Chap 3.3 - The Complexity of Algorithms

• Given an algorithm, how efficient is this algorithm for solving a problem given input of a particular size?
  – How much time does this algorithm use to solve a problem?
  – How much computer memory does this algorithm use to solve a problem?

• time complexity - analyze the time the algorithm uses to solve the problem given input of a particular size

• space complexity - analyze the computer memory the algorithm uses to solve the problem, given input of a particular size

The Complexity of Algorithms

• In this course, focus on time complexity.
• Measure time complexity in terms of the number of operations an algorithm uses
• Use big-\(O\) and big-Theta notation to estimate the time complexity
• Is it practical to use this algorithm to solve problems with input of a particular size?
• Compare the efficiency of different algorithms for solving the same problem.
Time Complexity

• For time complexity, determine the number of operations, such as comparisons and arithmetic operations (addition, multiplication, etc.).
• Ignore minor details, such as the “house keeping” aspects of the algorithm.
• Focus on the worst-case time complexity of an algorithm. Provides an upper bound.
• More difficult to determine the average case time complexity of an algorithm (average number of operations over all inputs of a particular size)

Worst-Case Complexity of Linear Search

Example: Describe the time complexity of the algorithm for finding the maximum element in a finite sequence.

```
procedure max(a_1, a_2, …, a_n: integers)
    max := a_1
    for i := 2 to n
        if max < a_i then max := a_i
    return max \{max is the largest element\}
```

Solution: Count the number of comparisons.

Average-Case Complexity of Linear Search

Example: Describes the average performance of the linear search algorithm.

Solution: Assume the element is in the list and that the possible positions are equally likely.

```
procedure linear search(x:integer, a_1, a_2, …, a_n: distinct integers)
    i := 1
    while (i ≤ n and x ≠ a_i)
        i := i + 1
    if i ≤ n then location := i
    else location := 0
    return location \{location is the subscript of the term that equals x, or is 0 if x is not found\}
```

Solution: Count the number of comparisons.
Worst-Case Complexity of Binary Search

```plaintext
procedure binary search(x: integer, a₁,a₂,…,aₙ: increasing integers)
i := 1 {i is the left endpoint of interval}
j := n {j is right endpoint of interval}
while i < j
    m := [(i + j)/2]
    if x > aₘ then i := m + 1
    else j := m
    if x = aᵢ then location := i
    else location := 0
return location {location is the subscript i of the term aᵢ equal to x, or
0 if x is not found}
```

Solution: Assume n = 2ᵏ elements. Note that k = log n.

Worst-Case Complexity of Bubble Sort

```plaintext
procedure bubblesort(a₁,…,aₙ: real numbers with n ≥ 2)
for i := 1 to n − 1
    for j := 1 to n − i
        if aⱼ > aⱼ₊₁ then interchange aⱼ and aⱼ₊₁
{a₁,…, aₙ is now in increasing order}
```

Solution

Worst-Case Complexity of Insertion Sort

```plaintext
procedure insertion sort(a₁,…,aₙ: real numbers with n ≥ 2)
for j := 2 to n
    i := 1
    while aⱼ > aᵢ
        i := i + 1
        m := aⱼ
    for k := 0 to j − i − 1
        aⱼ₊ₖ := aⱼ₊ₖ₋₁
    aᵢ := m
```

Solution:

Matrix Multiplication Algorithm

- matrix multiplication algorithm; C = A B where C is an
  m × n matrix that is the product of the m × k matrix A and
  the k × n matrix B.

```plaintext
procedure matrix multiplication(A,B: matrices)
for i := 1 to m
    for j := 1 to n
        cⱼ := 0
        A = [aᵢⱼ] is a m × k matrix
        B = [bᵢⱼ] is a k × n matrix
        for q := 1 to k
            cⱼ := cⱼ + aᵢᵢq bᵢⱼ
        return C {C = [cᵢⱼ] is the product of A and B}
```
Complexity of Matrix Multiplication

**Example**: How many additions of integers and multiplications of integers are used by the matrix multiplication algorithm to multiply two $n \times n$ matrices.

**Solution**

Matrix-Chain Multiplication

- Compute matrix-chain $A_1 A_2 \cdots A_n$ with fewest multiplications, where $A_1, A_2, \cdots, A_n$ are $m_1 \times m_2, m_2 \times m_3, \cdots, m_n \times m_{n+1}$ integer matrices. Matrix multiplication is associative.

**Example**: In which order should the integer matrices $A_1 A_2 A_3$ - where $A_1$ is $30 \times 20$, $A_2$ $20 \times 40$, $A_3$ $40 \times 10$ - be multiplied? **Solution**: two possible ways for $A_1 A_2 A_3$.

**Matrix-Chain Multiplication**

- Compute matrix-chain $A_1 A_2 \cdots A_n$ with fewest multiplications, where $A_1, A_2, \cdots, A_n$ are $m_1 \times m_2, m_2 \times m_3, \cdots, m_n \times m_{n+1}$ integer matrices. Matrix multiplication is associative.

**Example**: In which order should the integer matrices $A_1 A_2 A_3$ - where $A_1$ is $30 \times 20$, $A_2$ $20 \times 40$, $A_3$ $40 \times 10$ - be multiplied? **Solution**: two possible ways for $A_1 A_2 A_3$.

Understanding the Complexity of Algorithms

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Theta(1)$</td>
<td>Constant complexity</td>
</tr>
<tr>
<td>$\Theta(\log n)$</td>
<td>Logarithmic complexity</td>
</tr>
<tr>
<td>$\Theta(n)$</td>
<td>Linear complexity</td>
</tr>
<tr>
<td>$\Theta(n \log n)$</td>
<td>Linearithmic complexity</td>
</tr>
<tr>
<td>$\Theta(n^k)$</td>
<td>Polynomial complexity</td>
</tr>
<tr>
<td>$\Theta(b^n)$, where $b &gt; 1$</td>
<td>Exponential complexity</td>
</tr>
<tr>
<td>$\Theta(n!)$</td>
<td>Factorial complexity</td>
</tr>
</tbody>
</table>
Understanding the Complexity of Algorithms

Times of more than $10^{100}$ years are indicated with an *

### Complexity of Problems

- **Tractable Problem**: There exists a polynomial time algorithm to solve this problem. These problems are said to belong to the Class $P$.
- **Intractable Problem**: There does not exist a polynomial time algorithm to solve this problem.
- **Unsolvable Problem**: No algorithm exists to solve this problem, e.g., halting problem.
- **Class NP**: Solution can be checked in polynomial time. But no polynomial time algorithm has been found for finding a solution to problems in this class.
- **NP Complete Class**: If you find a polynomial time algorithm for one member of the class, it can be used to solve all the problems in the class.

### P Versus NP Problem

The $P$ versus $NP$ problem asks whether the class $P = NP$? Are there problems whose solutions can be checked in polynomial time, but cannot be solved in polynomial time?

- Note that just because no one has found a polynomial time algorithm is different from showing that the problem cannot be solved by a polynomial time algorithm.
- If a polynomial time algorithm for any of the problems in the NP complete class were found, then that algorithm could be used to obtain a polynomial time algorithm for every problem in the NP complete class.
- Satisfiability (in Section 1.3) is an NP complete problem.
- It is generally believed that $P \neq NP$ since no one has been able to find a polynomial time algorithm for any of the problems in the NP complete class.
- The problem of $P$ versus $NP$ remains one of the most famous unsolved problems in mathematics (including theoretical computer science). The Clay Mathematics Institute has offered a prize of $1,000,000 for a solution.