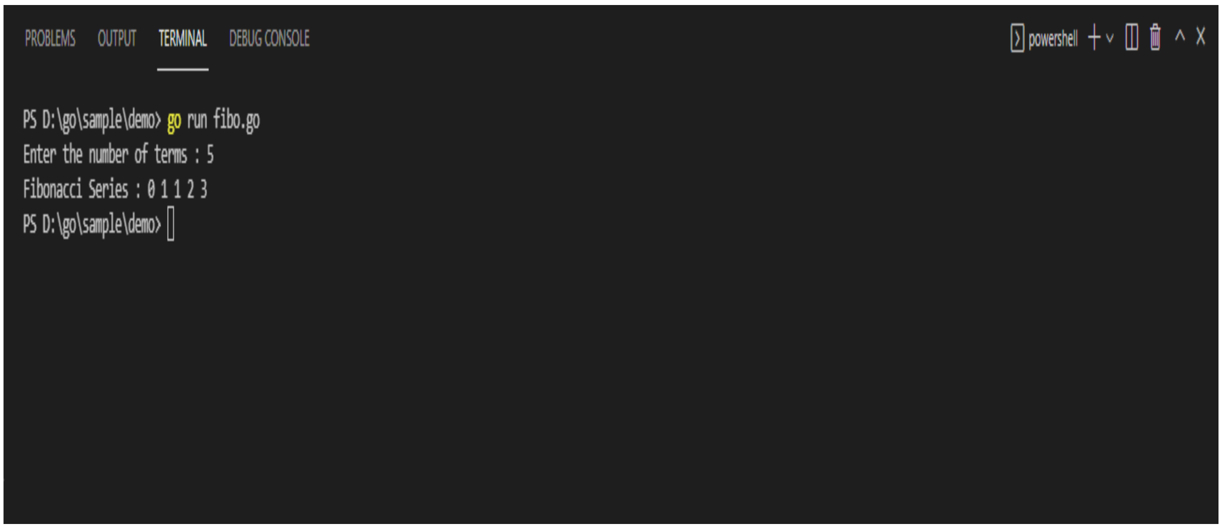
1. Go program to generate Fibonacci sequence.

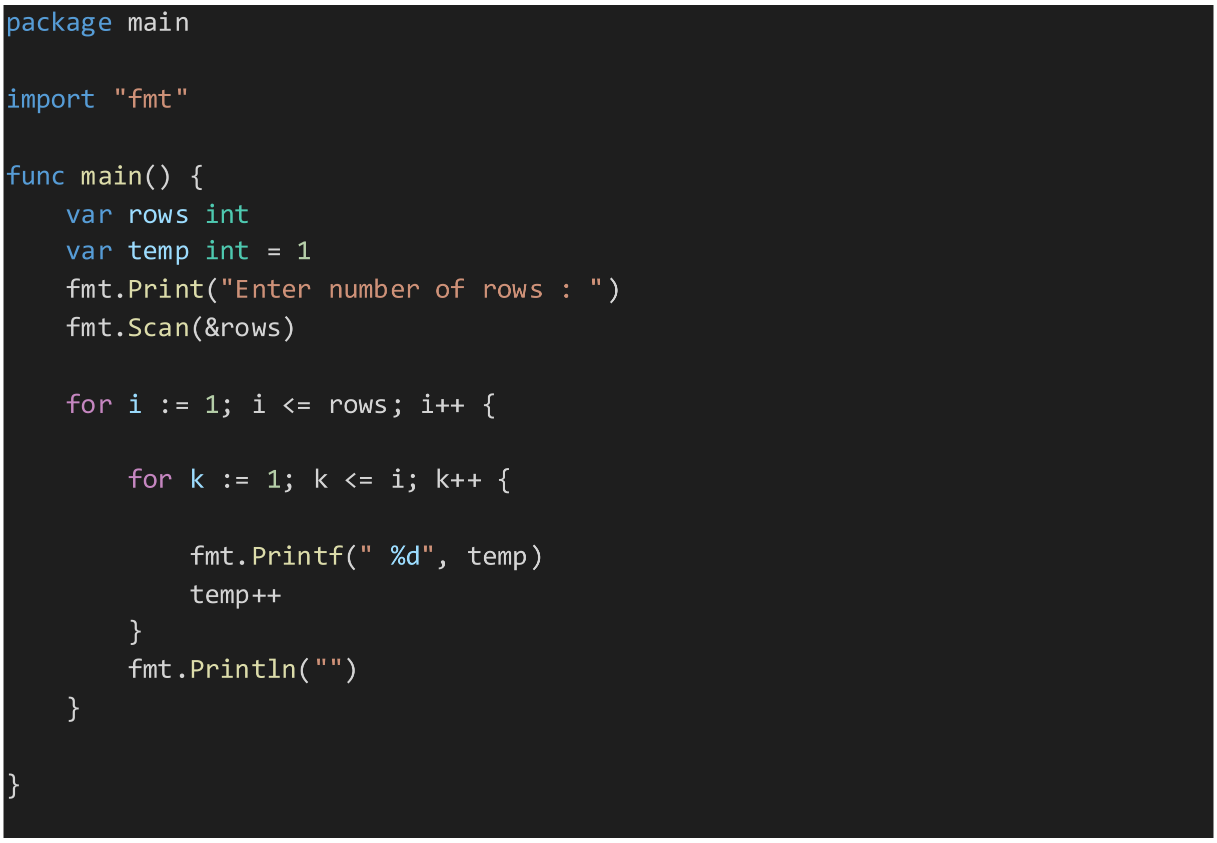
```go
package main

import "fmt"

func main() {
    var n int
    t1 := 0
    t2 := 1
    nextTerm := 0

    fmt.Print("Enter the number of terms : ")
    fmt.Scan(&n)
    fmt.Print("Fibonacci Series :")
    for i := 1; i <= n; i++ {
        if i == 1 {
            fmt.Print(" ", t1)
            continue
        }
        if i == 2 {
            fmt.Print(" ", t2)
            continue
        }
        nextTerm = t1 + t2
        t1 = t2
        t2 = nextTerm
        fmt.Print(" ", nextTerm)
    }
}
```

