DSA WORKOUT

**DSA-ARRAY**

**Array Sample Workouts**

**Sample WorkOut**

**Q.1 Sort the Array**

//Sort the array with loop ascending order / descending order

func sortArray(arr[]int){

    for i:=0;i<len(arr);i++{

        for j:=i+1;j<len(arr);j++{

            if arr[i]>arr[j]{                 //for descending order if arr[i]<arr[j] ---this condition need to given

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

            }

        }

    }

}

//Time Complexity = O(n^2)--- Quadratic Time Complexity

//Space Complexity = O(1) --- Constant Time Complexity

func main(){

    arr := []int{24,15,15,22,10,33,44,55}

    sortArray(arr)

    fmt.Println("Result of Sorted Array",arr)

}

**Q2. Move to last the give targeted element to the array**

//Move to last the target

func movetoEnd(nums []int, target int)  {

    var i = 0

    var j = len(nums) - 1

    // Move all non target elements to the front

    for i < j {

        if nums[i] != target && nums[j] == target {

            i++

            j--

            continue

        }

        nums[i],nums[j] = nums[j],nums[i]

        i++

        j--

    }

    // move all target element to the end

    j = len(nums)-1

    for i < j {

        if nums[i] == target{

            nums[i],nums[j] = nums[j],nums[i]

            j--

        }else{

            i++

        }

    }

}

func main() {

    arr := []int{1, 2, 3, 3, 5, 3, 5, 6, 3, 9}

    tar := 5

    movetoEnd(arr, tar)

    fmt.Println("After moving to the last ", arr)

}

//Time : = O(n)

//Space := O(1)

**Leetcode**

**A.0 1 Two Sum**

func twoSum(nums []int, target int) []int {

    //Hash table method to try the solution

    seen := make(map[int]int)

    for i,num := range nums{

        diff := target-num

        if index,ok := seen[diff];ok{

            return []int{index,i}

        }

        seen[num]=i

    }

    return []int{}

}

Time Complexcity = o(n)- Linear

Space Complexcity = o(1)- Constant

**A.1 2824 Count Pairs Whose Sum is Less Than Target**

func countPairs(nums []int, target int) int {

        count :=0

        for i:=0; i<len(nums);i++{

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

                if nums[i] + nums[j] < target{

                    count++

                }

            }

        }

        return count

}

Time Complexcity = o(n^2)

Space Complexcity = o(1)

**A3. 121 Best Time to Buy and Sell Stock**

func maxProfit(prices []int) int {

    profit := 0

    var minprofit = prices[0]

    for i := 0; i < len(prices); i++ {

        if prices[i] < minprofit {

            minprofit = prices[i]

        } else if (prices[i] - minprofit) > profit {

            profit = prices[i] - minprofit

        }

    }

    return profit

}

Time = O(n)

Space = O(1)

**DSA-LinkedList**

1.Prepend

//Prepend the linked list

func (l \*linkeList) prepend(n \*node) {

    secondnode := l.head

    l.head = n

    l.head.next = secondnode

    l.length++

    // for secondnode != nil {

    //  fmt.Printf("%d",secondnode.data)

    //  secondnode = secondnode.next

    //  l.length--

    // }

    // fmt.Println("\n")

}

2.Add

//Add to node

func (link \*LinkedList)Add(name string){

    newNode := &Node{name: name}

     if link.head == nil{

        link.head=newNode

     }else{

        current := link.head

        for current.next != nil{

            current = current.next

        }

        current.next = newNode

     }

}

3.Display

//Display the link

func (link \*LinkedList)Dispay(){

    current := link.head

    fmt.Println("Student list")

    for current != nil{

        fmt.Println(current.name,"")

        current = current.next

    }

    fmt.Println()

}

4.Remove

func (link \*LinkedList)Remove(name string){

    if link.head == nil{

        return

    }

    if link.head.name == name{

        link.head = link.head.next

    }

    current := link.head

    for current.next != nil && current.next.name != name{

        current = current.next

    }

    if current.next != nil{

        current.next = current.next.next

    }

}

Leetcode

Q1 203 Remove Linkedlist Elements

func removeElements(head \*ListNode, val int) \*ListNode {

     removeEle := &ListNode{Next :head}

     dumRemove := removeEle

     for dumRemove.Next != nil {

        if dumRemove.Next.Val == val{

            dumRemove.Next = dumRemove.Next.Next

        }else{

            dumRemove = dumRemove.Next

        }

     }

     return removeEle.Next

}

206.Reverse Linked List

/\*\*

 \* Definition for singly-linked list.

