25/30 nice!

Exercise 4

a)

There exists a program for $n \in \mathbb{N} \setminus \{0\}$ that generates w_n:

Algorithm 1

1: $k \leftarrow n$	▷ Assign n to k
2: $k \leftarrow 3 \cdot (k\hat{}2)$	▷ Compute $3k^2$
3: $k \leftarrow 4\hat{}k$	▷ Compute $4^k = 4^{3n^2}$, where n is the initial n
4: **for** $i := 1$ to k **do**	▷ Print "101" $k = 4^{3n^2}$ times
5: $\quad print(101)$	

sehr gut!

The only string in the code that depends on the length of w_n is the representation of n:
⇒ Everything else is a constant for all possible w_n of the form from above
⇒ Hence we can estimate the Kolmogorov Complexity:

$$K(w_n) \leq \lceil \log{(n+1)} \rceil + c, \text{ c constant}$$

5/5

Now we have to solve n for $|w_n|$. We have $|w_n| = |101| \cdot 4^{3n^2} = 3 \cdot (2^2)^{3n^2}$ $= 3 \cdot 2^{6n^2}$ (✓)

$\Rightarrow \frac{|w_n|}{3} = 2^{6n^2}$
$\Rightarrow \log_2 \frac{|w_n|}{3} = 6n^2$
$\Rightarrow \frac{\log_2 \frac{|w_n|}{3}}{6} = n^2$
$\Rightarrow n = \sqrt{\frac{\log_2 \frac{|w_n|}{3}}{6}}$
Hence we have that:

$$K(w_n) \leq \lceil \log{(n+1)} \rceil + c = \lceil \log{(\sqrt{\frac{\log_2 \frac{|w_n|}{3}}{6}} + 1)} \rceil + c$$

is an upper bound of the Kolmogorov Complexity

could write it a little bit more beautifully as $\log((\log w_n - \log_2 3)/6 + 1)$, but that's fine

very nice structure, keep it that way! Nearly perfect :)

b)

We define $y_n := 2^{3^{n+1}}$
It follows that $y_i < y_j$ for all $i, j \in \mathbb{N}$ with $i < j$, because the exponential function is strict increasing for base greater than 1. Good!
Assuming that print returns the binary representation of the number, we have the following algorithm:

Algorithm 2

1: $y \leftarrow n$	▷ Assign n to k
2: $y \leftarrow y + 1$	▷ Increment k
3: $y \leftarrow 3\hat{}y$	▷ Compute 3^y
4: $y \leftarrow 2\hat{}y$	▷ Compute $2^y = 2^{3^{n+1}}$, where n is the initial n
5: $print(y)$	▷ Print the binary representation of y

The representation of n is the only string in the code that is not constant
⇒ Everything else is a constant for all possible n
⇒ Hence we can estimate the Kolmogorov Complexity:

5/5

$$K(y_n) \leq \lceil \log_2 (n+1) \rceil + c, \text{ c constant}$$

We can reformulate the estimation as follows:

$$\begin{aligned} K(y_n) &\leq \lceil \log_2 (n+1) \rceil + c, \text{ c constant} \\ &= \lceil \log_2 \log_3 3^{n+1} \rceil + c \\ &= \lceil \log_2 \log_3 \log_2 (2^{3^{n+1}}) \rceil + c \\ &\overset{(*)}{=} \lceil \log_2 \log_3 \log_2 y_n \rceil + c \\ &\leq \log_2 \log_3 \log_2 y_n + c', \text{ with } c' \cancel{\leq c+1} \end{aligned}$$

const. don't worry about such details :-)

very good!

Exercise 5

Prove that, for all $n \in \mathbb{N}$ and $i < n$, there are at least $2^n - 2^{n-i}$ natural numbers x in the interval $[2^n, 2^{n+1} - 1]$ such that $K(x) \geq n - i$.
We notice that there are 2^n numbers in said interval.
(There are $b - a + 1$ natural numbers in the interval $[a, b]$ IF $a, b \in \mathbb{N}$)

8/10

There are exactly

$$\sum_{i=1}^{n-i-1} 2^i = 2^{n-i} - 2 \ (✓)$$

$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

(sum over the number of all possible bit strings of length 1 to n-i-1) bit-strings of length strictly less than $n - i$, thus there can be at most $2^{n-i} - 1$ different programs with

$$K(x) < n - i$$

not really, you mean "... programs ps with length(p) < n-i.."

