## Syntax

| | |
|---|---|
| Types | val npower = fn : real * int -> real |
| Strings | explode(), implode() convert to lists<br>#”c” is a character constant<br>^ appends |
| Ints | div does integer division |
| Types | datatype vehicle = Boat \| Plane;<br>datatype vehicle = Bike of int \| Car; |
| Case | case E of Bike x => F \| Plane => G; |
| Errors | exception Failure;<br>raise Failure;<br>E handle Failure => F \| Match => G; |
| Lambda | (fn x => E)<br>val prefix = (fn a => (fn b => a^b))<br>fun prefix a b = a ^ b;<br>opX (e.g. op<=) is operator as fun |
| Refs | Have type ‘a ref, created as ref E<br>!P returns contents, P := E updates<br>Allows iteration: while B do C |
| Arrays | Has a type of ‘a Array.array<br>Array.tabulate(n, f), Array.sub(A, i),<br>Array.update(A, i, e) |

## Algorithms

| | |
|---|---|
| Sorting | Bubble<br>Insertion<br>Quick<br>Merge |
| Queue | Implemented with two arrays and reverse: amortized constant cost |
| Trees | Binary search<br>Breadth-first: using a queue or iterative deepening (number of nodes at level n + 1 is greater than the number of nodes on all previous levels combined)<br>Priority queue: using a heuristic function during searching |
| Functional Arrays | Express index as binary number: leading 1 is discarded as it is present in each index, the remaining bits code from LSB to MSB for binary tree path taken |
| Lists | map applies a lambda expression<br>foldl, foldr apply expression recursively (e.g. foldl op+ (0, xs)) left or right along a list<br>exists, filter do the obvious |
| Lazy Lists | datatype ‘a seq = Nil \| Cons of ‘a * (unit -> ‘a seq);<br>Make sure forces (xf) are enclosed in delays (fn () => E) |
| Mutable Lists | datatype ‘a mlist = Nil \| Cons of ‘a * ‘a mlist ref; |