Assignment 1

Author: Lan Stare **Date:** 4. 9. 2025

Question 1

- 2.8×10^6 Explanation: We want $n \log(n) \leq 6.0 \times 10^7$. I will use base 2 for logarithms. I tried to use $n = 4000000$ and got a too big number, so through trial and error I got to $n \approx 2.8 \times 10^6$ and got the number: $5.996 \ldots \times 10^6$ which is a good enough approximation. I followed a similar procedure for the following examples. I have also rounded values so the number of elements processed might be a bit less.
- 1.5×10^5
- 1.3×10^1
- 1.7×10^2
- 2.0×10^0
- 1.1×10^1
- 4.0×10^6

Question 2

It is not always true that for every function f:

$$f(n) \in \mathcal{O}(f(n+1))$$

for example if we take $f : \mathbb{N} \to \mathbb{N}$ by

$$f(n) = \begin{cases} n, & \text{if } n \text{ is even,} \\ 1, & \text{if } n \text{ is odd.} \end{cases}$$

This function is a counterexample, since we can always choose k large enough so that $2k > c$, giving a contradiction. Therefore no such constant c exists, so $f(n) \notin O(f(n+1))$.

Question 3

- **Proof 1**
  If we choose $c := 2$ and $n_0 := 1$, then we get: $2n \leq 2n$ for all $n \geq n_0$

- **Proof 2**
  If we choose $c := 6$ and $n_0 := 5$, then we get: $3n + 5 \leq 6n$ for all $n \geq n_0$ due to the fact that $3 * 5 + 5 = 20$ and $6 * 5 = 30$ and the fact that the linear function $6n$ is increasing faster than $3n$ on the $\mathbb{R}$.

Question 4

- **a)** The worst case scenario is if n is a large number and all elements of the vector $v[]$ have to be odd from the element 5 onwards. This is when the function will keep executing untill i runs through all n. Time complexity: 2., 3., 6., 8., 10. rows happen in constant time since they only happen once. 4. and 5. row happen in $\mathcal{O}(n)$ since they have to compare i with n until $i == n$ and since the function will keep executing due to oddness of elements of vector $v[]$, this comparisson will happen n times. Therefore worst case complexity is $\mathcal{O}(n)$.

- **b)** The best case scenario is if $n \leq 0$ then the the time complexity of row 4 is constant, since it only has to compare two known elements and then the function jumps to return i, which is also constant. Time complexity is therefore constant. It only executes 4 constant operations.

Question 5

In [4]:
```python
def recursion(n):
    if n <= 1:
        return 1
    return n * recursion(n - 1)
```

Time complexity of the function recursion(n) is $\mathcal{O}$(n): in the first row we have a constant operation since we only compare n to a constant 1, the second row is also a constant operation since we only return a constant 1. The final row multiplies n with the result of function recursion(n). Multiplication is a constant operation, but the function recursion(n) is called again with argument n - 1. This means that the function will be called n times until it reaches the base case, which has constant time complexity. Therefore the time complexity is $\mathcal{O}$(n).