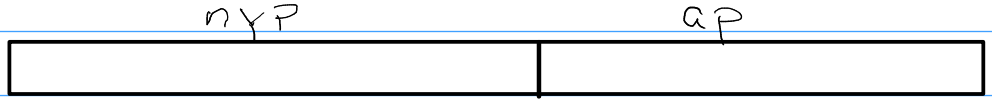
Selection Sort for $\leq$ [Iterative]

First thoughts and functional decomposition



A possible plan:

max-digit ?
- find a largest digit d in nyp
- "slide" d to the right and make it part of ap

remove-digit
- remove one occurrence of d from nyp
- iterate with the shorter nyp
- continue until nyp = 0, at which point return ap.

(1) and (2) will be in the GT

(1) Observe that every digit in nyp is $\leq$ any digit in ap and (2) ap is sorted

Observe: once we give the relation between nyp, ap and n, as well the specs for the helper functions, we have a design idea

(3) will be in the GT

sorted(N) = [sorted(nyp)][ap]

Notation: for integers $p, q > 0$ let's agree that [p][q] is shorthand for $p * 10^{\#digits(q)} + q$

eg: [12][345] = 12345

spec for max-digit

```
; pre: n ≥ 0 is an integer
(max-digit n)
; post: returns largest digit in n
```

spec for remove-digit

```
; pre:  n ≥ 0 is an integer
;       and d is a digit
(remove-digit n d)
; post: if n = [p][d][q], returns [p][q]
;       and if d does not occur in n,
;       returns n
```

Tentatively

gt is ① && ② && ③

Note that ③ has the form

Total Work = Work Remaining + Work Already Completed

Possible Problem — our idea is to stop when x*y*p = 0 and we want ③ to hold — but:

What is [0][7]?

Let's agree that [0][7] is just 7.

Makes most sense to code the main function at this point — in part because max-digit and remove-digit will themselves be loops, perhaps complicated; need their own developments.

```
(define (sel-sort n)
  (cond ((< n 10) n)
        (else (sel-iter n 0))))
```

looks like we need to make sense of [p][0] as well — analogously, [p][0] = [p]

```
(define (sel-iter nyp ap)
  (cond ((zero? nyp) ap)
        (else (let ((m (max-digit nyp)))
                (sel-iter
                  (remove-digit nyp m)
                  (make-number m ap))))))
```

Yikes! Another helper

spec for make-number

```
; pre: p ≥ 0 and q ≥ 0 are integers
  (make-number p q)
; post: returns [p][q]
```

With this wrapper, do we have

sorted(N) = [sorted(nyp)][ap]

when sel-iter is called for the first time?

Since: nyp = N and

[sorted(nyp)][0] =

sorted(nyp)

You can see that our IH is weak enough.

What about ① and ②?

① Observe that every digit in nyp is ≤ any digit in ap and ② ap is sorted

The idea is that ① should be vacuously true because we haven't yet put any digits in ap. BUT

⇝? So [0] is not 0?

Houston: we have a problem!

one work-around might be to redefine the bracket notation. But [1][0] is naturally 10, and I don't want to change this.

Maybe a better option would be → to change the wrapper. Instead of

```
(define (sel-sort n)
  (cond ((< n 10) n)
        (else (sel-iter n 0))))
```

perhaps

```
(define (sel-sort n)
  (cond ((< n 10) n)
        (else (let ((d (max-digit n)))
                (sel-iter
                  (remove-digit n d)
                  d))))))
```

Again we check ①, ②, and ③

sorted-N = [sorted (n d)] [ap]

TRUE

① is true because we moved a largest digit of n to ap

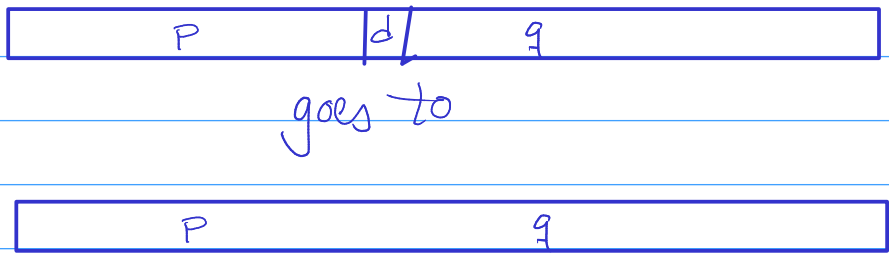
② is True because ap has only one digit

I'll leave you to check the strong enough and preservation conditions.

---

what about remove-digit ?

informal spec



Working notes GI

d does not occur in ap

Main GI equation would seem to be

N(without rightmost occurrence of d) =

[nyp(without rightmost occurrence of d)][ap]

Termination idea: scan from the right until d is found or nyp = 0 and then return [nyp][ap] (if d is found)

or

[0][ap] = [ap] = N

(if d is not found)

[This design satisfies a stronger post than we need — as it guarantees that the rightmost occurrence of d is the one removed]

You'll want that make-number function here as well:

(make-number p q) = [p][q] =

$p * 10^{\#digits(q)} + q$

~~~

iterative max-digit is easier than what we've done so far tonight, and as it grows late ...

~~~

Thanks for working with me!