Example

Algorithms

Objectives

- Given an algorithm, argue its correctness, or provide a counter example.
- Given an algorithm, bound its runtime complexity.
- Given two algorithms, compare their efficiencies.
Addition (Binary addition of 2 numbers) n-bit numbers

Algorithm: align the least significant digits, add one column, carry overflow to next column. Repeat.

Example: carry: 1 1 1 1

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>(53)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(88)</td>
</tr>
</tbody>
</table>
**Multiplication** (Binary multiplication of 2 numbers) *n-bit numbers*  

Algorithm: From grade school, we multiply first number by digit of second number, shifting by the digit's position. Finally, add intermediate values.

**Example:**

\[
\begin{array}{c}
1011 \\
\times 1101 \\
\hline
1011 \\
0000 \\
1011 \\
\hline
10001111
\end{array}
\]

\(11\) \(13\) \((11)\) \((0)\) \((44)\) \((88)\) \((143)\)

**Runtime:**

1 copy and shift \(\Rightarrow O(2n) \Rightarrow O(n)\)

1 copy and shift per bit \(\Rightarrow O(n^2)\)

1 addition \(\Rightarrow O(2n) \Rightarrow O(n)\)

1 addition per bit \(\Rightarrow O(n^2)\)

\(n \cdot \) copy + shifts + \(n-1\) additions \(\Rightarrow O(n^2) + O(n^2) \Rightarrow O(n^2)\)
Multiplication (Recursive version)

\[
x \cdot y = \begin{cases} 
2(x \cdot \lfloor y/2 \rfloor) & \text{if } y \text{ is even} \\
x + 2(x \cdot \lfloor y/2 \rfloor) & \text{if } y \text{ is odd}
\end{cases}
\]

Function \textit{multiply}(x, y):

\begin{align*}
\text{if } y &== 0: \quad \text{return } 0 \\
 z & = \text{multiply}(x, \lfloor y/2 \rfloor)
\end{align*}

\begin{align*}
\text{if } y \text{ is even:} \\
 \quad \text{return } 2 \cdot z
\end{align*}

\begin{align*}
\text{else:} \\
 \quad \text{return } x + 2 \cdot z
\end{align*}

Runtime: \( x, y \) are \( n \)-bit numbers

How many recursive calls?

How much work per call?
División

\(x, y\) are \(n\)-bit numbers

```python
function divide(x, y):
    if x == 0: return (0, 0)
    (q, r) = divide(\lfloor x/2 \rfloor, y)
    q = 2 \cdot q
    r = 2 \cdot r
    if \(x\) is odd:
        r = r+1
    if \(r \geq y\):
        r = r-y
        q = q+1
    return (q, r)
```
Factorial

\[ x \text{ is a } n \text{-bit number} \]

\[ x! = \begin{cases} x \cdot (x-1)! & : \text{if } x > 1 \\ 1 & : \text{if } x = 1 \end{cases} \]

function factorial(x):
    if x == 1:
        return 1
    return x \cdot \text{factorial}(x-1)

Prove the correctness of this function, or give counter example:

What is the runtime of this algorithm as a function of \( x \)? As a function of \( n \)?