APL Assignment

Author: xtp778

Exercise 1.1

In [38]:

```
%%bash
apl -s <<EOF
sumsq ← {(α*2)+(ω*2)}
1 2 3 4 5 sumsq 3 2 1 4 1
)OFF
EOF
```

```
10 8 10 32 26
```

Exercise 1.2

In [119]:

```
%%bash
apl -s <<EOF
divs ← {+/0⍷α|ιω}
divs37 ← {(3 divs ω)+(7 divs ω)}
divs37 3
divs37 6
divs37 7
divs37 12
)OFF
EOF
```

```
1
2
3
5
```

Exercise 1.3

In [10]:

```
%%bash
apl -s <<EOF
sumsecond ← {+/ω[(2×ι(ρω)÷2) - 1]}
sumsecond 1 2 3 4 5 7 92 2
)OFF
EOF
```

```
101
```

Exercise 2.1

This code could not execute in GNU APL (`bad char in execute+`), so the following was run in tryapl.org ():

```
xscanl ← { 0=⊃ρω: θ ⋄ ωω , (αα ∇∇ (ωω αα ⊃ω)) 1↓ω }
    (- xscanl 0) 1 2 3 4
0 ¯1 ¯3 ¯6
    o ← {0⌊α+ω}
    minsum ← {⌊/(o xscanl 0) ω}
```

An example to prove that the `minsum` works using the `xscanl` function requires a segment of relatively smaller numbers in a longer segment. For instance `[-2 1 -3]` in the sequence ¯2 1 ¯3 4 ¯1 2 1 ¯2 4 (borrowed from Wikipedia (https://en.wikipedia.org/wiki/Maximum_subarray_problem)).

```
minsum ¯2 1 ¯3 4 ¯1 2 1 ¯2 4
-4
```

Which is correct, since `-2 + 1 -3 = -4`. Introducing a segment that is smaller gives another result:

```
minsum ¯2 1 ¯3 4 ¯1 2 1 ¯2 4 ¯5 1 ¯3
-5
```

Which is actually unexpected, since `-5 > (-5 + 1 - 3)`. Appending a number (0) appears to fix the problem:

```
minsum ¯2 1 ¯3 4 ¯1 2 1 ¯2 4 ¯5 1 ¯3 0
-7
```

Which is correct, since `-5 + 1 -3 = -7`.

In conclusion the algorithm seems to work, although it has some challenges with locating minimum segment sum in the end of a list.

Exercise 2.2

The code for o: `o ← {0⌊α+ω}` is not possible to parallelise because it is not associative in the setting of `xscanl`. Associativity is important in this case because the order that the `xscanl` function recursively finds the smallest element matters. joining of α and ω matters. The function `⌊` will take the minimum of 0 and the sum of each element in α and ω. If the sum of two elements in α and ω is greater than 0, 0 will be returned. However, if the array is parsed in different orders with different splits the results will not be the same, resulting in different (unpredictable) behaviour when executed in parallel.

Exercise 3

In [42]:
```
%%bash
apl -s <<EOF
maxidx ← {ω[1↑⍒ω] 1↑⍒ω}
maxidx 1 2 3 4 5 7 92 2
)OFF
EOF
```

```
 92  7
```

Exercise 4

In [135]:
```
%%bash
apl -s <<EOF
longestdecr ← { 1 + ⌈/(⍴¨(⍵=(¯1⌽⍵-1))⊂⍵) }
longestdecr 5 4 7 1 9 8 7 2 ¯1 2 ⍝ 3
longestdecr 5 4 7 1 9 8 7 6 5 4 3 2 ¯1 2 ⍝ 8
longestdecr 5 4 7 1 8 ¯6 4 2 ¯1 2 ⍝ 2
longestdecr 5 3 1 ⍝ 1
)OFF
EOF
```

```
 3
 8
 2
¯∞
```

Note: This solution assumes that a list of length 2 or more is passed and that the longest sequence of decending numbers is at least 2.