C \cdot g(n) \forall n \geq k \).

(a) \( n^5 + 3n^2 + 13 \in O(n^5) \)

(b) \( n \log^3 n + 5n \log n \in O(n \log^3 n) \)

(c) \( 4n \log n + n^2 \log(\log(n)) \in O(n^2 \log(\log(n))) \)

solution:

(a) \( n^5 + 3n^3 + 13 \leq n^5 + 3n^5 + 13n^5 = 17n^5 \). Therefore, \( k = 1 \) and \( C = 17 \).

(b) \( n \log^3 n + 5n \log n \leq n \log^3 n + 5n \log^3 n = 6n \log^3 n \). Therefore, \( k = 1 \) and \( C = 6 \).

(c) \( 4n \log n + n^2 \log(\log(n)) \leq 4n^2 + n^2 \log(\log(n)) \leq 4n^2 \log(\log(n)) + n^2 \log(\log(n)) = 5n^2 \log(\log(n)) \).

Then, \( k = 1 \) and \( C = 5 \).