Output:



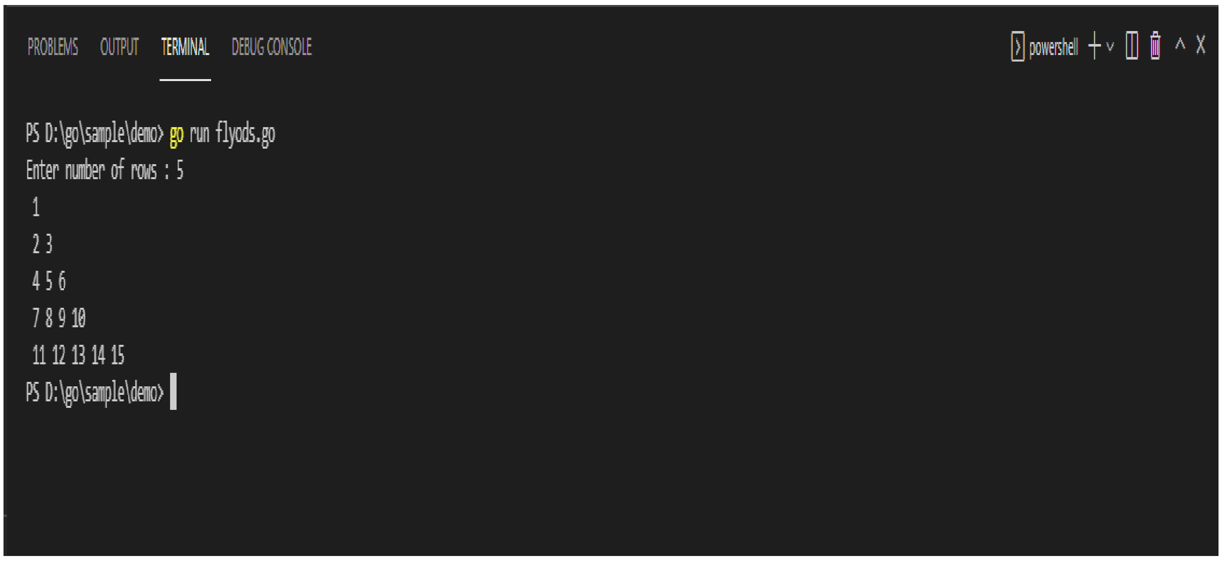
```
PS D:\go\sample\demo> go run fibo.go
Enter the number of terms : 5
Fibonacci Series : 0 1 1 2 3
PS D:\go\sample\demo> 
```

2. Program in go to print floyds triangle.



```go
package main

import "fmt"

func main() {
    var rows int
    var temp int = 1
    fmt.Print("Enter number of rows : ")
    fmt.Scan(&rows)

    for i := 1; i <= rows; i++ {

        for k := 1; k <= i; k++ {

            fmt.Printf(" %d", temp)
            temp++
        }
        fmt.Println("")
    }

}
```

Output:

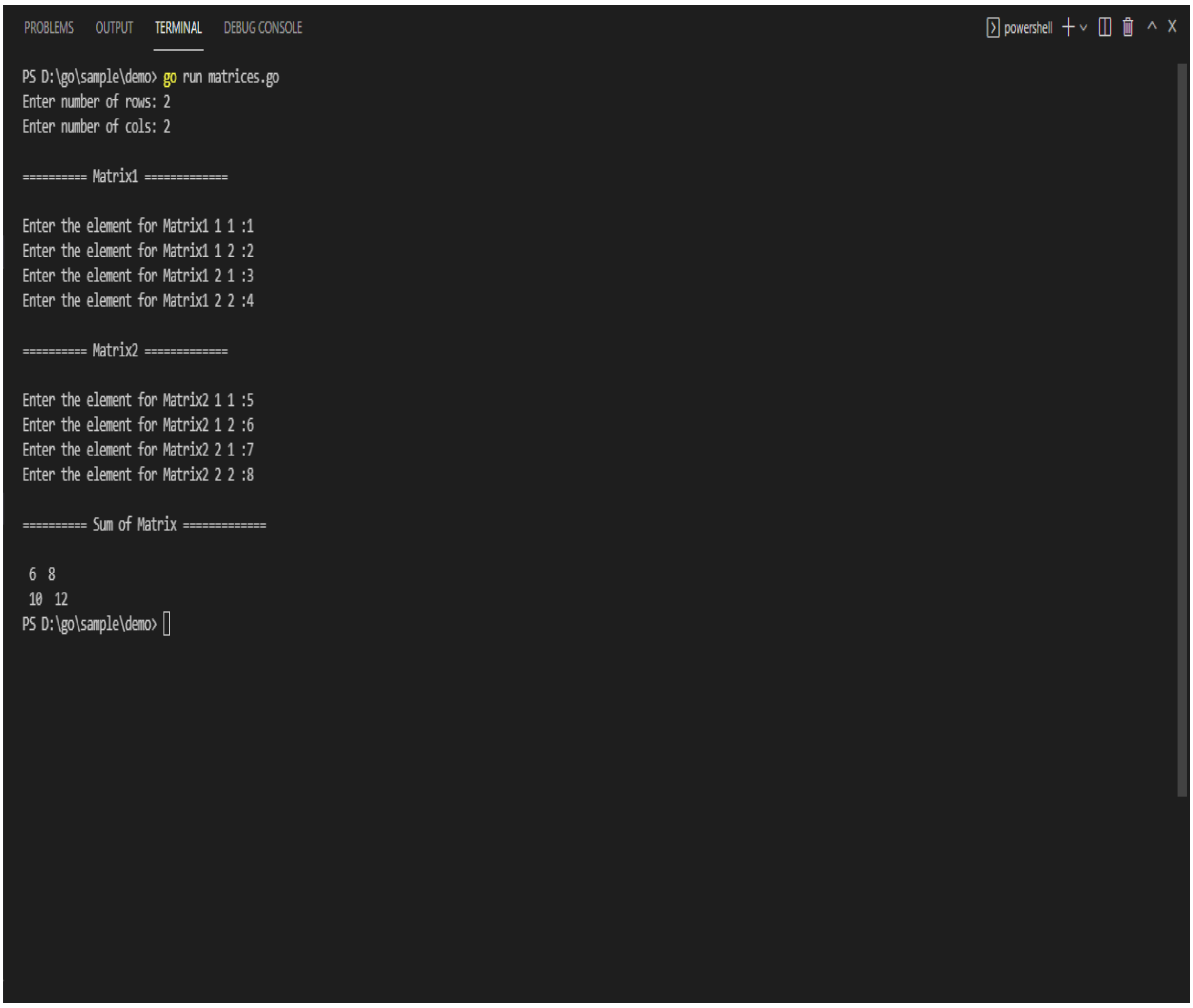


```
PS D:\go\sample\demo> go run flyods.go
Enter number of rows : 5
 1
 2 3
 4 5 6
 7 8 9 10
 11 12 13 14 15
PS D:\go\sample\demo>
```

