Introduction to Computers and Programming

Prof. I. K. Lundqvist

Lecture 9
April 7 2004

So far ...

- Data structures
- Algorithms
Complexity Analysis

“Just how good is my algorithm?”

- Best case vs. worst case
- Storage vs. Computation time
- Computing the computation time
- Big-O notation

In-class Exercise

- Write a procedure that reads an integer \( N \) and calculates the sum of all integers 1..\( N \)
Code Comparison

• How many have a solution that runs in linear time?

```ada
with Ada.Integer_Text_Io, Ada.Text_IO;
use Ada.Integer_Text_Io, Ada.Text_IO;

procedure CalcSum is
    N : Integer;
    Total_Sum : Integer;
begin
    Put_Line("Enter an Integer: ");
    Get(N);
    Total_Sum := 0;
    for I in 1..N loop
        Total_Sum := Total_Sum + I;
    end loop;
    Put(Total_Sum);
end;
```

• How many have a solution that runs in constant time?

```ada
with Ada.Integer_Text_Io, Ada.Text_IO;
use Ada.Integer_Text_Io, Ada.Text_IO;

procedure CalcSum is
    N : Integer;
    Total_Sum : Integer;
begin
    Put_Line("Enter an Integer: ");
    Get(N);
    Total_Sum := 0;
    Total_Sum := (N * (N + 1)) / 2;
    Put(Total_Sum);
end;
```
Complexity Analysis

- **Complexity**: rate at which storage or time grows as a function of the problem size
  - Growth depends on compiler, machine, ...

- **Asymptotic analysis**: describes the inherent complexity of a program, independent of machine and compiler
  - **Idea**: as problem size grows, the complexity can be described as a simple proportionality to some known function.

Common Notations for Big-O

- \( O(1) \)  
  - Constant time or space
- \( O(N) \)
- \( O(N^M) \)
- \( O(M^N) \)
- \( O(\log N) \)

Or a combination of these
O(1)

- Constant time or space, independently of what input we give to the algorithm

- Examples:
  - Access element in an array
  - Retrieve the first element in a list
  - ...

O(N)

- We have to search through all existing elements to find that the element we are looking for does not exist

- Examples:
  - Searching for element in a list that does not exist
  - Searching through a Binary Tree of size N where a value does not exist
O(log N)

- Example, a full balanced Binary Search Tree
- Can eliminate half of the BST every time the search
- Any algorithm that eliminates a large portion of the data set at each iteration is generalized into O(log N)

Binary Search

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>14</td>
<td>17</td>
<td>21</td>
<td>33</td>
<td>55</td>
<td>57</td>
<td>62</td>
<td>71</td>
<td>87</td>
<td>89</td>
<td>91</td>
<td>93</td>
<td>95</td>
<td>97</td>
</tr>
</tbody>
</table>

How many elements are examined in worst case?
Binary Search

Input:
Array to search, element to search for

Output:
Index if element found, -1 otherwise

Algorithm:
Set Return_Index to -1;
Set Current_Index to (UB + LB) /2

Loop
if the LB > UB
Exit;

if Input_Array(Current_Index) = element
Return_Index := Current_Index
Exit;

if Input_Array(Current_Index) < element
LB := Current_Index +1
else
UB := Current_Index - 1

Return Return_Index

O(N^M)

N := 1;
while N > 0 loop
Put("How many repetitions? ");
Get(N);
X := 0;

for I1 in 1..N loop
  for I2 in 1..N loop
    for I3 in 1..N loop
      for I4 in 1..N loop
        for I5 in 1..N loop
          X := X + 1;
          end loop;
        end loop;
      end loop;
    end loop;
  end loop;
end loop;
Put(X);
New_Line;
$O(M^N)$

- Example: Fibonacci algorithm
  - $f(0) = 1$
  - $f(1) = 1$
  - $f(n+2) = f(n) + f(n+1) \quad \forall \, n \geq 0$

$2^N$ calculations

Big-O

- $O(N+M)$
  - Sequential and unrelated tasks
  - Ex: to find the smallest $N_1$ and largest $N_2$
    number in a list and generate a new list of all the numbers in between $N_1$ and $N_2$

- $O(N \times M)$
  - Nesting of tasks
  - Ex: initializing a n-by-m matrix
Asymptotic Analysis: Big-O

