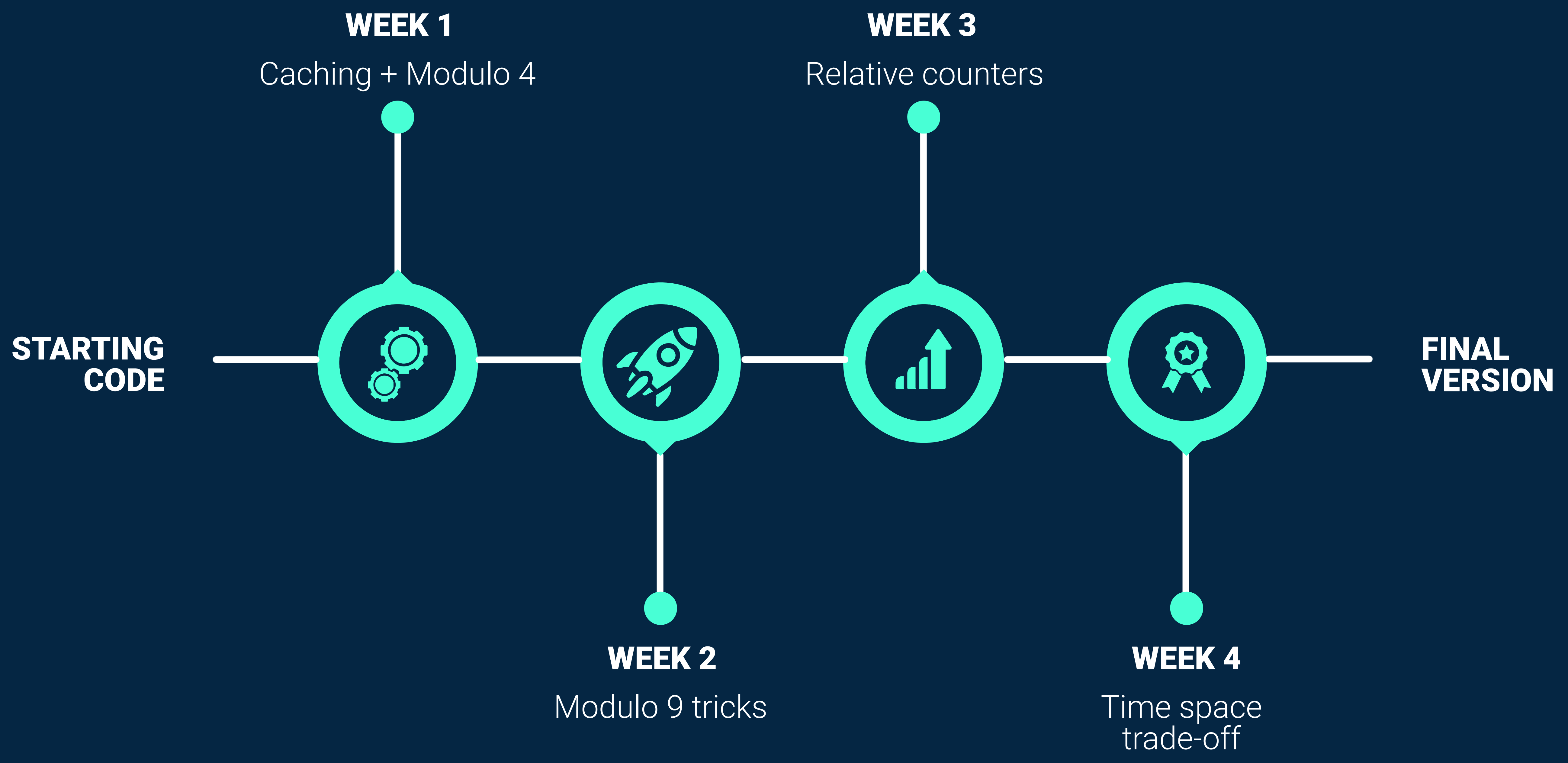
Collatz Conjecture in C

ΔΗΜΗΤΡΙΟΣ ΑΛΕΞΑΝΔΡΗΣ

CODING PROGRESSION



```c
int counter = 0;
while (n > 1) {
    if (n % 2 == 0)
        n = n/2;
    } else {
        n = 3*n+1;
    }
    counter++;
}
return counter;
```

STARTING CODE

The basic implementation of the assignment regarding the Collatz conjecture in C.

```
unter = 0;
ile (n > 1) {
    if (cache[n] != 0) {
        counter += cache[n];
        break;
    }
    if (n % 2 == 0) {
        n = n/2;
    } else {
        n = 3*n+1;
    }
    counter++;

rn counter;
```

CACHING

Implementing caching (memoization) on the code to speed up performance and lower the number of calculations

```
uint64_t q;

if (n % 4 == 1) {
    q = buffer[(3*n+1)/2] // 3 steps
    if (q != 0) { counter = q + 3} // + 3 ^^
    }
else if (n % 4 == 3)
    {/*basic algorithm*/}

else // n%4 == 2 , n%4 == 0
    {
        q = buffer[(n/2)];
        if (q!=0) { counter = q + 1 }
    }
```

