CS1101S Finals Reference AY23/24
by Zeepheru (AY23/24)

Orders of Growth

(Limits used for succinctness, not correct.)

Big Theta, **Big Omega**, and **Big O**:

$$\theta(g(n)) \iff \exists\, k_1, k_2 \in \mathbb{Z}^+\ \exists\, n_0 \in \mathbb{R}$$
$$(\forall\, n > n_0(k_1 \cdot g(n) \le r(n) \le k_2 \cdot g(n)))$$
$$O(g(n)) \iff \exists\, k \in \mathbb{Z}^+(\lim_{n\to\infty}(k \cdot g(n) \ge r(n)))$$
$$\Omega(g(n)) \iff \exists\, k \in \mathbb{Z}^+(\lim_{n\to\infty}(k \cdot g(n) \le r(n)))$$

Order (smol to big): 1, $\log n$, n, $n \log n$, n^2, n^3, 2^n, 3^n, n^n

Note: $r(n)$ has OOGs $\theta(r(n))$, $O(r(n))$, and $\Omega(r(n))$.

Common Recurrence Relations

$$\begin{aligned} T(n) &= O(1) + T(n-1) \implies O(n) \\ &= O(\log n) + T(n-1) \implies O(n \log n) \\ &= O(n) + T(n-1) \implies O(n^2) \\ &= O(1) + 2T(n-1) \implies O(2^n) \\ &= O(1) + T(\frac{n}{2}) \implies O(\log n) \\ &= O(n) + 2T(\frac{n}{2}) \implies O(n \log n) \\ &= O(n) + T(\frac{n}{2}) \implies O(n) \\ &= O(1) + 2T(\frac{n}{2}) \implies O(n) \end{aligned}$$

Generally, $T(n) = O(n^k) + T(n-1) \implies O(n^{k+1})$

Lists

⭐ Check if x is in list xs

```
if (!is_null(member(x, xs))) { }
```

Remove Duplicates

```
function remove_duplicates(xs) {
    return accumulate(
                (curr, wish) =>
                    pair(curr,
                         filter(x => x !== curr, wish)),
                null, xs);
}
```

Permutations

```
function permutations(ys) {
    // list => list of lists
    return is_null(ys)
        ? list(null)
        : accumulate(append, null,
            map(x => map(p => pair(x, p),
                         permutations(remove(x, ys))),
                ys));
}
```

Arrays

⭐ Reverse Index

```
A[len - i - 1];
```

List-array conversion

```
function list_to_array(xs) {
    const A = [];
    let i = 0;
    while (!is_null(xs)) {
        A[i] = head(xs);
        xs = tail(xs);
        i = i + 1;
    }

    return A;
}

function array_to_list(A) {
    function helper(i) {
        if (A[i] === undefined) {
            return null;
        } else {
            return pair(A[i], helper(i + 1));
        }
    }

    return helper(0);
}
```

Reminders :)

Return a block:

```
return {...; return x; };
```

`stream_tail` returns `null` if nothing left, `stream_ref` returns `undefined`.

On `equal()`:

```
const a = pair(null, null);
const b1 = pair(a, a);
const b2 = pair(a, a);

equal(b1, b2); # returns TRUE
```

QRFs

- `integers_from(start)` - Stream.
- `for_each(f, xs)` - Maps functions in place.
- `build_stream/list(f, n)` - 0 to $n-1$.
- `enum_stream/list(start, end)` - Includes `start` and `end`.
- `eval_stream(s, n)` - Generates list of first n stream values.