3. Program to add two matrices using multidimensional array.

```go
package main

import "fmt"

func main() {
    var matrix1 [100][100]int
    var matrix2 [100][100]int
    var sum [100][100]int
    var row, col int
    fmt.Print("Enter number of rows: ")
    fmt.Scanln(&row)
    fmt.Print("Enter number of cols: ")
    fmt.Scanln(&col)

    fmt.Println()
    fmt.Println("========== Matrix1 =============")
    fmt.Println()
    for i := 0; i < row; i++ {
        for j := 0; j < col; j++ {
            fmt.Printf("Enter the element for Matrix1 %d %d :", i+1, j+1)
            fmt.Scanln(&matrix1[i][j])
        }
    }

    fmt.Println()
    fmt.Println("========== Matrix2 =============")
    fmt.Println()

    for i := 0; i < row; i++ {
        for j := 0; j < col; j++ {
            fmt.Printf("Enter the element for Matrix2 %d %d :", i+1, j+1)
            fmt.Scanln(&matrix2[i][j])
        }
    }

    for i := 0; i < row; i++ {
        for j := 0; j < col; j++ {
            sum[i][j] = matrix1[i][j] + matrix2[i][j]
        }
    }

    fmt.Println()
    fmt.Println("========== Sum of Matrix =============")
    fmt.Println()

    for i := 0; i < row; i++ {
        for j := 0; j < col; j++ {
            fmt.Printf(" %d ", sum[i][j])
            if j == col-1 {
                fmt.Println("")
            }
        }
    }

}
```

Output:

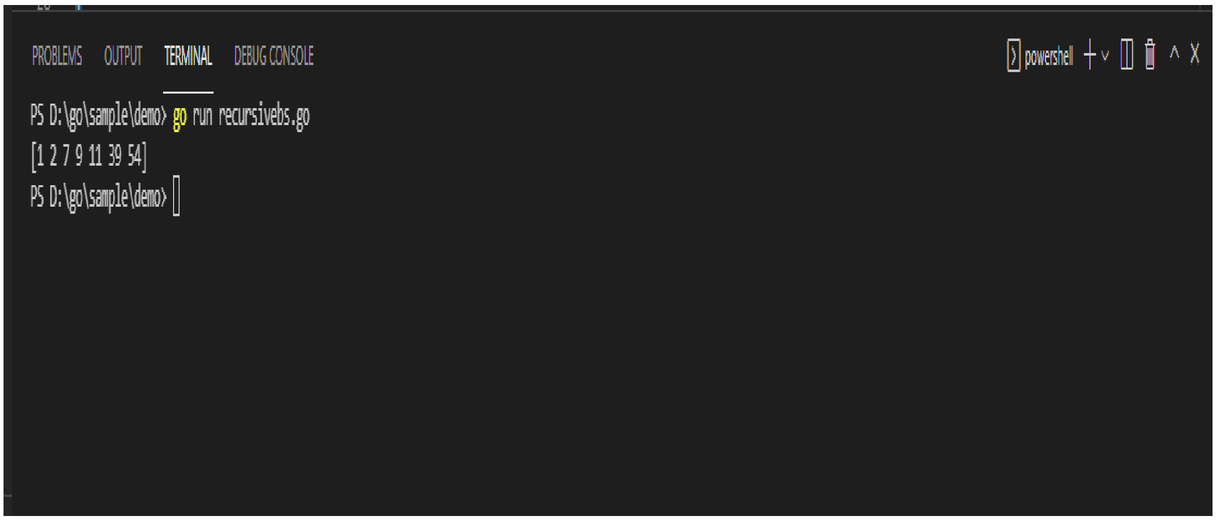


```
PS D:\go\sample\demo> go run matrices.go
Enter number of rows: 2
Enter number of cols: 2

========== Matrix1 ============

Enter the element for Matrix1 1 1 :1
Enter the element for Matrix1 1 2 :2
Enter the element for Matrix1 2 1 :3
Enter the element for Matrix1 2 2 :4

========== Matrix2 ============

Enter the element for Matrix2 1 1 :5
Enter the element for Matrix2 1 2 :6
Enter the element for Matrix2 2 1 :7
Enter the element for Matrix2 2 2 :8

========== Sum of Matrix ============

 6  8
 10  12
PS D:\go\sample\demo>
```

