Practical exercises 1

1.1 Recursion in C

a.
Running the recursive function with the number n = 196,607, yields the sum 2,147,385,344.
Incrementing n once, yields the sum -2,147,385,344.
What happens is that the 4-byte integer that holds the sum, is not large enough to store the sum of numbers from 1 to 196,608.

In the table below, we can see the arithmetic operation that is taking place:

	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
=	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Adding the two numbers 2,147,385,344 and 196,608 leads to the number 2,147,581,952, which is larger than the max value for a 4-byte two's complement binary number (2,147,483,647) and we get an integer overflow as shown in the table.

Running the function with extremely large values of n, leads to a segmentation fault, as we run out of space on the stack. We have not researched exactly at which n this happens, since it depends on what is on the stack before the function call and what is put on the stack every time the function calls itself.
Hence the value of n would, most likely, change when the program is changed, e.g. if adding new local variables to functions that are called before sum() is called. See line 48 in rec_sum.c.

b.
Choosing to look at the two local variables in the function print_locals() (defined in line 22 and 23). When printing their addresses, we see that they are 1 byte apart.
If one of them were static, they could be further apart, since they would be stored in different segments.

c.
An initialized global int is located in the data segment.

d.
The addresses of local variables in the recursive function decrease with recursion depth because local variables are put on the stack and new variables are pushed to the stack at the end that has lower addresses. The highest address is used for the first variable pushed on the stack, the next gets the address(es) below that one etc.