That is because every program is compiled into a bit-string (Machine code) and different programs are compiled into different bit-strings. each program only generates 1 output (✓)
For different numbers the program to generate that number is different. The bit-string in which the program is compiled is therefore also different.
We have just showed that there are at most $2^{n-i} - 1$ different programs with $K(x) < n - i$, it follows that there can be at most $2^{n-i} - 1$ different numbers (in the interval $[2^n, 2^{n+1} - 1]$) with $K(x) < n - i$.
⇒ Since there are 2^n different numbers in $[2^n, 2^{n+1} - 1]$, there are at least

$$2^n - (2^{n-i} - 1) = 2^n - 2^{n-i} + 1 > 2^n - 2^{n-i}$$

numbers in the interval $[2^n, 2^{n+1} - 1]$ with $K(x) \geq n - i$. good!

K-compl. of word w ≤ length of program p generating w
K(w) ≤ len(p)

Exercise 6

We make some observations about the elements of L:

- $|x_n| = i + j + k = 2k + k = 3k$ ✓
- For a given k, there are $2k+1$ numbers of length $3k$ why ?
- For a given k, the number of elements with size smaller than $3k$ is the sum:

→ not strictly smaller...

$$\begin{aligned}\sum_{i=1}^{k-1}(2i+1) &= 2\sum_{i=1}^{k-1} i + \sum_{i=1}^{k-1} 1 \quad (\checkmark)\\ &= 2\cdot\frac{(k-1)k}{2} + k - 1 \quad (\checkmark)\\ &= (k-1)k + k - 1\\ &= k(k-1+1) - 1\\ &= k^2 - 1 \quad (\checkmark) \qquad (k+1)^2 - 1 \ldots\end{aligned}$$

Hence, for x_n with $|x_n| = 3k$, we can conclude that $k^2 \leq n < (k+1)^2$. Now we can calculate k by finding the biggest power of 2 smaller than n { why?

- why? The parameter i determines the canonical ordering for words of equal size $3k$. Hence, for a given n, we can find i by subtracting the number of elements of size smaller than $3k$ from n.

7/10

From the observations from above, we can conclude that for a given n, it is possible to compute the corresponding i, j, k. Hence there exists a program for $n \in \mathbb{N} \setminus \{0\}$ that generates x_n:

Algorithm 3

```
m ← n                                    ▷ Assign n to m
k ← 1                                    ▷ Initialize k
while (k + 1) · (k + 1) ≤ m do           ▷ We calculate the biggest k such that k^2 ≤ n
    k ← k + 1                            ▷ This will be ___ ?
i = m − (k^2 − 1)                        ▷ Compute i from k and m
j = k − i                                ▷ Compute j from i and k
for i := 1 to i do                       ▷ Print "1" i times
    print(1)
for i := 1 to j do                       ▷ Print "0" j times
    print(0)
for i := 1 to k do                       ▷ Print "1" k times
    print(1)
```

✓

why does it work, i.e. proof that it really finds the n-th word in can. order

The only string in the code that depends on the length of x_n is the representation of n:
⇒ Everything else is a constant for all possible x_n
⇒ Hence we can estimate the Kolmogorov Complexity:

$$K(x_n) \leq \lceil \log(n+1) \rceil + c, \text{ c constant}$$

We can reformulate the estimation as follows:

$$
\begin{aligned}
K(x_n) &\leq \lceil \log{(n+1)} \rceil + c, \text{ c constant} && n \leq (k+1)^2 \\
&\leq \lceil \log_2{((k+1)^2+1)} \rceil + c \\
&= \lceil \log_2{(k^2+2k+1+1)} \rceil + c && k \geq 1 \\
&\leq \lceil \log_2{(k^2+2k^2+k^2+k^2)} \rceil + c \\
&= \lceil \log_2{(5k^2)} \rceil + c \\
&\leq \lceil \log_2{(9k^2)} \rceil + c \\
&= \lceil \log_2{(3k)^2} \rceil + c \\
&= \lceil \log_2{|x_n|^2} \rceil + c \\
&= \lceil 2\log_2{|x_n|} \rceil + c \quad \checkmark
\end{aligned}
$$

We can conclude that there exists a constant $c \in \mathbb{N}$ such that, for all $n \in \mathbb{N}$,

$$K(x_n) \leq \lceil 2\log_2{|x_n|} \rceil + c, \text{ c constant} \quad \checkmark$$

Some issues with off-by-one, but really nice solution!