Data Structures & Algorithm
============================

session 1 - Introduction
==========================

1. what is a data structure?

A way to organize data

Arrays, LinkedList, Trees, Stack, Queue, Hash Table

Array

A collection of elements with similar data type

Array of Integers [101, 103, 105, 200, 250]

2. what is an Algorithm -

[1,4,6,8,10,12,15] - 7 elements

Searching
============
linear search
Binary search

Sorting
=========
Quick sort
Merge sort

Algorithm is a way to solve a particular problem.

3. why companies ask DSA in interviews.

Any top product based company ask you data structure..

related to logic..

4. how important is DSA for data engineers..

it is really important for top product based companies.

5. to what level we should prepare DSA as a Data Engineer

Easy to Medium level of questions..

==============

p1
===
Arrays
Linked List - singly linkedList
Hash Table
Strings
Sorting - Quick sort, Merge sort
Searching - Linear search, Binary search
Hashing
Time Complexity, Space Complexity

p2
===
DP

Trees - BST, Tree traversals

p3
===
Heap - Min & Max heap

Stack

Queue

Session 2 - Time Complexity
============================

Arrays..

100 elements in the array

```
a = [1, 3, 5, 8, 10 , 12, 15 ......    1000]
     0  1  2  3  4    5    6           99
```

a[6]

what is the element that is stored at 3 index

how many operations the system has to perform to get an element at a particular index..

sequential access or random access?

random access

1 operations

head 1 -> 3 -> 5 -> 8 -> 10 -> 12 ->15 ..... -> 1000

multiple operations are required (sequencial access)

[1, 3, 5, 8, 10 , 12, 15, 30 ,57, 63 , 90 ......   1000]

x elements if time taken is y

2x elements time taken would be 2y

whether 57 is present in this array or not

linear search -

best case in terms of time - 1 operation would be required

worst case is when the element is not present and we have to search with all the elements.. - 100 operations

we denote the worst case with big O notation
if the size of array is n

O(n)

Time complexity - we talk about number of operations

so our intent is to find how the time grows when data grows..

1024 elements

[1, 3, 5, 8, 10 , 12, 15, 30 ,57, 63 , 90 ......   1000]

binary search when array is sorted.

63 , 90] - 11 elements
3   4

63 is present or not.

the middle most element is (0+10)/2 = 5th index

is 63 > 12 or not

1024 - array

512 - 1

256 - 2

128 - 3

64 - 4

32 - 5

16 - 6

8 - 7

4 - 8

2 - 9

1 - 10

you have a sorted array of 1024 elements and you have to search for a particular number in this array...

as per linear search - O(n) 1024

binary search - O(log n) 10

1000000 elements in an array

if you have to find for a number in this array using linear search what is the time complexity...

O(n) - 1000000

20 operations

1
2
4
8
16
32
64
128
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576

O(log n)

O(log n) < O(n)

16 elements in array

O(1) < O(log n) < O(n) < O(nlogn) < O(n2) < O(n3) < O(2n) < O(n!)

O(1) - 1

O(log n) - 4
O(n) - 16
O(nlogn) - 64 (quick sort, merge sort)
=====================
O(n2) - 256
O(n3) - 4096
O(2n) - 65536
0(n!) - very very very big number

1000

O(n2) + O(n) ~ O(n2)

1000000 + 1000

Space complexity - sometimes when we give extra space (Auxillary space) then we are able to reduce the time complexity

in interviews they will give you a problem..

brute force... O(n2)

O(nlogn)

O(n)

Session 3 - Arrays
====================

1. element present at a particular index

1000 elements in the below array

[1,5,2,8,9,10,4]

element present on 500th index..

O(1)

2. array of 1000 elements, you have to tell whether there are duplicates in the array or not.

[1,5,3,8,9,10,4,3,15,18,20]
.
here we know that 3 is repeating 2 times...

