```mermaid
flowchart TD
    task1["task1"]
    create["will create and write and close file the output"]
    encryption["encryption"]
    decryption["decryption: since encrypted holds binary, need to use keydecryption that says paw in binary for xor"]

    task1 --> create
    task1 --> encryption
    task1 --> decryption
```

task1

will create and write and close file the output

encryption

google tells me to use esi and edi for xor byte by byte

use loop to do so

decryption: since encrypted holds binary, need to use keydecryption that says paw in binary for xor

decryption works as expected and displays properly. intially did not and tried convert to hex to ascii. turns out it was unesescarry as system converts it for me when write to file

i will use w1 activity where we manually xor to encrypt (by hand). since i know what to expect as result, it will be a good idea to use what i have already done and understand

ok i try to write encryption to file. i do not get result i want. first it print to terminal even though I used correct file writing code

ok looks like i have to convert, it worked for a bit but i changed somethig and now it doesnt work.

i should save different files as i make major changes so i dont lose stuff

many attempts to get it to work, it shows red dots, strange symbols, wrong numbers. It is a big struggle to get to hex rep

try to convert to hex, best i can do now is **1407, the * being the red dots.**

now it works but prints to terminal depsite correct file command

decide not to convert and go back to print and adjust buffer sizes

adjusted size of variable and cleaned up code, now works

tried not using encrypted and decrypted to store and it caused more issues, reverting

still only display red dots.

```
                              ┌──────────────────────┐        ┌──────────────────────┐
                              │ open write and close │───────▶│ make counter_fun     │
                              │                      │        │ and counter_rec files│
                              └──────────────────────┘        └──────────────────────┘
┌──────────────┐
│              │
│    task2     │
│              │
└──────┬───────┘
       │
       ▼
┌──────────────────┐        ┌──────────────────────┐
│ lets try to plan │───────▶│  recursive function  │
│ better than task1: 2│     └──────────┬───────────┘
│    programs      │                   │
└────────┬─────────┘                   ▼
         │                  ┌──────────────────────────┐
         ▼                  │ very similar to loop function│
┌──────────────────┐        │ almost everything else in the│
│ function count using│     │ code will be same. just modify│
│ ecx loop and     │        │ the count function itself a little│
│ increment counter│        └──────────┬───────────┘
│ 20000 times      │                   │
└────────┬─────────┘                   ▼
         │                  ┌──────────────────────────┐
         ▼                  │ instead of loop, outer    │
┌──────────────────┐        │ function for pop and push │
│ convert each digit to│    │                           │
│ ascii function   │        │ inner function that is    │
└────────┬─────────┘        │ recursively called to     │
         │                  │ increment counter         │
         ▼                  └──────────┬───────────┘
┌──────────────────┐                   │
│ append counter and│                  │
│ newline          │                   │
└────────┬─────────┘                   │
         │                             ▼
         ▼                  ┌──────────────┐
┌──────────────┐            │  test works  │
│  test works  │            └──────┬───────┘
└──────┬───────┘                   │
       │                           │
       └───────────┬───────────────┘
                   ▼
        ┌──────────────────┐                                    ┌──────────────────┐
        │ okay all set up for│         ┌──────────────────┐      │ write up in pdf  │
        │ time command     │          │ compare worst times│────▶│                  │
        └────────┬─────────┘          └─────────┬────────┘       └──────────────────┘
                 │                              ▲                          ▲
                 ▼                              │                          │
        ┌──────────────────┐          ┌──────────────────┐      ┌──────────────────┐
        │ says non zero code,│        │ run 6 times each │─────▶│ compare average  │
        │ and shows no time │         │ function.        │      │ times            │
        └────────┬─────────┘          └─────────┬────────┘      └──────────────────┘
                 │                              ▲
                 ▼                              │
        ┌──────────────────┐          ┌──────────────────┐
        │ test it on several│         │ make scripts to  │
        │ different programs,│        │ repeat runs      │
        │ same result      │         └─────────┬────────┘
        └────────┬─────────┘                   ▲
                 │                              │
                 ▼                  ┌──────────────────────────┐
        ┌──────────────────┐        │ set start time, echo run  │
        │ find alternative │───────▶│ executable, end time,     │
        │ method           │        │ execution time = end - start│
        └──────────────────┘        │ echo print to terminal and│
                                    │ append to txt file        │
                                    └──────────────────────────┘
```

Data:
The table below displays the seconds for each function and their 6 runs. It also shows the average for each function type.

	Recursive	Function (Loop)
run 1	0.00406456	0.00282121
run 2	0.00324845	0.00304127
run 3	0.0030756	0.00281644
run 4	0.0029459	0.00295281
run 5	0.00277233	0.00274944
run 6	0.00458097	0.00283933
average	0.003447968333	0.002870083333

Times of Runs
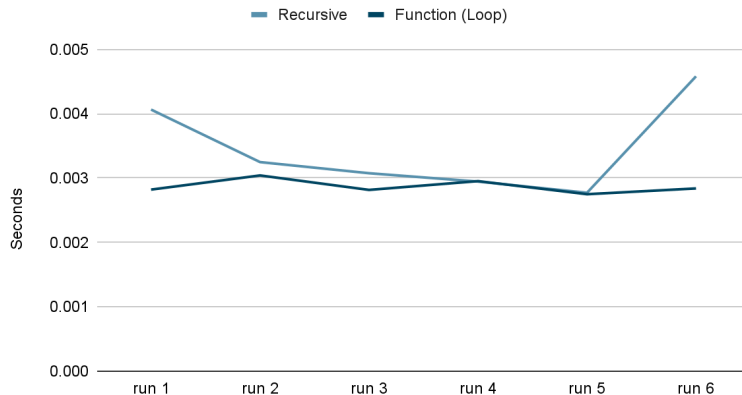
Average Times
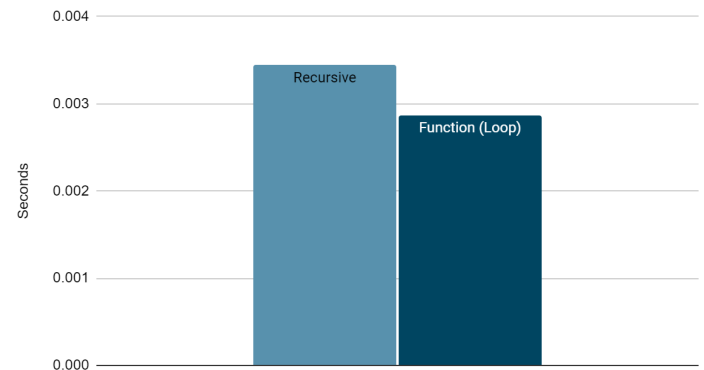
Analysis

Across the runs, the loop function has lower execution times. Additionally the loop function has lower average time. Overall, for a counter of 20,000, recursion performs worse in terms of execution time. The loop function is more efficient because it uses a simple iterative approach. Whereas the recursive function must maintain a call stack and that can lead to overhead and extra computation cost.

In conclusion, the function using a loop is more efficient than the recursive function in this specific case. However, the choice between recursion and loops depends on the problem's nature.