```go
// this code is achieved with golang 1.17
// package reflect is used to implement Calling Subprograms Indirectly

package main

import (
    "flag"
    "fmt"
    "log"
    "math"
    "strconv"
    "sync"
    "time"
)
import "reflect"

var primeLTE101 = []int{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}
// PrimeArray is used to generate prime number until sqrt(max)
var PrimeArray = []int{}
var ResultChan chan int
var ThreadLock sync.WaitGroup
var OutputLock sync.WaitGroup
var OutputCounter int
var JustCount bool


type IndirectMethodsLT100 struct {}

func (t *IndirectMethodsLT100) CheckPrime (primeCheckMin,primeCheckMax int) int {
    outputCounter :=0
    for _,primeNum:= range primeLTE101{
        if primeNum > primeCheckMax{
            break
        }
        if primeNum >= primeCheckMin{
            if !JustCount{
                fmt.Printf("%d is a prime number \n",primeNum)
            }

            outputCounter++
        }
    }
    return outputCounter
}

type IndirectMethodsGTE100 struct {}

func (t *IndirectMethodsGTE100) CheckPrime (primeCheckMin,primeCheckMax int)int{
    outputCounter :=0
    // just output the prime number in the list first
    if primeCheckMin<primeLTE101[len(primeLTE101)-1]{
        for _,primeNum:= range primeLTE101{
            if primeNum >= primeCheckMin{
                if !JustCount{
                    fmt.Printf("%d is a prime number \n",primeNum)
                }
                outputCounter++
            }
        }
        primeCheckMin= primeLTE101[len(primeLTE101)-1]+2
    }
    if primeCheckMin%2==0{
        primeCheckMin++
    }
    // start to use the prime list to check for larger prime number
    for currentNum := primeCheckMin;currentNum<=primeCheckMax;currentNum+=2{
        checkMax := math.Sqrt(float64(currentNum))
        //check with each prime number
        for _,primeNum := range primeLTE101{
            //fmt.Printf("%f",checkMax)
            if float64(primeNum)> checkMax{
                //this is a prime number
                if !JustCount{
                    fmt.Printf("%d is a prime number \n",currentNum)
                }
                outputCounter++
                break
            }
            if currentNum%primeNum==0{
                //not prime
                break
            }
        }
    }
    return outputCounter
}

type IndirectMethodsGTE10000 struct {}

func (t *IndirectMethodsGTE10000) CheckPrime (primeCheckMin,primeCheckMax int)int{
    // populate the prime list that the largest prime should be gte sqrt(primeCheckMax) first
    start := time.Now()
    OutputCounter =0
    PrimeArray = primeLTE101
    primeNumberAtLeast := int(math.Ceil(math.Sqrt(float64(primeCheckMax))))
    for currentNum := 103;;currentNum+=2{
        checkMax := int(math.Ceil(math.Sqrt(float64(currentNum))))
        //check with each prime number
        if PrimeArray[len(PrimeArray)-1] >=primeNumberAtLeast{
            // populate completed
            break
        }
        for _,primeNum := range PrimeArray {
            if primeNum> checkMax{
                //this is a prime number
                PrimeArray = append(PrimeArray,currentNum)
                //fmt.Printf("add %d to array",currentNum)
                break
            }
            if currentNum%primeNum==0{
                //not prime
                break
            }
        }
    }
    // PrimeArray populate completed
    // output the prime number in the list first
    if primeCheckMin< PrimeArray[len(PrimeArray)-1]{
        for _,primeNum:= range PrimeArray {
            if primeNum >= primeCheckMin{
                //fmt.Printf("%d is a prime number \n",primeNum)
                OutputCounter++
            }
        }
        primeCheckMin= PrimeArray[len(PrimeArray)-1]+2
    }
    if primeCheckMin%2==0{
        primeCheckMin++
    }
    // start to use the prime list to check for larger prime number
    // use multithreading for larger dataset
    threadNum:=64
    // use single thread for little dataset
    if (primeCheckMax-primeCheckMin)<=10{
        threadNum = 1
    }else if (primeCheckMax-primeCheckMin)/2<=threadNum{
        threadNum = (primeCheckMax-primeCheckMin)/2
    }

    ResultChan = make(chan int,threadNum*100)
    ThreadLock = sync.WaitGroup{}