4. Sorting using recursive bubblesort.

```go
package main

import "fmt"

func bubbleSort(arr []int, size int) []int {
    if size == 1 {
        return arr
    }

    var i = 0
    for i < size-1 {
        if arr[i] > arr[i+1] {
            arr[i], arr[i+1] = arr[i+1], arr[i]
        }

        i++
    }

    bubbleSort(arr, size-1)

    return arr
}

func main() {
    var n = []int{1, 39, 2, 9, 7, 54, 11}

    fmt.Println(bubbleSort(n, len(n)))
}
```

Output:

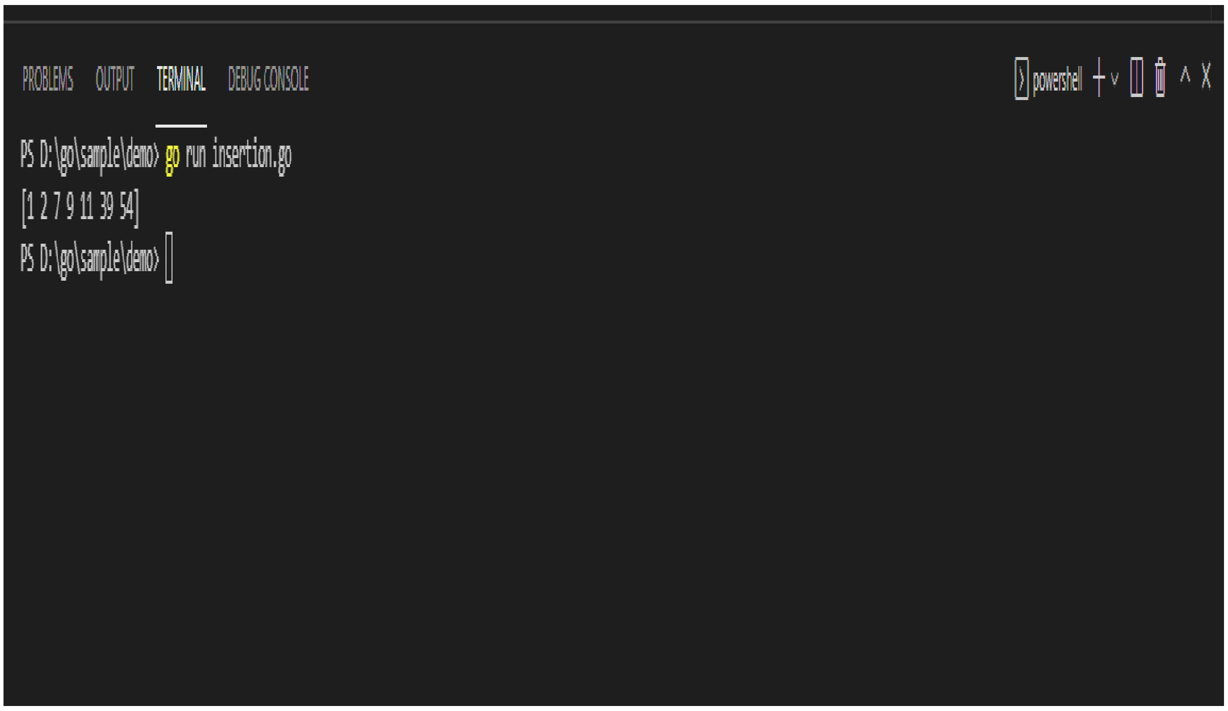


```
PS D:\go\sample\demo> go run recursivebs.go
[1 2 7 9 11 39 54]
PS D:\go\sample\demo>
```

5. Go program to sort an array using Insertion sort.

```go
package main

import "fmt"

func main() {
    var n = []int{1, 39, 2, 9, 7, 54, 11}

    var i = 1
    for i < len(n) {
        var j = i
        for j >= 1 && n[j] < n[j-1] {
            n[j], n[j-1] = n[j-1], n[j]

            j--
        }

        i++
    }

    fmt.Println(n)
}
```

Output:

