

# Data Structures, Algorithms, and Functions

in Go

Anubhav Saini  
2021-03-09

# Table of Content

Table of Content	<b>2</b>
Sum of Pair	<b>3</b>
Local peak finding	<b>5</b>
Sorting	<b>6</b>
Binary search	6
Bubble sort	7
Selection sort	8
Insertion sort	9
Merge sort	10

# Sum of Pair

## Problem statement

Find two numbers in an array such that their sum equals the given sum.

```
type Pair struct {
    First int
    Second int
}

func sumofpair(array []int, given int) (out Pair) {
    out = Pair{First: -1, Second: -1}
    for i := 0; i < len(array); i += 1 {
        for j := 0; j < len(array); j += 1 {
            if i == j {
                continue
            }
            if array[i] + array[j] == given {
                out.First, out.Second = i, j
                return out
            }
        }
    }
    return out
}
```

## Problem statement

Find three numbers in an array such that their sum equals the given sum

```
type PairOf3 struct {
    Pair
    Third int
}

func (p PairOf3) String() string {
    return fmt.Sprintf("%d, %d, %d", p.First, p.Second, p.Third)
}

func sumofpair3 (array []int, given int) (out PairOf3) {
    out = PairOf3{
        Pair: Pair{-1, -1},
        Third: -1,
    }
    l := len(array)
    for i := 0; i < l; i++ {
        for j := 0; j < l; j++ {
            if i == j {
                continue
            }
            twos := array[i] + array[j]
            for k := 0; k < l; k++ {
                if i == k || j == k {
                    continue
                }
                sum := array[k] + twos
                if sum == given {
                    out.First, out.Second, out.Third = i, j, k
                    return out
                }
            }
        }
    }
    return out
}
```

# Local peak finding

## Problem statement

Find a peak in an array of numbers. A peak is a number that is greater than or equal to its neighbours. Elements at the array boundaries cannot be peaks.

```
type LocalPeak struct {
    Index int
    Value int
}

func localPeakFinding(array []int) []LocalPeak {
    var peaks []LocalPeak
    for i := 1; i < len(array)-1; i++ {
        if array[i] > array[i-1] && array[i] > array[i+1] {
            peaks = append(peaks, LocalPeak{Index: i, Value: array[i]})
        }
    }
    return peaks
}
```

```
func main() {
    input := []int{13, 1, 18, 11, 6, 17, 23, 11, 22, 1}
    fmt.Println(input, localPeakFinding(input))
}
```

```
[13 1 18 11 6 17 23 11 22 1] [{2 18} {6 23} {8 22}]
```

This solution deviates from JavaScript one; here we are finding multiple peaks.

# Sorting

## Binary search

### Problem Statement

Given a sorted array of non-decreasing numbers. Find the given number. Use loops. Return -1, 0 if not found.

```
func binarySearch(array []int, given int) (index int, value int) {
    low, high := 0, len(array)
    for low < high {
        index = (low + high) / 2
        value = array[index]
        if value == given {
            return index, given
        } else if value > given {
            high = index
        } else {
            low = index + 1
        }
    }
    return -1, 0
}
```

```
fmt.Println(binarySearch([]int{1, 2, 3, 4, 5, 5, 5, 6, 7, 8, 8, 9}, 8))
fmt.Println(binarySearch([]int{1, 2, 3, 4, 5, 5, 5, 6, 7, 8, 8, 9}, 80))
9 8
-1 0
```

```
func binarySearchR(array []int, given, low, high int) (index, value int) {
    if low >= high {
        return -1, 0
    }
    mid := (low + high) / 2
    if array[mid] == given {
        return mid, given
    } else if array[mid] > given {
        high = mid
    } else {
        low = mid + 1
    }
    return binarySearchR(array, given, low, high)
}
```

```
fmt.Println(binarySearchR([]int{1, 2, 3, 4, 5, 5, 5, 6, 7, 8, 8, 9}, 8, 0,
11))
fmt.Println(binarySearchR([]int{1, 2, 3, 4, 5, 5, 5, 6, 7, 8, 8, 9}, 80, 0,
11))
10 8
-1 0
```

# Bubble sort

## Problem statement

Ascending sort an array of numbers using the bubble sort algorithm.

```
func bubbleSort(array *[]int) {  
    a := *array  
    for i := 0; i < len(a); i++ {  
        for j := 0; j < len(a); j++ {  
            if a[j] > a[i] {  
                a[i], a[j] = a[j], a[i]  
            }  
        }  
    }  
}
```

# Selection sort

## Theory

Select the minimum element, and put it in its place. Repeat for the rest of the array.

Alternatively, you can select the maximum element and put it in its place.

```
func selectionSort(array *[]int) {  
    a := *array  
    for i := 0; i < len(a); i++ {  
        min := i  
        for j := i+1; j < len(a); j++ {  
            if a[min] > a[j] {  
                min = j  
            }  
        }  
        a[i], a[min] = a[min], a[i]  
    }  
}
```



# Insertion sort

## Theory

Pick one element; put it in its place relative to the already sorted array.

```
func insertionSort(array *[]int) {  
    a := *array  
    for i := 1; i < len(a); i++ {  
        for j := i - 1; j >= 0; j-- {  
            if a[j] > a[j+1] {  
                a[j], a[j+1] = a[j+1], a[j]  
            }  
        }  
    }  
}
```

## Merge sort

```
func merge(array *[]int, low, mid, high int) {
    a := *array
    left, right := low, mid+1
    if a[mid] <= a[right] {
        return
    }
    for left <= mid && right <= high {
        if a[left] >= a[right] {
            value := a[right]
            for index := right; index > left; index-- {
                a[index] = a[index-1]
            }
            a[left] = value
            right++
        }
        left++
    }
}

func mergeSort(array *[]int, low, high int) {
    if low < high {
        mid := (low + high) / 2
        mergeSort(array, low, mid)
        mergeSort(array, mid+1, high)
        merge(array, low, mid, high)
    }
}
```

