Chapter 2: Fundamentals of the Analysis of Algorithm Efficiency

2.1. Analysis Framework

Issues:
- Correctness
- Time efficiency
- Space efficiency
- Optimality

Note: An algorithm is said to be asymptotically optimal if, roughly speaking, for large inputs it performs at worst a constant factor worse than the best possible algorithm. A consequence of an algorithm being asymptotically optimal is that, for large enough inputs, no algorithm can outperform it by more than a fixed constant factor.

Approaches:
- Theoretical analysis
- Empirical analysis

1. Measuring an Input’s Size

Time efficiency is analyzed by determining the number of repetitions of the basic operation as a function of input size

- Influenced by the data representation, e.g. matrix
- Influenced by the operations of the algorithm, e.g. spell-checker
- Influenced by the properties of the objects in the problem, e.g. checking if a given integer is a prime number.

Here are some examples:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Size of input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find x in an array</td>
<td>The number of the elements in the array</td>
</tr>
<tr>
<td>Multiply two matrices</td>
<td>The dimensions of the matrices</td>
</tr>
<tr>
<td>Sort an array</td>
<td>The number of elements in the array</td>
</tr>
<tr>
<td>Traverse a binary tree</td>
<td>The number of nodes</td>
</tr>
<tr>
<td>Solve a system of linear equations</td>
<td>The number of equations, or the number of the unknowns, or both</td>
</tr>
</tbody>
</table>
2. Units for Measuring Running Time

Using standard time units (e.g. milliseconds) is not appropriate (Why?)
Counting all operations in an algorithm is not appropriate either (Why?)

The approach: identify and count the basic operation(s) in the algorithm.

Basic operation:
- Applied to all input items in order to carry out the algorithm.
- Contributes most towards the running time of the algorithm.

Typical basic operations for some problems are the following:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find x in an array</td>
<td>Comparison of x with an entry in the array</td>
</tr>
<tr>
<td>Multiplying two matrices</td>
<td>Multiplication of two real numbers</td>
</tr>
<tr>
<td>with real entries</td>
<td></td>
</tr>
<tr>
<td>Sort an array of numbers</td>
<td>Comparison of two array entries plus moving elements in the array</td>
</tr>
<tr>
<td>Traverse a tree</td>
<td>Traverse an edge</td>
</tr>
</tbody>
</table>

The work done by an algorithm, i.e. its complexity, is determined by the number of the basic operations necessary to solve the problem and by the execution time of the basic operation.

Let
\[
T(n): \text{running time} \\
c_{\text{op}}: \text{execution time for basic operation} \\
C(n): \text{number of times basic operation is executed}
\]

Then we have:
\[
T(n) \approx c_{\text{op}} C(n)
\]

Types of formulas for basic operation count
- Exact formula
  e.g., \( C(n) = n(n-1)/2 \)
- Formula indicating order of growth with specific multiplicative constant
  e.g., \( C(n) \approx 0.5 n^2 \)
- Formula indicating order of growth with unknown multiplicative constant
  e.g., \( C(n) \approx cn^2 \)

Example:
Let \( C(n) = 3n(n-1) \approx 3n^2 \)

Suppose we double the input size. How much longer the program will run?
\[
T(2n) / T(n) = (c_{op} C(2n)) / (c_{op} C(n)) = 3(2n)^2 / 3n^2 = 4
\]

**Note that:**
- we did not need to know \( c_{op} \)
- the coefficient ‘3’ did not play a role in obtaining the answer.

The coefficients and other constants in the expression for the number of operations do not matter a lot when determining the running time of an algorithm for a large input size. Instead, we concentrate on the order of growth of \( C(n) \).

### 3. Orders of Growth

Consider the table below:

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \log_2 n )</th>
<th>( n )</th>
<th>( n \log_2 n )</th>
<th>( n^2 )</th>
<th>( n^3 )</th>
<th>( 2^n )</th>
<th>( n! )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3.3</td>
<td>( 10^1 )</td>
<td>( 3.3 \cdot 10^1 )</td>
<td>( 10^2 )</td>
<td>( 10^3 )</td>
<td>( 10^3 )</td>
<td>( 3.6 \cdot 10^6 )</td>
</tr>
<tr>
<td>( 10^2 )</td>
<td>6.6</td>
<td>( 10^2 )</td>
<td>( 6.6 \cdot 10^2 )</td>
<td>( 10^4 )</td>
<td>( 10^5 )</td>
<td>( 1.3 \cdot 10^{30} )</td>
<td>( 9.3 \cdot 10^{157} )</td>
</tr>
<tr>
<td>( 10^3 )</td>
<td>10</td>
<td>( 10^3 )</td>
<td>( 1.0 \cdot 10^4 )</td>
<td>( 10^6 )</td>
<td>( 10^9 )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>( 10^4 )</td>
<td>13</td>
<td>( 10^4 )</td>
<td>( 1.3 \cdot 10^5 )</td>
<td>( 10^8 )</td>
<td>( 10^{12} )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>( 10^5 )</td>
<td>17</td>
<td>( 10^5 )</td>
<td>( 1.7 \cdot 10^6 )</td>
<td>( 10^{10} )</td>
<td>( 10^{15} )</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>( 10^6 )</td>
<td>20</td>
<td>( 10^6 )</td>
<td>( 2.0 \cdot 10^7 )</td>
<td>( 10^{12} )</td>
<td>( 10^{18} )</td>
<td>( )</td>
<td>( )</td>
</tr>
</tbody>
</table>

**Table 2.1.** Values (some approximate) of several functions important for analysis of algorithms

When we increase \( n \) by an order of 6, the value of the function \( \log_2 n \) is increased just 6 times, while the value of \( n^3 \) is increased by an order of 18. This illustrates how important function growth is for analyzing the complexity of an algorithm. Function growth is discussed in more details in Section 2.2.

### 4. Worst-case, Best-case, and Average-case Efficiencies

For some algorithms efficiency depends not only of the input size, but also on the specifics of the input.

Consider for example sequential search when the item to be sought is the first element and when it is the last element. In the first case, the number of operations is a constant, it does not depend on the size of the file. In the second case the number of operations is proportional to the number of the elements in the file.
We define

- **Worst case**: \( W(n) \) – maximum over inputs of size \( n \)
- **Best case**: \( B(n) \) – minimum over inputs of size \( n \)
- **Average case**: \( A(n) \) – “average” over inputs of size \( n \)

**Notes about the average case:**
- Number of times the basic operation will be executed on typical input
- NOT the average of worst and best case
- Expected number of basic operations repetitions considered as a random variable under some assumption about the probability distribution of all possible inputs of size \( n \)

**Exercise: Sequential search**

*Problem:* Given a list of \( n \) elements and a search key \( K \), find an element equal to \( K \), if any.

*Algorithm:* Scan the list and compare its successive elements with \( K \) until either a matching element is found (*successful search*) or the list is exhausted (*unsuccessful search*)

- **Worst case**: \( W(n) \) \( \approx \) \( n \)
- **Best case**: \( B(n) \) \( \approx \) \( 1 \)
- **Average case**: \( A(n) \) \( \approx \) \( n/2 \)