5.3 Performance analysis of recursive algorithms [Section 2.4]

To analyze a recursive algorithm, we need to:

1. Identify the basic operation in the recursive algorithm.

2. Set up a recurrence relation (function), with an initial condition, for the number of times the basic operation is executed. The result is a recursive function $M$ of the input size.
   
   Example:
   
   \[
   M(n) = \begin{cases} 
   0 & \text{for } n = 1, \\
   M\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + 1 & \text{for } n > 1.
   \end{cases}
   \]

3. Solve the recurrence relation, i.e., remove the recursion from the definition of $M$. In other words, express $M$ as a non-recursive function of the input size.
   
   Example:
   
   $M(n) = \log n$.
   
   There are two ways to solve a recurrence relation:

   (a) Use backward substitutions.
   
   (b) Make a guess and use mathematical induction to prove it.

4. Find the order of growth of $M$ using $O$ and/or $\Theta$.

---

5. Divide and conquer

Example:
The factorial algorithm.
Basic operation: multiplication.

**Algorithm 1**: A recursive algorithm that computes the factorial of a positive integer.

```plaintext
Algorithm Factorial(n)
  if n = 0 then
    return 1;
  else
    return Factorial(n - 1) * n;
end
```

A recurrence relation for the basic operation count:

\[
M(n) = \begin{cases} 
0 & \text{for } n = 0, \\
M(n - 1) + 1 & \text{for } n > 0.
\end{cases}
\]

Backward substitutions:

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

We can use mathematical induction to prove our solution (easy).
\( M(n) \in \Theta(n) \).
5. Divide and conquer

Example:
Binary search.

Algorithm 2: Recursive binary search for an item K in an array \( A[l..r] \).

```plaintext
Algorithm BinarySearchRec(A[], l, r, K)
if l > r then
  return -1
else
  m := \left\lfloor \frac{l + r}{2} \right\rfloor;
  if K = A[m] then
    return m;
  else
    if K < A[m] then
      return BinarySearchRec(A[], l, m - 1);
    else
      return BinarySearchRec(A[], m + 1, r);
  end
end
```

Basic operation: comparison.
A recurrence relation for the basic operation count:

\[
M(n) = \begin{cases} 
1 & \text{for } n = 1, \\
M(\lfloor \frac{n}{2} \rfloor) + 1 & \text{for } n > 1.
\end{cases}
\]

Since we divide by 2, we assume that \( n = 2^k \).

\[
M(2^k) = \begin{cases} 
1 & \text{for } k = 0, \\
M(2^{k-1}) + 1 & \text{for } k > 0.
\end{cases}
\]
5. Divide and conquer

Backward substitutions:

\[
\begin{align*}
M(2^k) &= [M(2^{k-2}) + 1] + 1 \\
&= [M(2^{k-3}) + 1] + 2 \\
&= \ldots \\
&= M(2^{k-k}) + k \\
&= 1 + k \\
&= M(2^{0}) + k \\
&= 1 + \log n
\end{align*}
\]

We can use mathematical induction to prove our solution (easy).

\( M(n) \in O(\log n) \).