Data Structure Workouts

1. Learn what is Data Structure & Algorithms.
2. Learn the basics of Memory Allocation and Memory leak.
3. Learn the concept of Complexity Analysis.
   NB: The complexity of common operations of all data structures should be covered.
4. Learn about Asymptotic analysis (Big-O notation).
5. Learn the concepts of Array. Complete at least three sample workouts & do at least 3 problems from any competitive coding websites (Hacker Rank, Code Chef, Leet code, Algo Expert, etc.)
6. Learn the concepts of the Linked list. Complete at least three sample workouts
   a. Construction of Singly linked list & Doubly linked list.
   b. Convert array to a linked list
   c. Add a node at the end & beginning
   d. Delete node with the value specified
   e. Insert a node after & before a node with x data
   f. Print all elements by order & reverse by order
   g. Write a program to remove duplicates in a sorted singly linked list
7. Learn the concepts of String. Complete at least three sample workouts.
   Eg: Write a function to replace each alphabet in the given string with another alphabet occurring at the n-th position from each of them.
8. Learn about Linear Search & Binary Search. Complete at least 3 sample workouts in each of them
9. Learn the concepts of Recursion. Complete at least 3 sample workouts.
10. Learn about the applications of all structures you covered this week

Write a short description about this task

A Data structure is a way to store and organise data so it can be used efficiently .It is the way of arranging data on a computer so that it can be accessed and updated efficiently.

In Dom we are using Tree Data structure

In Browsers back and forward buttons - Stack ds

In os job scheduling -queue ds

Algorithms : Algorithms is a set of well defined instruction to solve a particular problem

An algorithm must have

1. *Well defined inputs and output*
2. *Each step should be clear and unambiguous*
3. *An algorithm must be language independent*

We evaluate the performance of an algorithm in terms of its input size

Write a short description about this task

Basics of memory allocation and memory leak

Memory allocation is the process of setting a portion computer's memory for a specified purpose such as storing data or executing code

Memory allocation refers to the act of reserving memory space during the execution of program.This space can be used to store data such as variables ,arrays

There are two types of memory allocation

static memory allocation and Dynamic memory allocation

Static memory allocation refers to the memory allocation that takes place during the compile time so memory allocated is fixed and we cannot increase or decrease during runtime

Dynamic memory Allocation refers to memory allocation at the time of execution of program

Heap is a segment of memory where dynamic memory allocation takes place

Unlike stack where memory is allocated or deallocated in a defined order Heap is an area of memory where memory is allocated or deallocated without any order or randomly

Pointers play an important role in dynamic memory allocation

Allocated memory can be accessed through pointers

Memory leaks occur when a program allocate memory but fails to release it when it is no longer needed This can happen when a program has an error or bug that prevents it from properly releasing memory or when a program allocates memory but does not have a way to reallocate it,over time memory leaks can cause program to leak more and more memory which can eventually leads crashes or other errors.To prevent memory leaks programmers should always make sure to release memory that is no longer needed

Write a short description about this task

Concept of complexity analysis

Complexity analysis also known as algorithm analysis .it is a way of evaluating the efficiency and performance of the algorithm it involves measuring how the memory and running time of an algorithm increases as the size of input data increases

There are two types of complexity analysis

1.Time complexity

2. Space complexity

Time complexity :it is the amount of time takes to run as a function of length of input .The length of input determines how many operations the algorithm will do

It will provide information about variance in execution time as no input increases or decreases.

