Tutorial 9

The goal of this tutorial is to reinforce the following material:

- Efficiency
- Strings
Efficiency

• When looking at a function, it is often useful to understand its running time.

• To do this, we compute the running time as a function of the input size, and use Big O notation to simplify the computation.

In this course, you will see the following running times:

\[ O(1) \quad O(\log n) \quad O(n) \quad O(n \log n) \quad O(n^2) \quad O(n^3) \quad O(2^n) \]
Recursive Functions

Recall the steps for a recursive function:

1. Identify the order of the function *excluding* any recursion

2. Determine the size of the input for the next recursive call(s)

3. Write the full *recurrence relation* (combine step 1 & 2)

4. Look up the closed-form solution in a table
## Recurrence Relations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = O(1) + T(n - k_1) )</td>
<td>( = O(n) )</td>
</tr>
<tr>
<td>( T(n) = O(n) + T(n - k_1) )</td>
<td>( = O(n^2) )</td>
</tr>
<tr>
<td>( T(n) = O(n^2) + T(n - k_1) )</td>
<td>( = O(n^3) )</td>
</tr>
<tr>
<td>( T(n) = O(1) + T\left(\frac{n}{k_2}\right) )</td>
<td>( = O(\log n) )</td>
</tr>
<tr>
<td>( T(n) = O(1) + k_2 \cdot T\left(\frac{n}{k_2}\right) )</td>
<td>( = O(n) )</td>
</tr>
<tr>
<td>( T(n) = O(n) + k_2 \cdot T\left(\frac{n}{k_2}\right) )</td>
<td>( = O(n \log n) )</td>
</tr>
<tr>
<td>( T(n) = O(1) + T(n - k_1) + T(n - k'_1) )</td>
<td>( = O(2^n) )</td>
</tr>
</tbody>
</table>

where \( k_1, k'_1 \geq 1 \) and \( k_2 > 1 \)
Exercise: Snip strings

Implement the following functions and find time complexity:

// snip_left(s) removes (and returns) the first
// character in s
// (returns 0 if s is the empty string)
// effects: modifies s
char snip_left(char *s);

// snip_right(s) removes (and returns) the last
// character in s
// (returns 0 if s is the empty string)
// effects: modifies s
char snip_right(char *s);
Exercise: Word Search

Implement the following functions and find time complexity:

```c
// print_word(ws, row, col, direction, len)
// prints the len letters
// that appear in ws at location row, col
// in the given direction
// requires: row < ws->height, col < ws-width
// word does not extend beyond the
// ws boundaries [not asserted]
// direction is one of: UP, LEFT, RIGHT, DOWN
// effects: displays output
void print_word(struct word_search *ws, int row, int col, int direction, int len);
```