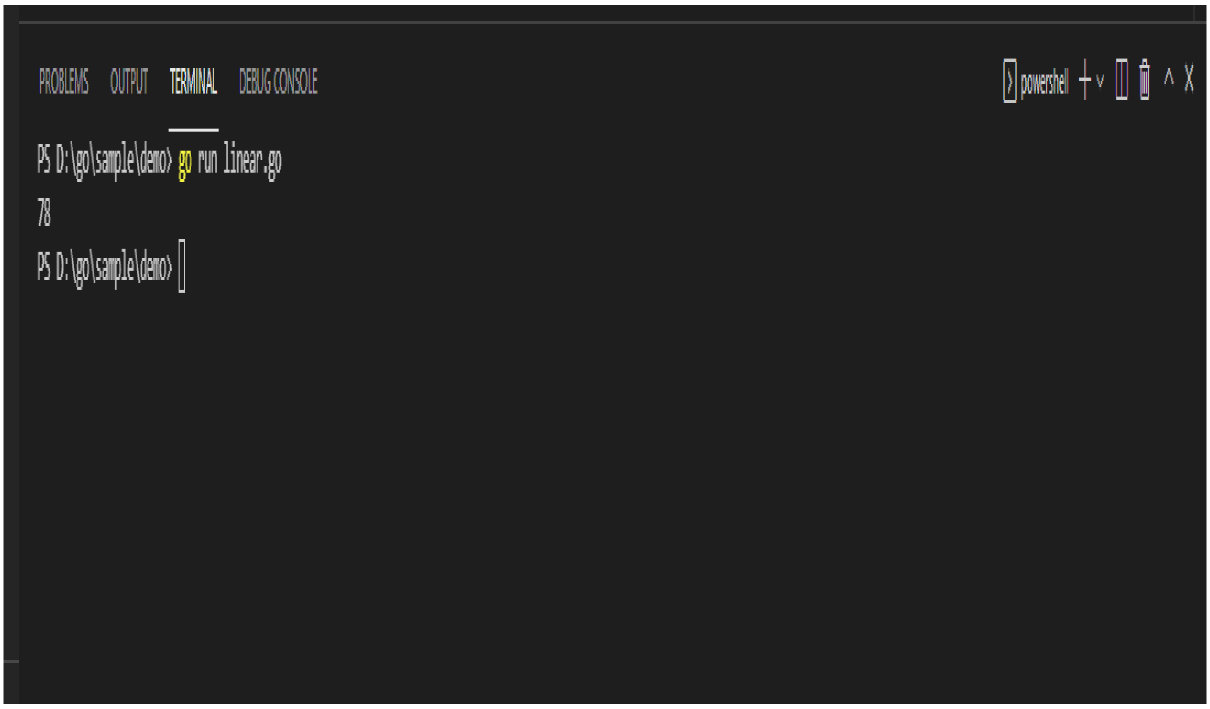


```
PS D:\go\sample\demo> go run insertion.go
[1 2 7 9 11 39 54]
PS D:\go\sample\demo>
```

6. Go program to search element in array using linear search.

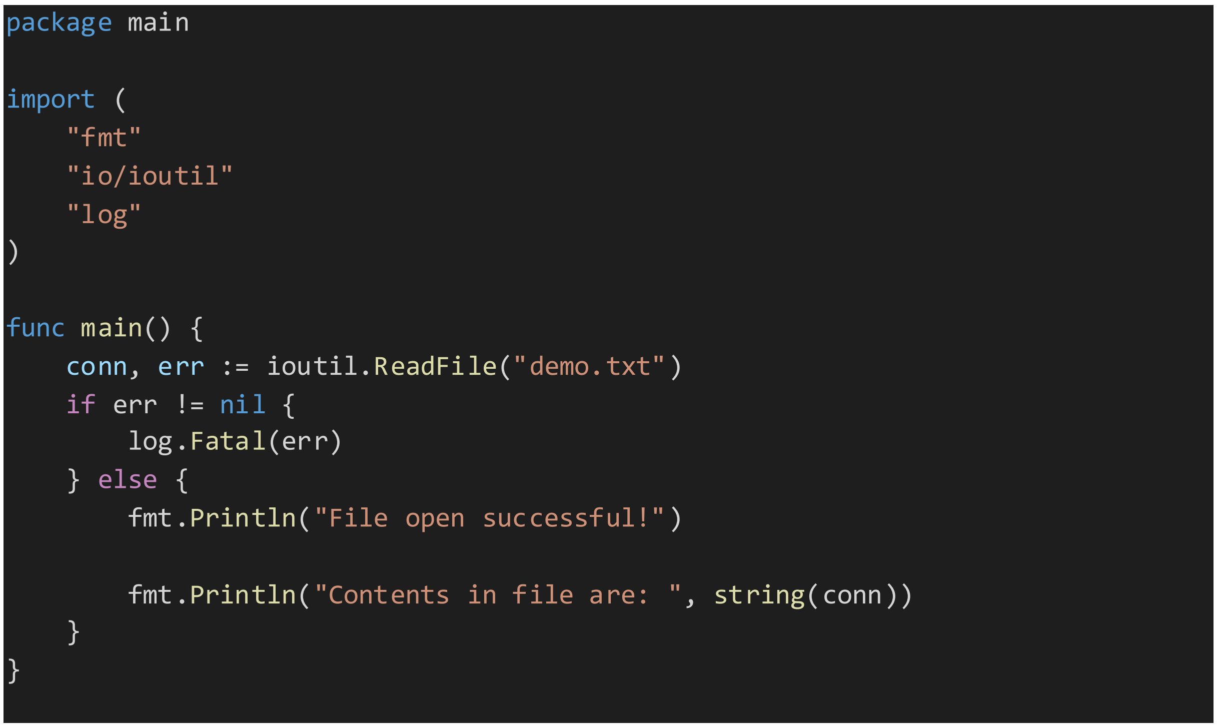
```go
package main

import "fmt"

func linearSort(arr []int, s int) int {
    for _, v := range arr {
        if s == v {
            return v
        }
    }

    return -1
}

func main() {
    var n = []int{9, 1, 33, 21, 78, 42, 4}

    fmt.Println(linearSort(n, 78))
}
```

Output:



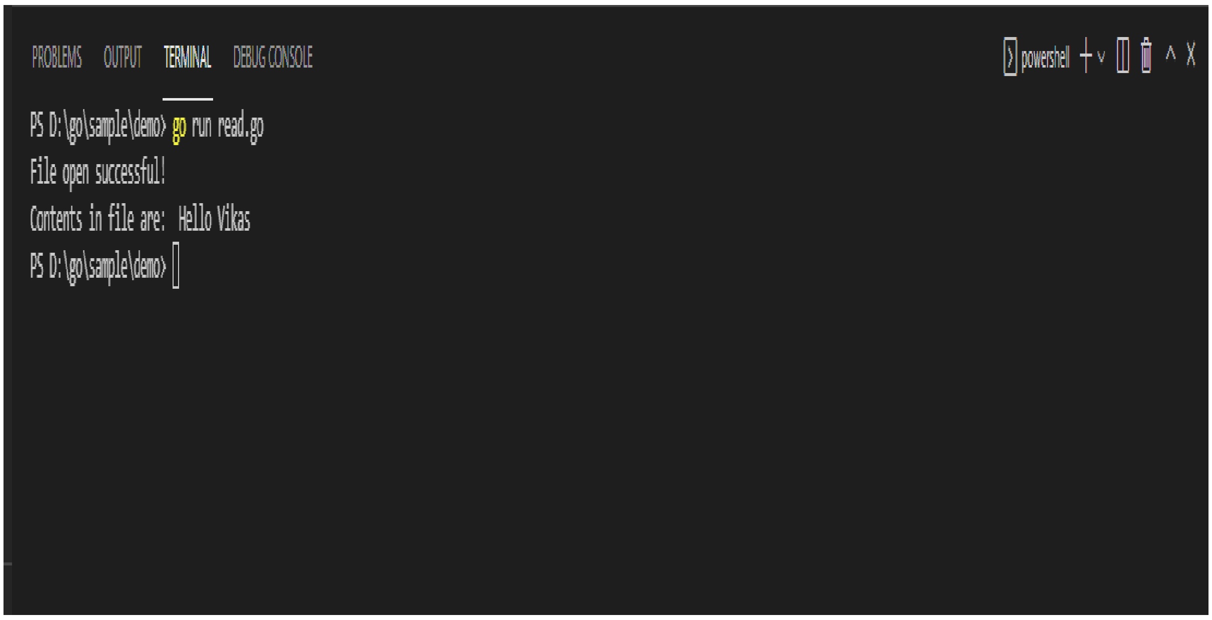
```
PS D:\go\sample\demo> go run linear.go
78
PS D:\go\sample\demo>
```

7. Go program to read a file.



```go
package main

import (
    "fmt"
    "io/ioutil"
    "log"
)

func main() {
    conn, err := ioutil.ReadFile("demo.txt")
    if err != nil {
        log.Fatal(err)
    } else {
        fmt.Println("File open successful!")

        fmt.Println("Contents in file are: ", string(conn))
    }
}
```

Output:

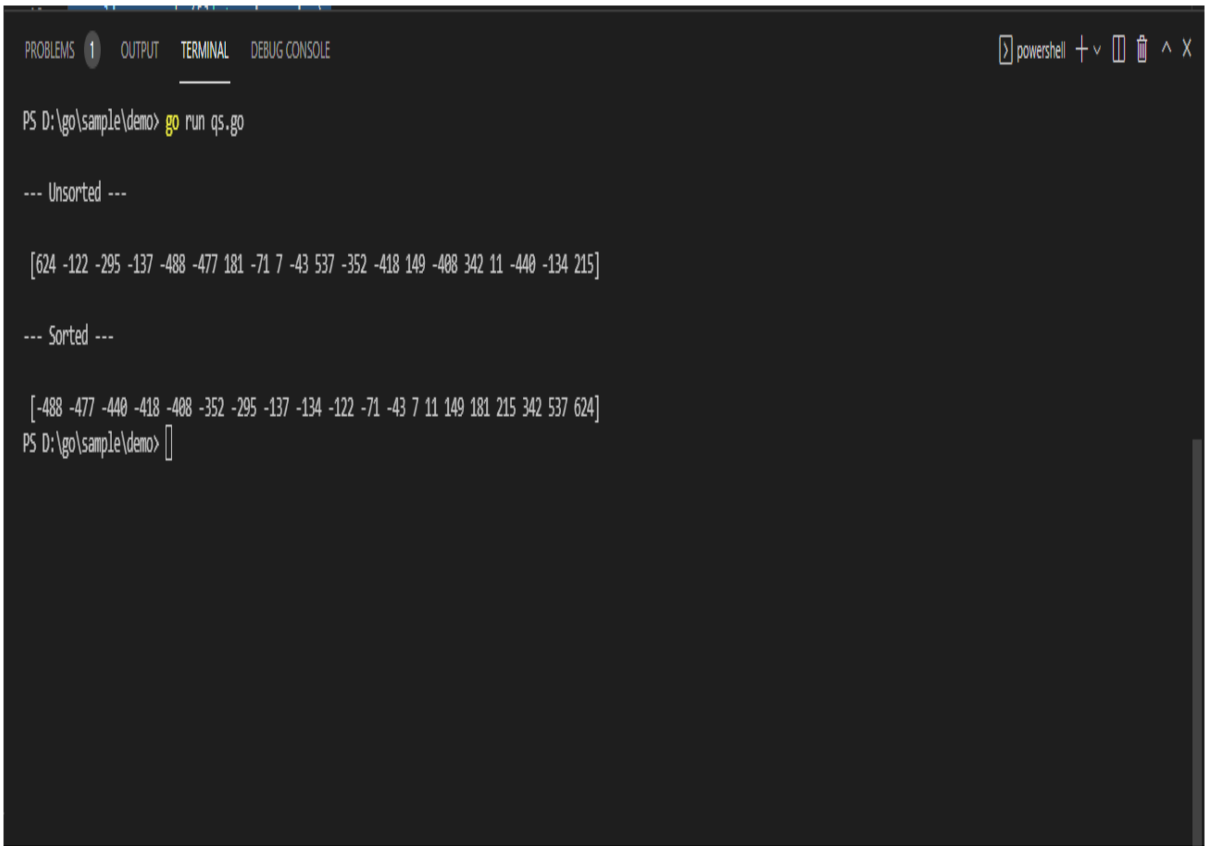


```
PS D:\go\sample\demo> go run read.go
File open successful!
Contents in file are:  Hello Vikas
PS D:\go\sample\demo> 
```