 \* type ListNode struct {

 \*     Val int

 \*     Next \*ListNode

 \* }

 \*/

func reverseList(head \*ListNode) \*ListNode {

    var previous \*ListNode

     current := head

    for current != nil {

        temp := current.Next

        current.Next = previous

        previous = current

        current = temp

    }

    return previous

}

**DSA – Strings**

Sample Workout

Q.1 – Replace each alphabet in the given string with another alphabet occurring at the n- the position from each of them.

//Enode the replace each alphabet in the given string with another alphabet occurring  at the n-th position from each of them.

func Encode(s string, n int)string{

    result := ""

    change := rune(n%26)

    for \_,char := range s{

        if char <= 'z'-change{

            result = result+string(char+change)

        }else{

            result = result+string(char-26+change)

        }

    }

    return result

}

func main() {

    //Encode

    value := "hai"

    key := 2

    fmt.Println("After Adding the alapha",Encode(value,key))

    //Strings in the Golang - strings contains many charcter

Q2 . Reverse String

//Reverese String

func ReverseString(s string)string{

     result := ""

     for i:=len(s)-1;i>=0;i--{

        result += string(s[i])

     }

     return result

}

Q3.Pallindrom String

//Pallindrome String

func IsPlalindrome (s string)bool{

    for i := 0; i<(len(s)+1)/2; i++{ //O(n)

        if s[i] != s[len(s)-i-1]{

            return false

        }

    }

    return true

}  
//O(n) T O(1)s

Q4. Counting the vowels in word

//checking the vowels

func IsvowelCount(s byte) bool{

    if s =='a'|| s == 'e'|| s == 'i'|| s == 'o'|| s == 'u'{

        return true

    }

    return false

}

//Count Vowels

func CountVowels(w string)int{

    count := 0

    for i :=0 ; i<len(w); i++{

        if IsvowelCount(w[i]){

            count++

        }

    }

    return count

}

DSA –Binary Search

Q1. Binary Search

Q1. //Binary search

func binarySearch(nums []int , val int)bool{

    low, high := 0, len(nums)-1

    for low <= high{

        mid := (low+high)/2

        if  val < nums[mid] {

            low = mid+1

        }else if val > nums[mid]{

            high = mid-1

        }else{

            return true

        }

    }

    return false

}

Q3. First Occurrence of Target

//First Occurance

func firstOccurance(nums []int , target int )int{

    low,high := 0,len(nums)-1

    firstOccur := -1

    for low <= high{

        mid := (low+high)/2

        if nums[mid] == target{

            firstOccur = mid

            high = mid-1

        }else if nums[mid]<target{

            low = mid+1

        }else{

            high = mid-1

        }

    }

    return firstOccur

}

Q3.Last Occurrence

//Last Occurrence

func lastOccurance(nums []int , val int )int{

    low,high := 0,len(nums)-1

    lastOccur := -1

    for low <= high{

        mid := (low+high)/2

        if nums[mid]== val{

            lastOccur = mid

            low = mid+1

        } else if val < nums[mid]{

            high = mid-1

        }else{

            low = mid+1

        }

    }

    return lastOccur

}

DSA –Linear Search

Q1. LinearSearch

//Linear Search

func LinearSearch(nums []int, target int) bool {

    for i := 0; i < len(nums); i++ {

        if nums[i] == target {

            return true

        }

    }

    return false

}

Q2. Linear Search in range loop

func LinearSearc(nums []int,target int)int{ // Time complexcity O(n), O(1) is space complexcity

    for i,item := range nums{

        if item == target{

            return i+1

        }

    }

    return 0

}