MODULO 4

Implementing mod 4 tricks to speed up peformance
and skip calculations from heavy loops.

```
register Int j = 0;
for(register Int i = 3; i <= end; i += 36) {
    /* 3 */ entries[j++] = iterate_from(i+ 0, scores+(i+ 0));
    /* 7 */ entries[j++] = iterate_from(i+ 4, scores+(i+ 4));
    /* 2 */ entries[j++] = (scores[i+ 8] = -2, i/3*2+5); //mod(9) == 2
    /* 6 */ entries[j++] = iterate_from(i+12, scores+(i+12));
    /* 1 */ entries[j++] = iterate_from(i+16, scores+(i+16));
    /* 5 */ entries[j++] = (scores[i+20] = -2, i/3*2+13); //mod(9) == 5
    /* 0 */ entries[j++] = iterate_from(i+24, scores+(i+24));
    /* 4 */ entries[j++] = (scores[i+28] = -5, ((i+24)/9)*8+3); //mod(9) == 4
    /* 8 */ entries[j++] = (scores[i+32] = -2, i/3*2+21); //mod(9) == 8
}
```

MODULO 9

Implementing mod 9 tricks to speed up peformance and skip calculations from heavy loops.

ericr.nl

```
register Int j = 0;
for(register Int i = 3; i <= end; i += 36) {
    /* 3 */ entries[j++] = iterate_from(i+ 0, scores+(i+ 0));
    /* 7 */ entries[j++] = iterate_from(i+ 4, scores+(i+ 4));
    /* 2 */ entries[j++] = (scores[i+ 8] = -2, i/3*2+5); //mod(9) == 2
    /* 6 */ entries[j++] = iterate_from(i+12, scores+(i+12));
    /* 1 */ entries[j++] = iterate_from(i+16, scores+(i+16));
    /* 5 */ entries[j++] = (scores[i+20] = -2, i/3*2+13); //mod(9) == 5
    /* 0 */ entries[j++] = iterate_from(i+24, scores+(i+24));
    /* 4 */ entries[j++] = (scores[i+28] = -5, ((i+24)/9)*8+3); //mod(9) == 4
    /* 8 */ entries[j++] = (scores[i+32] = -2, i/3*2+21); //mod(9) == 8
}
```

RELATIVE COUNTERS

Using relative counters to lower memory usage for L1 cache and speed up CPU Computations.

For example:
for n=3
3, 10, 5, 16, 8, 4, 2
We would store in scores[3] the cost relative to 2 (store 6)

```c
static const Int __k = 5;
static const Tiny __c[32] = { 0, 3, 2, 2, 2, 2, 2, 4, 1, 4, 1, 3, 2, 2, 3, 4, 1,
static const Tiny __d[32]  = { 0, 2, 1, 1, 2, 2, 2, 20, 1, 26, 1, 10, 4, 4, 13, 4


#pragma GCC optimize 3
static Int iterate_from(const Int i, Score* restrict count_to_ref) {
    register Int count = 0;
    BigInt cursor = i; // Use extra var on 64 bits, because those ones go crazy
    do {
        Int b = cursor % 32;
        Int number_of_odd = __c[b];
        cursor = pow3(number_of_odd) * (cursor / 32) + __d[b];
        count += (__k - number_of_odd) + 2*number_of_odd;
    } while(cursor >= i);
    *count_to_ref = (Score) count;
    return cursor;
}
```

TIME SPACE TRADE-OFF

Using a method called "time space trade-off" to use precomputation of a small array of numbers calculate bigger numbers faster.

```
#pragma GCC optimize 3
static Int iterate_from(
```

```
register unsigned int j = 0;
```

```
restrict count_to_ref) {
```

EXTRAS ++;

Using #pragma GCC optimize 3, hints the compiler to use specific optimization tricks that will speed up the code. Avoiding the need for the use of assembly.

Register to make the variable faster to access.

Restrict to tell the compiler that the memory address is going to be accessed only by that pointer.

SOURCES ++;

Wikipedia

Time–space tradeoff

The section *As a parity sequence* above gives a way to speed up simulation of the sequence. To jump ahead k steps on each iteration (using the f function from that section), break up the current number into two parts, b (the k least significant bits, interpreted as an integer), and a (the rest of the bits as an integer). The result of jumping ahead k is given by

$$f^k(2^k a + b) = 3^{c(b,\,k)} a + d(b, k).$$

The values of c (or better 3^c) and d can be precalculated for all possible k-bit numbers b, where $d(b, k)$ is the result of applying the f function k times to b, and $c(b, k)$ is the number of odd numbers encountered on the way.[30] For example, if $k = 5$, one can jump ahead 5 steps on each iteration by separating out the 5 least significant bits of a number and using

$c(0...31, 5) = \{0, 3, 2, 2, 2, 2, 2, 4, 1, 4, 1, 3, 2, 2, 3, 4, 1, 2, 3, 3, 1, 1, 3, 3, 2, 3, 2, 4, 3, 3, 4, 5\}$,
$d(0...31, 5) = \{0, 2, 1, 1, 2, 2, 2, 20, 1, 26, 1, 10, 4, 4, 13, 40, 2, 5, 17, 17, 2, 2, 20, 20, 8, 22, 8, 71, 26, 26, 80, 242\}$.

Ericr.nl

Modulo tricks