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:

\[ O(1) \quad O(\log n) \quad O(n) \quad O(n \log n) \quad O(n^2) \quad O(n^3) \quad O(2^n) \]
Recurrence Relations

\[
T(n) = O(1) + T(n - k_1) = O(n)
\]
\[
T(n) = O(n) + T(n - k_1) = O(n^2)
\]
\[
T(n) = O(n^2) + T(n - k_1) = O(n^3)
\]
\[
T(n) = O(1) + T(n - \frac{n}{k_2}) = O(\log n)
\]
\[
T(n) = O(1) + k_2 \cdot T(\frac{n}{k_2}) = O(n)
\]
\[
T(n) = O(n) + k_2 \cdot T(\frac{n}{k_2}) = O(n \log n)
\]
\[
T(n) = O(1) + T(n - k_1) + T(n - k_1') = O(2^n)
\]

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:

// 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);