2.Space complexity:

Amount of memory taken by an algorithm to run as a function of input size

Types of time complexity

1.constant:When an algorithm is not reliant on its input size n it is said to have constant time complexity with order o(1)

2.linear:when the running time of algorithm rises linearly with the length of the input it is said to have linear time complexity,when a function checks all the values of input data set it is said to have time complexity of order o(n)

3.quadratic:when execution time of an algorithm rises non linearly n2 with the length of input it is said to have quadratic time complexity

*In general nested loop falls in quadratic time complexity where one loop take o(n) and the inner for loop also takes o(n) ie o(n)*o(n)=o(n2)*

4.Logarithmic:when an algorithm lowers the amount of input data in each step .it is said to have logarithmic time complexity Binary trees and Binary search functions are some of algorithms with logarithmic time complexity

Write a short description about this task

Asymptotic Notations: it is a way to describe the behaviour of an algorithm as the length of the input grows to infinity

Big-o notations

In theoretical terms Big O notation is used to examine algorithms performance or complexity

Big o notations: it is the maximum amount of time or space that the algorithm will take to run as the input size reaches infinity (worst case)

Big Omega Notation: it gives the minimum amount of time or space that the algorithm will take to run the input size reaches infinity(best case)

Big theta notation: it gives the estimate of the algorithms running time and space that is both upper and lower bound as the input reaches infinity(Average case)

Write a short description about this task

Link to the folder containing code and screenshot of the output

Array is a Data Structure that can hold a collection of values

Arrays can contain a mix of different Data Types such as string boolean numbers or objects all in same array.Arrays are resizable we don't have to declare the size of array before creating it

Java script arrays are zero indexed and the insertion order is maintained

Arrays are iterables and can be used with for loop

Push pop shift unshift map filter reduce concat for of etc are some of array methods

Array

Big o Time complexity: insert or remove from end :o(1)

Insert or remove from beginning o(n)

Accessing an element is constant complexity o(1) because fetching the first element is no different from from any element from the array

Searching for an element has linear time complexity ie o(n) because the search element can be the last element of the array

Push and pop are constant o(1)

Shift, unshift ,slice ,splice, and concat are linear time complexity 0(n)

forEach ,map ,filter, reduce are of o(n) linear complexity

Write a short description about this task

Link to the folder containing code and screenshot of the output

Linked List:

Linked list is a linear data structure that includes a series of connected nodes

Each node consist of a data value and a pointer that points to the next node

```
class Node{
    constructor(value)
    {
        this.value=value
        this.next=null
    }
}
class Linkedlist{
    constructor(){
        this.head=null
        this.tail=null
        this.size=0
    }
    addFirst(value)
    {   const node=new Node(value)
        if(!this.head)
        {
            this.head=node;
            this.tail=node
            return;
        } node.next = this.head;
        this.head = node;
        this.size++

    }
    addEnd(value)
    {const node=new Node(value)
        if(!this.head)
        {
            this.head=node
            this.tail=null
        }
```

```
            this.tail.next=node
            this.tail=node
            this.size++
            return
        }




        insert(value,index)
        {
            if(index==0)
            {
                this.addFirst(value)
                return
            }
            const node=new Node(value)
            let current=this.head
            let i=0
            while(current)
            {
                if(i==index-1)
                {
                    node.next=current.next
                    current.next=node
                    if(node.next==null)
                    {
                        this.tail=node
                    } this.size++
                    return

                }current=current.next
                i++
            }


        }

        remove(value)
        {
            if(!this.head)
```

```
            {
                return
            }
            if(this.head.value==value)
            {
                this.head= this.head.next
                if(this.head==null)
                {
                    this.tail=null
                }
                this.size--
                return
            }
            let current=this.head
            while(current.next)
            {
                if(current.next.value==value)
                {
                    current.next=current.next.next
                    if(current.next==null)
                    {
                        this.tail=current
                    }
                    this.size--
                    return
                }current=current.next
            }
        }
    print()
    {
        const data=[]
        let current=this.head
        while (current)
        {
            data.push(current.value)
            current=current.next
        }
        console.log(data.join("->"))
    }
}
const list=new Linkedlist()
list.addFirst(100)
```

```
list.addFirst(500)
list.addFirst(10)
list.addEnd(333)
list.insert(442,1)
list.insert(444,2)
list.print()
list.remove(442)
list.print()
```

Write a short description about this task

Link to the folder containing code and screenshot of the output

String

String is defined as a Array of characters .In javascript string is a sequence of characters with no special character requested to make the end of the string

Javascript uses dynamic memory allocation t o store strings .IE memory is allocated at run time based on the length of the string .