    // use PrimeArray to check larger prime number
    for i:=0;i<threadNum;i++{
        // add a lock for each thread to avoid early exit
        ThreadLock.Add(1)
        // provision specific range for each thread
        subCheckStart := primeCheckMin+int(math.Floor(float64(i)*float64(primeCheckMax-primeCheckMin)/float64(threadNum)))
        subCheckEnd := primeCheckMin+int(math.Floor(float64(i+1)*float64(primeCheckMax-primeCheckMin)/float64(threadNum)))-1
        if subCheckStart%2==0{
            subCheckStart+=1
        }
        if i == threadNum-1{
            subCheckEnd++
        }
        //log.Printf("start %d,end %d",subCheckStart,subCheckEnd)
        // start an async go routine for specific data range
        go CheckPrimeSub(subCheckStart,subCheckEnd)
    }
    OutputLock.Add(1)
    // start an async go routine to output
    go outputResult()
    ThreadLock.Wait()
    fmt.Printf("finish calculation in %f second\n",time.Since(start).Seconds())
    close(ResultChan)
    OutputLock.Wait()
    return OutputCounter
}

// outputResult is called by go routine to read from ResultChan channel and output/count
func outputResult(){
    for primeNum := range ResultChan{
        OutputCounter++
        if !JustCount{
            fmt.Printf("%d is a prime number\n",primeNum)
        }
    }
    OutputLock.Done()
}

// CheckPrimeSub is called by go routine to calculate prime number in range
func CheckPrimeSub(primeCheckMin,primeCheckMax int){
    for currentNum := primeCheckMin;currentNum<=primeCheckMax;currentNum+=2{
        checkMax := int(math.Ceil(math.Sqrt(float64(currentNum))))
        //check with each prime number
        for _,primeNum := range PrimeArray {
            if primeNum> checkMax{
                //this is a prime number
                ResultChan <- currentNum
                break
            }
            if currentNum%primeNum==0{
                //not prime
                break
            }
        }
    }
    ThreadLock.Done()
}

// main takes two positive int input for range and a -counting flag for only count how many prime number is in range
// The indirect subprogram is defined at primeMethod
// primeMethod is called later on, by the input size
// primeMethod will choose the appropriate function from IndirectMethodsGTE10000/IndirectMethodsGTE100/IndirectMethodsLT100
// the Theory used in defining a prime number is:
// if a number can't be fully divided by each prime number less than the root(sqrt) of the number. Then this number is a prime number
func main() {
    flag.BoolVar(&JustCount, "counting", false, "set this flag for massive dataset")
    flag.Parse()
    primeCheckStartStr := flag.Args()[0]
    primeCheckEndStr := flag.Args()[1]
    primeCheckStart,err := strconv.Atoi(primeCheckStartStr);if err!=nil{
        log.Fatalf("input1 is not valid int")
    }
    primeCheckEnd,err := strconv.Atoi(primeCheckEndStr);if err!=nil{
        log.Fatalf("input2 is not valid int")
    }
    if primeCheckStart>primeCheckEnd{
        //swap
        primeCheckStart+=primeCheckEnd
        primeCheckEnd=primeCheckStart-primeCheckEnd
        primeCheckStart=primeCheckStart-primeCheckEnd
    }
    if primeCheckStart<0{
        log.Fatalf("minus input detected")
    }
    // use of Call() method
    // decide which function to use
    var funcToUse interface{}
    if primeCheckEnd>=10000{
        funcToUse = new(IndirectMethodsGTE10000)
    }else if primeCheckEnd >=100{
        funcToUse = new(IndirectMethodsGTE100)
    }else{
        funcToUse = new(IndirectMethodsLT100)
    }

    fmt.Printf("outputting prime number between %d,%d\n",primeCheckStart,primeCheckEnd)
    // define the function that will be used later
    primeMethod := reflect.ValueOf(funcToUse).MethodByName("CheckPrime")
    reflectValue := make([]reflect.Value, primeMethod.Type().NumIn())
    reflectValue[0]=reflect.ValueOf(primeCheckStart)
    reflectValue[1]=reflect.ValueOf(primeCheckEnd)
    // indirectly call the function upon runtime
    val := primeMethod.Call(reflectValue)
    if val[0].Int()==0{
        fmt.Printf("no prime number between %d,%d\n",primeCheckStart,primeCheckEnd)
    }else{
        fmt.Printf("%d prime number between %d,%d\n",val[0].Int(),primeCheckStart,primeCheckEnd)
    }
}
```