- Mathematical concept that expresses “how good” or “how bad” an algorithm is

**Definition:** \( T(n) = O(f(n)) \) – “\( T \) of \( n \) is in Big-Oh of \( f \) of \( n \)”

iff there are constants \( c \) and \( n_0 \) such that:
\[
T(n) \leq cf(n) \text{ for all } n \geq n_0
\]

**Usage:** The algorithm is in \( O(n^2) \) in [best, average, worst] case.

**Meaning:** For all data sets big enough (i.e., \( n > n_0 \)), the algorithm always executes in less than \( cf(n) \) steps in [best, average, worst] case.

Big-O is said to describe an “upper bound” on the complexity.

Big-O Examples

Finding value \( X \) in an array (average cost).

\[
T(n) = c_s n/2.
\]

\( T(n) = O(f(n)) \) iff \( T(n) \leq cf(n) \) for all \( n \geq n_0 \)

For all values of \( n > 1 \), \( c_s n/2 \leq c_s n \).

**Therefore,** by the definition, \( T(n) \) is in \( O(n) \) for \( n_0 = 1 \) and \( c = c_s \).
Big-O Example

\( T(n) = c_1 n^2 + c_2 n \) in average case.

\[ T(n) = O(f(n)) \text{ iff } T(n) \leq cf(n) \text{ for all } n \geq n_0 \]

\( c_1 n^2 + c_2 n \leq c_1 n^2 + c_2 n^2 \leq (c_1 + c_2)n^2 \) for all \( n > 1 \).

\( T(n) \leq cn^2 \) for \( c = c_1 + c_2 \) and \( n_0 = 1 \).

Therefore, \( T(n) \) is in \( O(n^2) \) by the definition

Big-O Simplifications

<table>
<thead>
<tr>
<th>( O(2^N) )</th>
<th>Same as</th>
<th>( O(N) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( O(5 \times 3^N) )</td>
<td>Same as</td>
<td>( O(3^N) )</td>
</tr>
<tr>
<td>( O(4711) )</td>
<td>Same as</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( O(N+1) )</td>
<td>Reduces to</td>
<td>( O(N) )</td>
</tr>
<tr>
<td>( O(N^2 + \log N) )</td>
<td>Reduces to</td>
<td>( O(N^2) )</td>
</tr>
<tr>
<td>( O(N \times \log N + 2^N + 50000) )</td>
<td>Reduces to</td>
<td>( O(2^N) )</td>
</tr>
</tbody>
</table>
**Big-O Simplifications**

<table>
<thead>
<tr>
<th>Function</th>
<th>Simplification</th>
<th>Reduced to</th>
</tr>
</thead>
<tbody>
<tr>
<td>(O(N+P+Q))</td>
<td>Same as</td>
<td>(O(N+P+Q))</td>
</tr>
<tr>
<td>(O(5N^3 + 7N + 2P + Q*R))</td>
<td>Reduces to</td>
<td>(O(5N^3 + 2P + Q*R))</td>
</tr>
<tr>
<td>(O(N^2 \log P + N))</td>
<td>Same as</td>
<td>(O(N^2 \log P + N))</td>
</tr>
<tr>
<td>(O(N*M+N^2))</td>
<td>Same as</td>
<td>(O(N*M+N^2))</td>
</tr>
</tbody>
</table>

---

**Faster Computer or Algorithm?**

The old computer processes 10,000 instructions per hour.

What happens when we buy a computer 10 times faster?

<table>
<thead>
<tr>
<th>Function (T(n))</th>
<th>(n)</th>
<th>(n')</th>
<th>(n'/n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10n)</td>
<td>1,000</td>
<td>10,000</td>
<td>10</td>
</tr>
<tr>
<td>(20n)</td>
<td>500</td>
<td>5,000</td>
<td>10</td>
</tr>
<tr>
<td>(5n \log n)</td>
<td>250</td>
<td>1,842</td>
<td>7.37</td>
</tr>
<tr>
<td>(2n^2)</td>
<td>70</td>
<td>223</td>
<td>3.16</td>
</tr>
<tr>
<td>(2^n)</td>
<td>13</td>
<td>16</td>
<td>------</td>
</tr>
</tbody>
</table>