8. Go program to sort an array using quick sort.

```go
package main

import (
    "fmt"
    "math/rand"
    "time"
)

func main() {

    slice := generateSlice(20)
    fmt.Println("\n--- Unsorted --- \n\n", slice)
    quicksort(slice)
    fmt.Println("\n--- Sorted ---\n\n", slice)
}

func generateSlice(size int) []int {

    slice := make([]int, size, size)
    rand.Seed(time.Now().UnixNano())
    for i := 0; i < size; i++ {
        slice[i] = rand.Intn(999) - rand.Intn(999)
    }
    return slice
}

func quicksort(a []int) []int {
    if len(a) < 2 {
        return a
    }

    left, right := 0, len(a)-1

    pivot := rand.Int() % len(a)

    a[pivot], a[right] = a[right], a[pivot]

    for i, _ := range a {
        if a[i] < a[right] {
            a[left], a[i] = a[i], a[left]
            left++
        }
    }

    a[left], a[right] = a[right], a[left]
    quicksort(a[:left])
    quicksort(a[left+1:])

    return a
}
```

Output:

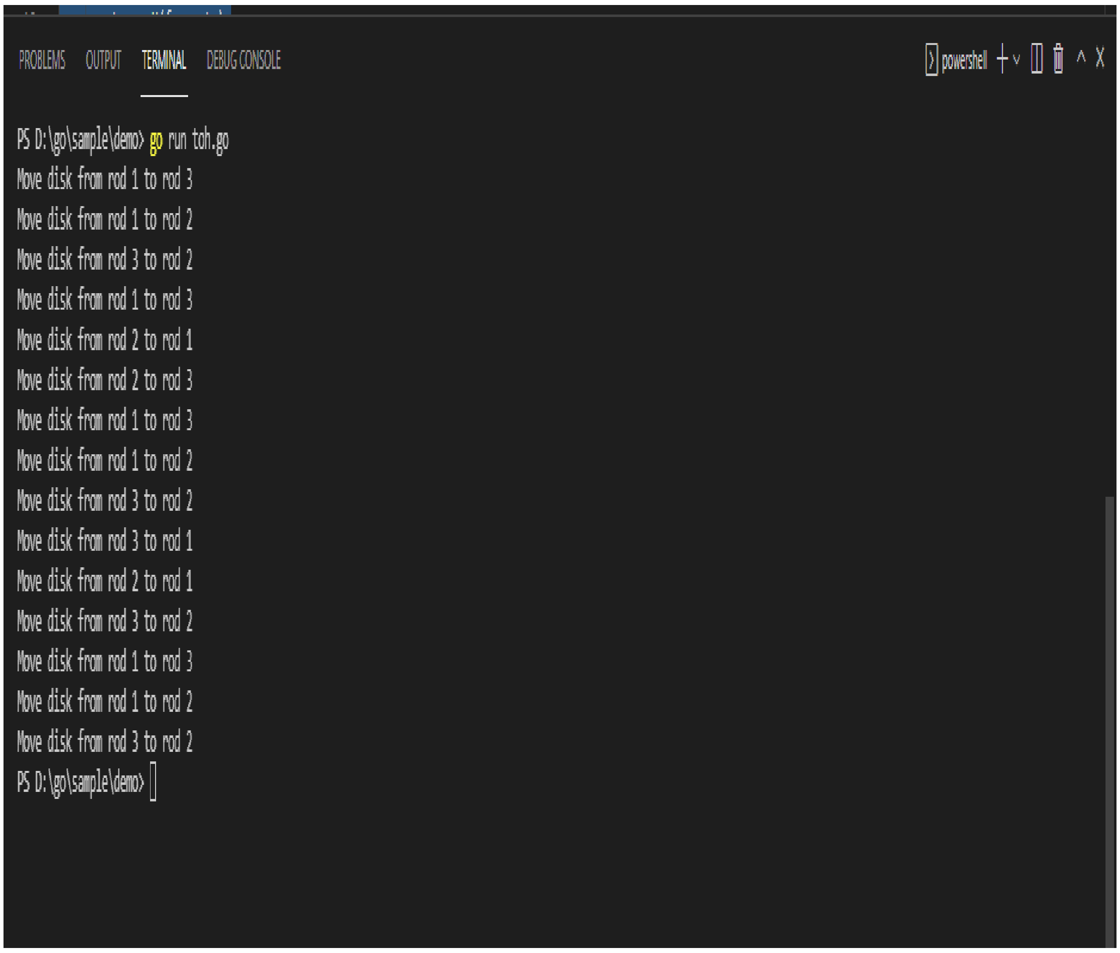


```
PS D:\go\sample\demo> go run qs.go

--- Unsorted ---

 [624 -122 -295 -137 -488 -477 181 -71 7 -43 537 -352 -418 149 -408 342 11 -440 -134 215]

--- Sorted ---

 [-488 -477 -440 -418 -408 -352 -295 -137 -134 -122 -71 -43 7 11 149 181 215 342 537 624]
PS D:\go\sample\demo>
```

9. Golang program to implement tower of Hanoi.

```go
package main

import "fmt"

type solver interface {
    play(int)
}

type towers struct {
}

func (t *towers) play(n int) {
    t.moveN(n, 1, 2, 3)
}

func (t *towers) moveN(n, from, to, via int) {
    if n > 0 {
        t.moveN(n-1, from, via, to)
        t.moveM(from, to)
        t.moveN(n-1, via, to, from)
    }
}

func (t *towers) moveM(from, to int) {
    fmt.Println("Move disk from rod", from, "to rod", to)
}

func main() {
    var t solver
    t = new(towers)
    t.play(4)
}
```