Javascript strings are immutable which means once a string it cannot be changed .if you need to modify a string you must create a new string .This means that when you create a new string by modifying an existing string the run time must allocate new memory to new strings

```
function palindrome(string)
{
    for(i=0;i<string.length;i++)
    {
        if(string[0]==string[string.length-1])
        {
            return `${string} is palindrome`
        }

    else {
        return  `${string} is not palindrome`
    }
}
}
const string="haiii"
const result=palindrome(string)
console.log(result)
```

```
function frequency(string)
{
    let count=[]
    for(i=0;i<string.length;i++)
    {
        let char=string[i]
        if(count[char])
        {
            count[char]++
        }
        else
        {
            count[char]=1
        }



    }
    return count
}

const string="wonderland"
 const result=frequency(string)
console.log(result)
```

```
function stringreverse(string)
{
    let newstr=""
    for(let i=string.length-1;i>=0;i--)
    {
        newstr+= string[i]
    }
    return newstr
}
const string="Hello i am learning js with DSA"
const result=stringreverse(string)
console.log(result)
const string="The quick brown fox jumps over the lazy dog"
```

```
const substring="brown"
if(string.includes(substring))
{
    console.log(`${string} contains the substring ${substring}`)
}
else
{
    console.log(`${string} does not contain the substring`)
}
```

Write a short description about this task

Link to the folder containing code and screenshot of the output

Linear search and Binary search:

Linear search: is a searching algorithm that sequentially checks each element in a list or an array until a match is found or the entire list has been searches it is also known as sequential search

The algorithm starts at the beginning of the list and compares each element with the target value if the element matches the target the search ends and the index of the element is returned,if the end of the list is reached without finding a match the search ends and -1 is returned

Linear search is a simple and easy to implement algorithm,but it has a time complexity of o(n) where n is the number of elements in the list .This means that as the size of the list grows the time taken to search the list also increases .There fore for large datasets more efficient search algorithms like binary search or hash tables may be appropriate

```
function linearSearch(arr,target)
{
    for(let i=0;i<arr.length;i++)
    {
        if(arr[i]===target)
        {
            return i
        }

      }       return -1
```

```
}
const arr=[2,3,4,5,6,7,8,9]
const result=linearSearch(arr,7)
console.log(result)
```

Binary Search:

Binary search is a searching algorithm that is used to search for an element in a sorted list or an array ,it works by repeatedly dividing the search interval in half until the target List or an array. It works by repeatedly dividing the search interval in half until the target element is found ot the search interval is empty
The algorithm starts by comparing the target element with the middle element of the list if the target element matches the middle element the search ends and the index of the middle element is returned .if the target element is less than the middle element the search continues in the lower half of the list .if the target element is greater thwan the middle element the search continues in the lower half of the list if the target element is greater than the middle element the search continues in the upper half of the list

The process is repeated until the target element is found or the search interval is empty. If the search interval is empty and the target element is not found, -1 is returned.

Binary search has a time complexity of O(log n), where n is the number of elements in the list. This means that as the size of the list grows, the time taken to search the list increases much more slowly than linear search. Therefore, binary search is a more efficient algorithm for searching large sorted datasets.

```
function binarySearch(arr,target)
{
    let left=0;
    let right=arr.length-1
    while (left<=right)
    {
```

```
        let mid=Math.floor((left+right)/2)
        if(arr[mid]==target)
        {
            return mid
        }
        else if(arr[mid]<target)
        {
            left=mid+1
        }
        else
        {
            right=mid-1
        }
    }
    return -1
}
const arr=[2,5,7,9,11,44]
const result=binarySearch(arr,2)
console.log(result)
```

