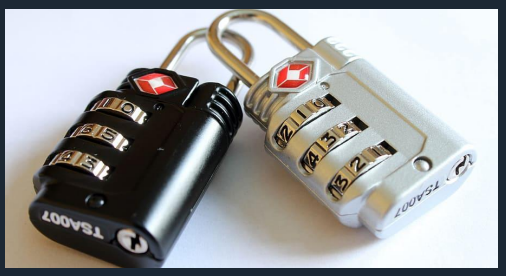
CoE 163

Computing Architectures and Algorithms

Software Exercise 01: Profiling and Assembly

Synopsis: Formulate and profile two different algorithms to compute the minimum number of steps needed to unlock some combination locks

Problem Statement



Combination locks are one of the easiest locks to pick without specialized equipment. These locks are unlocked when the dials in them show the unlock combination. Each dial contains digits from 0 to 9, and the numbers wrap around (i.e. incrementing from 9 will turn that dial to 0). Anyone can "brute-force" unlock them by trying out each and every combination possible.

You are working for a lock company and you are assigned a task of knowing the minimum number of steps needed to unlock a combination lock given its initial and unlocked final value for quality control purposes. A "step" is defined as an increment or decrement in value of a digit by one. For example, if the initial configuration of a lock shows 1239 and the unlock value is 2348, four (4) steps are only needed, like so:

- 1 -> 2 - one step
- 2 -> 3 - one step
- 3 -> 4 - one step
- 9 -> 8 - one step

Since you expect to compute locks with an arbitrary number of combination dials, you have decided to formulate two programs and choose the fastest one, with one made using your preferred programming language. Because of your overzealousness and also curiosity on whether you can squeeze more time from your tests, you have decided to write the other program using a mixture of C/C++/Rust and assembly.

Input

Input to your program would be three numbers denoting the number of combination locks N , the initial configuration of the lock S, and the unlock configuration E respectively.

Output

Your program should output the minimum number of steps P needed to change the lock from its initial to the final unlock configuration.

Constraints

$0 < N \leq 100$

Sample Inputs/Outputs

Sample Input 1

```
4 1239 2348
```

Sample Output 1

```
4
```

Sample Input 2

```
1 2 3
```

Sample Output 2

```
1
```

Sample Input 2

```
4 1234 9899
```

Sample Output 2

```
15
```

Additional Description/Requirements

In addition to the algorithm, we expect you to have at least two programs for the simulation. First, formulate a function for computing the minimum number of steps. The function should take in three parameters N, S, and E, and return an integer P. Then, enclose that function call in a loop. In a language named C, the function signature would look like this:

```
int find_min_combi(int num_dials, int start, int end);
```

Then, enclose this function in a loop that runs it at least 1000 times, corresponding to processing 1000 combination locks. For example, in an ancient language named Turbo Pascal, this is how you may write your profiling algorithm.

```
program CombiLockTest;

var i: integer;

begin
  { TIMER start }
  for i:= 1 to 1000 do begin
     { COMBILOCK CALCULATOR here }
  end;
  { TIMER end }
end.
```

The profiling algorithm should be run on two sets of test cases - one whose locks are of the a) *same length*, and of b) *different lengths*.

You may use any programming language for your first program, but the second program should be written at least partially in x86_64 assembly. You may either code the second program in assembly as a whole using your preferred assembler (NASM is used in the references) or use inline assembly only for the function in C/C++/Rust. Profile each of your two programs using your preferred profiling algorithm.

You also would like to write a short journal entry on which program you choose as your tester. Explain why that algorithm was correct, how long did the two or more programs take to run at least 1000 times, and reasons why each of them ran faster or slower relative to others. Also explain the differences between the programs and which of those are better in terms of runtime, programming effort, and code organization.

Upload both your programs (as two or three single source code files; in TXT if the system does not support the file extension) and your short journal (PDF or TXT file) to your remote work repository named UVLe.

Grading Rubric

30% Algorithm code and correctness (preferred language)
15% Algorithm code and correctness (C/C++/Rust and x86_64 asm)
20% Profiling code (preferred language)
10% Profiling code (C/C++/Rust and x86_64 asm)
25% Short journal