Вариант 3

Лабораторная работа 4

package main

import (

"fmt"

"math"

)

const V = 3

const rad = 0.001

type Interval struct {

l, r float64

}

func (i Interval) Add(other Interval) Interval {

return Interval{i.l + other.l, i.r + other.r}

}

// Возвращает наибольшее значениее из 2-ух интервалов

func Comparison(X, Y Interval) Interval {

var res Interval

if X.r < Y.l {

return Y

} else if X.l > Y.r {

return X

}

return res

}

func makeArr(N int) []Interval {

arr := make([]Interval, N)

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

arr[i].l = math.Sin(float64(V+i)) - rad

arr[i].r = math.Sin(float64(V+i)) + rad

}

return arr

}

func displayIntervals(arr []Interval) {

fmt.Println("\nИнтервалы:")

for i := 1; i < len(arr); i++ {

fmt.Printf("[%15.5f, %15.5f]\n", arr[i].l, arr[i].r)

}

fmt.Println()

}

func calculateSum(arr []Interval) Interval {

var sum Interval

for \_, interval := range arr {

sum = sum.Add(interval)

}

return sum

}

func sortSlice(arr []Interval) {

for i := 1; i < len(arr)-1; i++ {

for j := i + 1; j < len(arr); j++ {

if Comparison(arr[i], arr[j]) == arr[i] {

arr[i], arr[j] = arr[j], arr[i]

}

}

}

}

func main() {

var N int

fmt.Print("Введите N: ")

fmt.Scan(&N)

arr := makeArr(N + 1)

// Вычисляем сумму до сортировки

sumBeforeSorting := calculateSum(arr)

fmt.Println("Исходный массив")

displayIntervals(arr)

// Сортируем интервалы по полю l (по левому краю каждого интервала)

// sort.Slice(arr, func(i, j int) bool {

// return arr[i].l < arr[j].l

// })

sortSlice(arr)

fmt.Println("Сортированные интервалы по полю l (по левому краю каждого интервала):")

displayIntervals(arr)

// Вычисляем сумму после сортировки

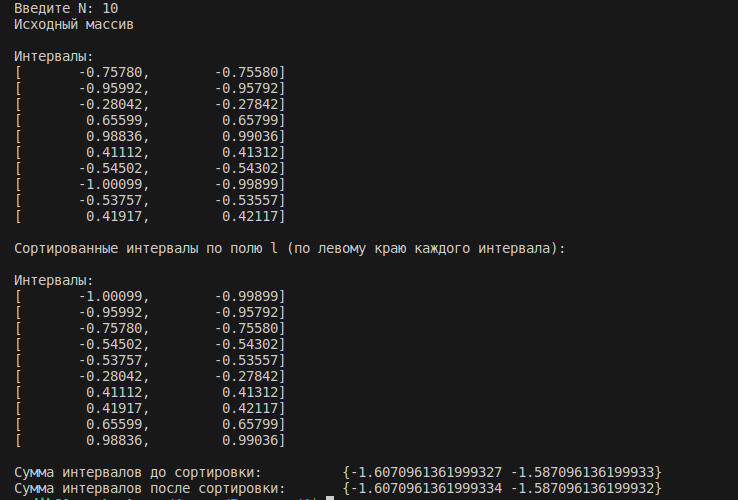
sumAfterSorting := calculateSum(arr)

fmt.Println("Сумма интервалов до сортировки: \t", sumBeforeSorting)

fmt.Println("Сумма интервалов после сортировки: \t", sumAfterSorting)

// Нет необходимости освобождать память в Go, сборка мусора заботится об этом

}



Лабораторная работа 5