Binary search using recursion

```
function binarysearchRecursion(arr,target,start=0,end=arr.length-1)
{
    if (start>end)
    {
        return -1
    }
    const mid=Math.floor((start+end)/2)
    if(arr[mid]==target)
    {
        return mid;
    }
    else if(arr[mid]>target)
    {
        return binarysearchRecursion(arr,target,start,mid-1)
    }
    else
    {
        return binarysearchRecursion(arr,target,mid+1,end)
    }
```

```
}

const arr=[2,3,4,5,6,7,8,9,10]
const result=binarysearchRecursion(arr,8,start=0,end=arr.length-1)
console.log(result)
```

Write a short description about this task

Link to the folder containing code and screenshot of the output

Recursion:

The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called recursive function.A recursive function must have a condition to stop calling itself otherwise the function is called indefinitely

Once the condition is met the function stops calling itself ,this condition is called base condition

Direct Recursion: occurs when a function calls itself with in its own body

Indirect Recursion: occurs when a function calls another function which that eventually calls the original function

Tailed Recursion:if the recursive function is the last operation of the function .This means that there is no additional computation to be done after the recursive call and the compiler or interpreter can optimise the recursion by replacing the current stack frame with new one rather than creating a new stackframe for each recursive call .This optimization is called tail call optimization

Non tail recursion: if the recursive call is not the last operation of function .This means there is an additional computation to be done after the recursive call and the compiler or interpreter cannot optimise recursion call using tail call optimisation

1.Factorial program using recursion

Function factorial(n)

{ if(n==0)

{

return 1

}

else

```
{
return n* factorial(n-1)
}
}
Const result=factorial(5)
console.log(result)
```

2. Print the elements of an array using recursion

```
Function PrintArray(ar,index)
{
If (index==arr.length)
{return
}
Else
{
console.log(arr[index]
}
printArray(arr,index+1)
}
Const arr=[1,2,3,4,5,6]
print array(myArray,0)
```

3.Fibinocci series

```
function fibonacci(n) {
  if (n <= 1) {
    return n;
  } else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}
console.log(fibonacci(5)
```

Write a short description about this task

1.Applications of linked list

Linked lists are a fundamental data structure in computer science and have numerous applications in various fields. Here are some common applications of linked lists:

1. *Implementing data structures: Linked lists are used to implement various data structures such as stacks, queues, and hash tables.*
2. *Dynamic memory allocation: Linked lists are useful in dynamic memory allocation where memory is allocated at runtime. The linked list can be used to store memory blocks of different sizes.*
3. *File processing: Linked lists are used in file processing to keep track of the contents of the file. Each node in the linked list contains a piece of data, and the linked list can be used to traverse the entire file.*
4. *Graph algorithms: Linked lists are used to represent graphs, where each node in the linked list represents a vertex in the graph. The linked list can be used to traverse the graph and perform various graph algorithms such as depth-first search and breadth-first search.*
5. *Music player applications: Linked lists are used to implement the playlist feature in music player applications. Each node in the linked list contains a song, and the linked list can be used to play songs in a particular order.*
6. *Navigation applications: Linked lists are used in navigation applications to store information about routes and locations. Each node in the linked list contains information about a particular location, and the linked list can be used to navigate through different locations.*
7. *Operating systems: Linked lists are used in operating systems to maintain a list of processes and their states. Each node in the linked list represents a process, and the linked list can be used to schedule processes and allocate resources.*

Doubly Linked List Applications:

1. *Implementation of stack and queue data structures*
2. *Browsing history in web browsers*
3. *Undo/Redo functionality in text editors*

4. *Music player playlists*
5. *Navigation systems for cars and other vehicles*
6. *Binary trees can be implemented using a doubly linked list as well.*

Circular Linked List Applications:

1. *Implementation of circular buffers in embedded systems and communication systems*
2. *Round-robin scheduling algorithm in operating systems*
3. *Implementing a ring topology in computer networks*
4. *Implementing a clock in real-time systems*
5. *Music players playlists*
6. *Simulations of processes that repeat themselves over time, like a seasonal pattern or a repetitive manufacturing process.*