=> brute force solution - n + (n-1) + (n-2) + (n-3)

5 + 4 + 3 + 2 + 1 = 15

(n * n-1)/2

n2
--   ~ n2
2

3*n ~ n

O(n2) + O(n) ~ O(n2)

O(3n) ~ O(n)

O(1) - constant time

O(log n) - logarithmic time

O(n) - linear time

O(n2) - quadratic time

O(n3) - cubic time

O(2n) - exponential time

[1,5,3,8,9,10,4,3,15,18,20]

if this particular array is sorted

[1,3,3,4,5,.........,18,20] assuming this is the array

n elements I will do n comparisons in the worst case

a single pass through the array

O(n)

input - [1,5,3,8,9,10,4,3,15,18,20]
output - [1,3,3,4,5,.........,18,20]

sorting - quick sort, merge sort
O(nlogn)

1024 elements in the array

1024 * 10 = 10240 operations are required to sort the array

Total time complexity = O(nlogn) + O(n) ~ O(nlogn)

1000

bruteforce = O(n2) = 1000000
optimized sol = O(nlogn) = 10240

at the expense of some extra/auxillary space we could even reduce the time complexity further...

Session 4 - Arrays
===================

1. you are given an array of n elements & you have to find the maximum element in the array.

[1,5,3,8,9,10,4,3,15,18,20,12,19]

brute force - sort the array O(nlogn)

[1,3,3,5,9,10..... 19,20]

just return the last element in the array and that should be your answer..

O(1) to get the last element

O(nlogn) + O(1) ~ O(nlogn)

[1,5,3,8,9,10,4,3,15,18,20,12,19]

=> max = 20

O(n)

2. There is an array of 1000 elements, find the element which repeat maximum number of times..

[1,2,1,2,2,3,2,6,1,8,9]

answer should be 2

=> brute force - O(n2)

num_max_frequency = 2
count = 4

=> [1,1,1,2,2,2,2,3,6,8,9]
    . . . .
O(nlogn) + O(n) ~ O(nlogn)

3. Consider you have an array of 999 elements.

1-1000, but one number is missing and we need to find the missing number in the array...

[101,205,23...... 900] - 999

=> brute force - first check for 1 in the array
check for element 2
check for element 3...

1000 * 999 checks.. O(n2)

=> we can sort the array

O(nlogn)

[1,2,3,4,5,...... 849,851,852.....1000]

in a single pass O(n) you will be able to tell the missing number.

O(nlogn) + O(n) ~ O(nlogn)

=> how will you represent number 13 in binary representation

```
    8 4 2 1
    1 1 0 1
```

the binary rep of 13 is 1101

   21

```
16 8 4 2 1
1  0 1 0 1
```

the binary rep of 21 is 10101

AND, OR, XOR

AND -
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

OR -
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

XOR -

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

21 XOR 21

```
10101 -
10101 -
========
00000
========
```

=> Property of XOR operation is that when we XOR a number with itself it results in 0

=> when we XOR a number with 0 we get the same number..

 21 XOR 21 XOR 21

0 XOR 21 = 21

```
10101
00000
========
10101
```

if you xor a number even number of time we get 0
if you xor it odd number of times you get the same number

=> [101,205,23...... 900]

we know the above array has 999 elements

1-1000

4 elements in the array

1-5 (one number is missing)

[2,4,5,1]

1 xor 2 xor 3 xor 4 xor 5 xor 2 xor 4 xor 5 xor 1

3

O(2*n) ~ O(n)

=> 1-5 (one number is missing)

[2,4,5,1]

we know the range of numbers is 1 to 5
so what is the sum of all number from 1 to 5
1+2+3+4+5 = 15

Sum of all numbers from 1 to n = n*(n+1)/2 in a single pass we would iterate over the array and can calculate the sum of all element in the array = 12

O(n)

15 - 12 = 3

10*11/2 = 55

1000 * 1001/2 = 500 * 1001 = 500500

sum of all number from 1 to 1000 is 500500

O(1)

if 850 is the missing element then sum of all elements in the array 499650

500500 - 499650 = 850

O(n)

to sum up the range of elements O(1)

to sum up all the elements in the array O(n)

Session 5 - Real Life application
====================================

consider you have build a contact list

Name
phone number

I have to implement such a contact list -
1. add
2. search

array contains all elements of same type

contact_type = name,phonenum

Array - []

unsorted array
[100,101,102,103,105,98,87]

generally arrays are of fixed size...