Output:

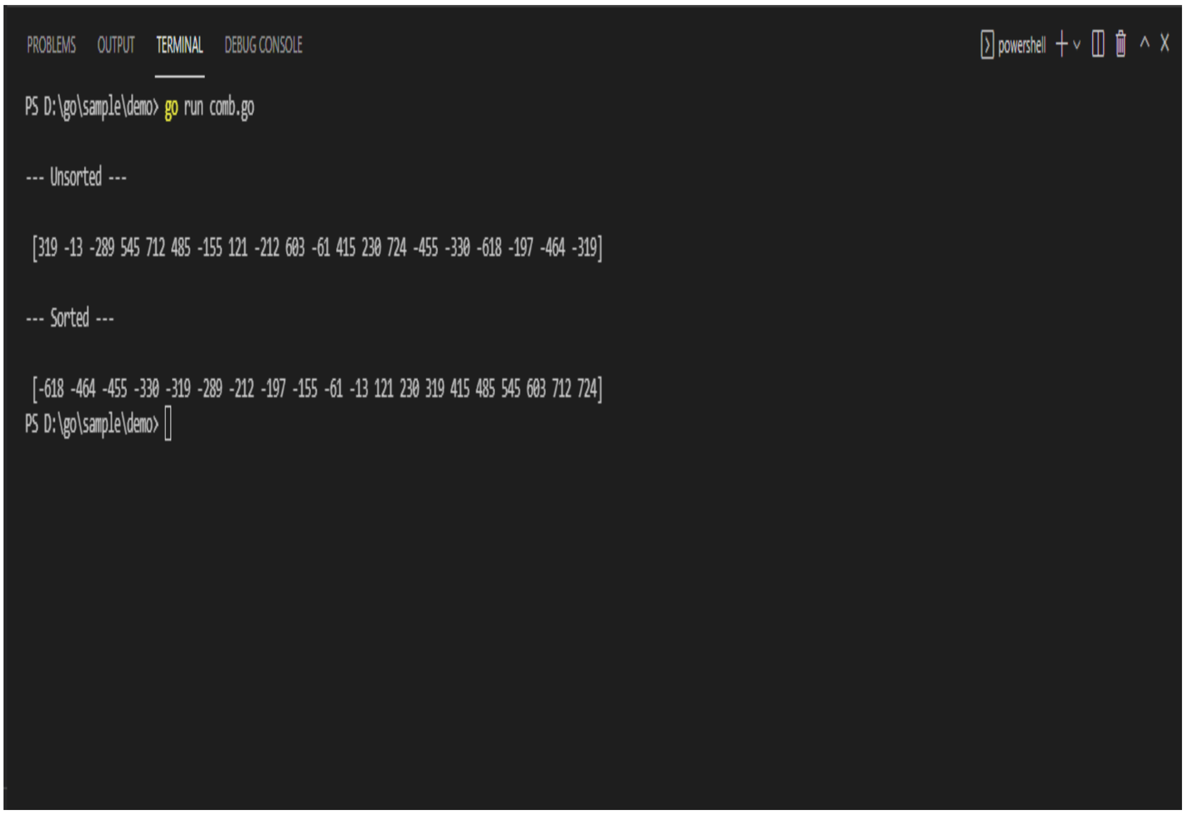


```
PS D:\go\sample\demo> go run toh.go
Move disk from rod 1 to rod 3
Move disk from rod 1 to rod 2
Move disk from rod 3 to rod 2
Move disk from rod 1 to rod 3
Move disk from rod 2 to rod 1
Move disk from rod 2 to rod 3
Move disk from rod 1 to rod 3
Move disk from rod 1 to rod 2
Move disk from rod 3 to rod 2
Move disk from rod 3 to rod 1
Move disk from rod 2 to rod 1
Move disk from rod 3 to rod 2
Move disk from rod 1 to rod 3
Move disk from rod 1 to rod 2
Move disk from rod 3 to rod 2
PS D:\go\sample\demo>
```

10. Go program to implement comb sort.

```go
package main

import (
    "fmt"
    "math/rand"
    "time"
)
func main() {

    slice := generateSlice(20)
    fmt.Println("\n--- Unsorted --- \n\n", slice)
    combsort(slice)
    fmt.Println("\n--- Sorted ---\n\n", slice)
}
func generateSlice(size int) []int {

    slice := make([]int, size, size)
    rand.Seed(time.Now().UnixNano())
    for i := 0; i < size; i++ {
        slice[i] = rand.Intn(999) - rand.Intn(999)
    }
    return slice
}
func combsort(items []int) {
    var (
        n       = len(items)
        gap     = len(items)
        shrink  = 1.3
        swapped = true
    )
    for swapped {
        swapped = false
        gap = int(float64(gap) / shrink)
        if gap < 1 {
            gap = 1
        }
        for i := 0; i+gap < n; i++ {
            if items[i] > items[i+gap] {
                items[i+gap], items[i] = items[i], items[i+gap]
                swapped = true
            }
        }
    }
}
```

Output:



```
PS D:\go\sample\demo> go run comb.go

--- Unsorted ---

 [319 -13 -289 545 712 485 -155 121 -212 603 -61 415 230 724 -455 -330 -618 -197 -464 -319]

--- Sorted ---

 [-618 -464 -455 -330 -319 -289 -212 -197 -155 -61 -13 121 230 319 415 485 545 603 712 724]
PS D:\go\sample\demo>
```