1 Computational Problems

We can specify a computational problem by specifying what inputs are acceptable and what outputs are acceptable. This is called your input/output specification, or I/O spec for short.

An optimization problem is a computational problem in which the output is chosen among many candidates by selecting the candidate with the smallest or biggest value (this can vary depending on how that value is defined).

An algorithm is a precisely specified computational method that for any legal input is guaranteed to produce a legal output. Algorithms can vary in computational efficiency, or how many operations are required to reach the desired output for any given input. For example, exponentiation using multiplication (evaluating $x^k$) can be carried out in two different ways: we can either multiply $x$ $k$ times (a naive algorithm), or we can use repeated squaring (a clever algorithm). Taking $n$ to be the number of digits in the exponent, the naive algorithm uses $10^n$ operations while the clever algorithm uses only $\frac{n^3}{6}$ operations. We see that, although this does not seem to make a substantial difference in runtime when using small numbers, as $n$ grows our clever algorithm drastically outperforms our naive algorithm.

2 Algorithm Analysis and Big O

Extending our analysis above, we can use formulas to describe the relationship between input size and number of operations required to carry out a procedure. This is called Big O.

2.1 Why is analysis useful?

There are many reasons why algorithm analysis is necessary in the field of computer science. These include:

- help decide whether a computational task is feasible
• help decide which algorithm is best for a given task
• help decide when you have an algorithm that suffices
• help you focus on what part of a big complicated program will be the computational bottleneck

2.2 Simple Model: Counting Operations

In order to analyze a program, we create a model of the computer that captures some aspects of how the computer works. This model is merely an approximation of reality; as we will see, it is not important that the model is exact, only that it captures the correct trends in our programs.

Here are the steps of our simple model:

1. Define basic operations (cons, car, cdr, binding a formal argument to an actual argument, etc.)
2. Measure computational time by the number of basic operations performed

2.3 Analysis Examples

Example 1

Let’s count the number of operations involved in evaluating the expression \((+ab)(+cd))\).

<table>
<thead>
<tr>
<th>Evaluation Step</th>
<th>Number of Basic Operations Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluate symbol *</td>
<td>1</td>
</tr>
<tr>
<td>Evaluate symbol +</td>
<td>1</td>
</tr>
<tr>
<td>Evaluate symbol a</td>
<td>1</td>
</tr>
<tr>
<td>Evaluate symbol b</td>
<td>1</td>
</tr>
<tr>
<td>Bind formal args to actual args</td>
<td>2</td>
</tr>
<tr>
<td>Apply addition procedure</td>
<td>1</td>
</tr>
<tr>
<td>Evaluate (+ c d)</td>
<td>6</td>
</tr>
<tr>
<td>Bind arguments</td>
<td>2</td>
</tr>
<tr>
<td>Apply multiplication procedure</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>16</strong></td>
</tr>
</tbody>
</table>

Quiz 1: Write len procedure

Our task is to write the len procedure, which takes in a list, L, and returns the number of items in L. Here is the len procedure:

```scheme
(define (len L)
  cond
    [(empty? L) 0]
    [(cons? L) (+ 1 (len (cdr L)))]
)```
Quiz 2: Write English2cow procedure

We want to write a procedure, English2cow, which takes in a list L and returns a list of the same length as L all of whose elements are the symbol moo.

\[
\text{(define (English2cow L)} \text{)}
\text{(cond)
\text{[ (empty? L) ’() ]}
\text{[ (cond? L) (cons (quote moo) (English2cow (cdr L)))]})
\]

Now let’s analyze English2cow!

We begin by looking at the base case where our input list L is empty. Counting operations as we did above, we find that the base case takes 7 operations. We then consider an input list of length 1. Our procedure will take 23 operations. Continuing this pattern, we see that 39 operations are required for a list with 2 elements, 55 operations are required for a list with 3 elements, and so on. Thus, we can construct an equation for the number of operations required by our English2cow procedure on a list with an arbitrary number of elements \( n \):

\[
H(n) = 7 + 16n
\]  

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/csci0170/feedback](http://cs.brown.edu/courses/csci0170/feedback)