Lab 3

In Lab 3 you will use the following F# program to eliminate all non-prime numbers from a list leaving a list of prime numbers as the result. This example can be found at: https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/lists

1. You should first try running the program and look at the generated list. Here is the program:

```
let main argv =
    let IsPrimeMultipleTest n  x =  x = n || x % n <> 0

    let rec RemoveAllMultiples listn listx =
        match listn with
        | head :: tail -> RemoveAllMultiples tail (List.filter (IsPrimeMultipleTest head) listx)
        | [] -> listx

    let GetPrimesUpTo n =
        let max = int (sqrt (float n))
        RemoveAllMultiples [ 2 .. max ] [ 1 .. n ]

    printfn "Primes Up To %d:\n %A" 10000 (GetPrimesUpTo 10000)
```

```
Primes Up To 10000:
[1; 2; 3; 5; 7; 11; 13; 17; 19; 23; 29; 31; 37; 41; 43; 47; 53; 59; 61; 67; 71; 73; 79; 83; 89; 97; 101; 103; 107; 109; 113; 1
```

2. After you have compiled and executed the program, you measure the CPU time by utilizing the TotalProcessorTime property of System.Diagnostics.Process. This returns a TimeSpan object which indicates the amount of CPU time that the associated process has spent running the F# function Here's how you use it:

```
open System.Diagnostics

// Setup for determining CPU time
let proc = Process.GetCurrentProcess()
let beginTime = proc.TotalProcessorTime

// your code to select primes

let cpuTime = (proc.TotalProcessorTime - beginTime).TotalMilliseconds
printfn "CPU Time = %d ms\n" (int64 cpuTime)
```

```
open System
open System.Diagnostics

[<EntryPoint>]
let main argv =
    // Setup for determining CPU time
    let proc = Process.GetCurrentProcess()
    let beginTime = proc.TotalProcessorTime

    let isPrimeMultipleTest n  x =  x = n || x % n <> 0

    let rec removeAllMultiples listn listx =
        match listn with
        | head :: tail -> removeAllMultiples tail (List.filter (isPrimeMultipleTest head) listx)
        | [] -> listx

    let getPrimesUpTo n =
        let max = int (sqrt (float n))
        removeAllMultiples [ 2 .. max ] [ 1 .. n ]

    printfn "Primes Up To %d:\n %A" 10000 (getPrimesUpTo 10000)

    let cpuTime = (proc.TotalProcessorTime - beginTime).TotalMilliseconds
    printfn "CPU Time = %d ms\n" (int64 cpuTime)
    0 // return an integer exit code
```

```
Primes Up To 10000:
[1; 2; ...]  // shortened for readability
CPU Time = 62 ms
```

3. Next you convert the aforementioned program to use a for-do loop instead of recursion and pattern matching. Use a mutable list that is initialized with natural numbers from [1..1000000]. Apply a filter (similar to the one in the given program) to this list to eliminate all non-prime numbers. The argument to the filter is a reference to the mutable list. Compile and run the program. Make sure it generates the list of prime numbers.

```
open System
open System.Collections.Generic

[<EntryPoint>]
let main argv =
    let removeAllMultiples listn listx =
        let mutable output = new List<int>()
        output.AddRange(listx)
        for i in listn do
            for j in listx do
                if (j % i = 0 && j <> i) then
                    output.Remove(j) |> ignore
        output


    let getPrimesUpTo n =
        let max = int (sqrt (float n))
        removeAllMultiples [ 2 .. max ] [ 1 .. n ]

    printfn "Primes Up To %d:\n %A" 10000 ((getPrimesUpTo 10000).ToArray())
    0 // return an integer exit code
```

```
Primes Up To 10000:
[|1; 2; 3; 5; 7; 11; 13; 17; 19; 23; 29; 31; 37; 41; 43; 47; 53; 59; 61; 67; 71; 73; 79; 83; 89; 97; 101; 103; 107; 109; 113;
```

4. Finally, measure the CPU time taken by the non-recursive program.

```
Primes Up To 10000:
[|1; 2; ...|] // shortened for readability
CPU Time = 265 ms
```