I can add a new phone number at the last position

unsorted array
================
add - O(1)
search - O(n)

sorted array
================

[100,101,102,104,105,109]

lets say I want to add 103

right now at index 3 we have 104

[99,100,101,102,103,104,105,109]

add - O(n)
searching - log(n)

[100,101,102,103,105,98,87]

Sorting - O(nlogn)

[99,100,101,102,103,104,105,109]

m times you are search for a phone number

```
O(nlogn) + O(mlogn)
--------------------
 m
```

so if the value of m is >=n then you should not feel the burden of preprocessing cost

```
O(nlogn) + O(nlogn)
--------------------   = O(logn)
 n
```

LinkedList
===========
in case of linkedlist the add operation will be O(1) as we can add a new element at the very starting of linkedlist

for searching in case of worst case we have to go through each node of the linkedlist O(n)

sorted array

add - O(n)
search - O(logn)

can we bring some extra space to reduce the time complexity..

lets say he have unsorted array and we have to sort the phone numbers O(nlogn)

can we achieve sorting in lesser time
also can we sort the elements without comparing with each other.

bin sorting
============

1000 - 2000 in an array

[1002,1001,1050,1049....]   O(nlogn)
-1000
2,1,50,49

[1001,1002...]

lets take one more array which can store 1000 elements

[,1001,1002]

1. all the elements should be integer
2. the elements should not be repeating
3. we should know the range of numbers

By just bringing the extra space my add and search operations run in O(1)

7000000000 - 9999999999
0 to 299999999

3 billion entries 50 GB of your memory

you might just store 200 contacts in your phone for that we have taken extra provision of 3 billion array elements...

[
7272727272

...

9090909090
]

hashing...

key should be
Sumit

1. what if the key is string and not numeric
2. what if the range of numbers is bigger

fixed space of 1000 elements(extra/auxillary space)

9100000007 % 1000 = 7 (I will keep 9100000007 in 7th index of the array)

9100007118 % 1000

in this case our hash function is mod 1000

by using some extra space we are able to reduce on the time complexity...

how to reduce collisions

1. use a good hash function which distributes the values evenly.

2. if you have 100 slots and you have to store 1000 contact numbers...

on average each index will have a linkedlist with roughly 10 elements...

sumit 9990009990

A - 65
a - 97

[] 100 elements

s - 116*1
u - 120*2
m - 100*3
i - 105*4
t - 110*5
=========
3456 % 100 = 56

timus - 25 index

t - 110*1
i - 105*2
m - 100*3
u - 120*4
s - 116*5
==========
3578 % 100 = 78

sumit 1
kapil 2
satish 5

0
1 sumit
2 kapil
3
4
5 satish

Array 1000

Hashing

add - O(1)

search - O(1)

Session 6 - Real Life application 2
=======================================

you are trying to create a new gmail account

trendytech.sumit@gmail.com

billions of gmail id's already created by people..

it is really impossible to store all the created gmail id's in memory..

we have to then go to the disk
2k to 3k times slower than memory..

Bloom filter

since it is a compressed kind of database we may get false positives...

this means that in case of satish@gmail.com

we know that no one has earlier created this email id...

but since all the 3 bits were set by other email id's

false positive

(1% of times...)

false negatives..

can we ever encounter a case when the system says that you can go ahead with this email id

a 1 mb of such database can help us to store 1000000 indexes..

but it is a probabilistic datastructures..

false positives are possible (1%)

false negatives is not possible