Property	Recursion	Iteration
Definition	Function calls itself.	A set of instructions repeatedly executed.
Application	For functions.	For loops.
Termination	Through base case, where there will be no function call.	When the termination condition for the iterator ceases to be satisfied.
Usage	Used when code size needs to be small, and time complexity is not an issue.	Used when time complexity needs to be balanced against an expanded code size.
Code Size	Smaller code size	Larger Code Size.
Time Complexity	Very high(generally exponential) time complexity.	Relatively lower time complexity(generally polynomial-logarithmic).
Space Complexity	The space complexity is higher than iterations.	Space complexity is lower.
Stack	Here the stack is used to store local variables when the function is called.	Stack is not used.

Property	Recursion	Iteration
Speed	Execution is slow since it has the overhead of maintaining and updating the stack.	Normally, it is faster than recursion as it doesn’t utilize the stack.
Memory	Recursion uses more memory as compared to iteration.	Iteration uses less memory as compared to recursion.
Overhead	Possesses overhead of repeated function calls.	No overhead as there are no function calls in iteration.
Infinite Repetition	If the recursive function does not meet to a termination condition or the base case is not defined or is never reached then it leads to a stack overflow error and there is a chance that the an system may crash in infinite recursion.	If the control condition of the iteration statement never becomes false or the control variable does not reach the termination value, then it will cause infinite loop. On the infinite loop, it uses the CPU cycles again and again.

Q1. Explain complexity in data structure?

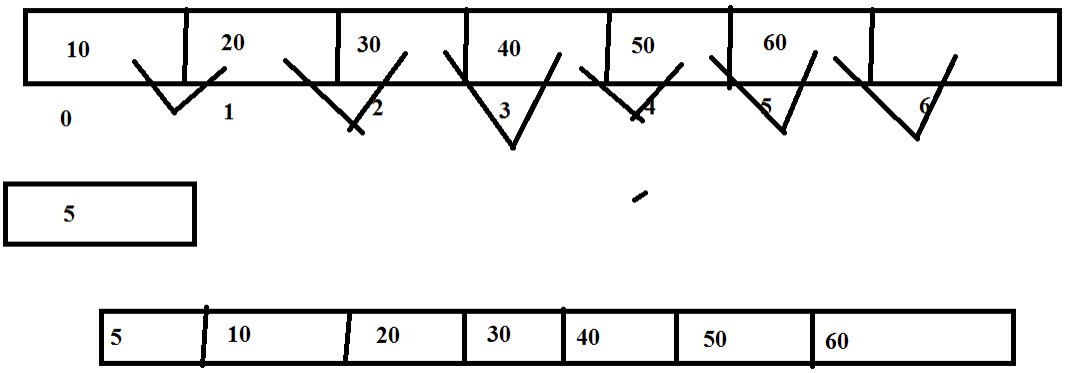
A data structure is the organization of the data in a way so that it can be used efficiently

Efficiency of the data structures is always measured in terms of time and space.

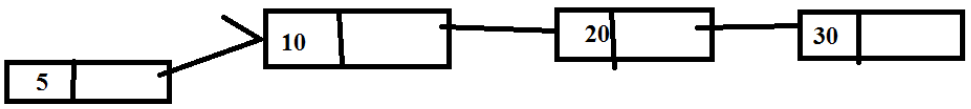
An **ideal** data structure could be the one that takes the least possible time for all its operations and consumes the least memory space.

on what basis we could compare the time complexity of the data structure

A. on the basis of operation performed them



Array



Linked List

Q1. How to find the time complexity?

Ans-->

Method 1: Examine the exact running time

pick some machine and turn the timer on. run the operation for different inputs on data structure you want to compare on by one and see how much time a particular operation will take on these data structures.

The one who takes less time is the best performer

for better analysis you can compare the data structure on different machine

problem with this approach

it might be possible some input size first data structures gives the best performer and for the other input size, and second gives the best performer.

therefore the exact running time is not the best solution to calculate the time complexity

Q3. How to find complexity of f(n)?

Ans-->we can compare two data structure for a particular operation by comparing their f(n) values

we interested in growth rate f(n)

$f(n)=5n^2+6n+12$

for n=1

% of running time due $5^{2=}$ (5/(5+6)+12)*100=21.74%

5*1=5

6*1=6

12=12

1000+0+0

N	$5n^2$	6n	12
1	21.74%	26.09%	52.17%
10	87.41%	10.49%	2.09%
100	98.79%	1.19%	0.02%
1000	99.88%	0.12%	0.0002%

n*n+4n+3--------->o(n^2)

Q1. Explain o(1) ?

Ans-->Time complexity of a function (or a set statement) is considered as 0(1) if it doesn't contain loop recursion and call to any other non-constant time function

Example

```
#include<stido.h>

void main(){

int a,b,x;

printf("Enter A : ");

scanf("%d",&a);

printf("Enter B : ");

scanf("%d",&b);

x=a;

a=b;

b=x;

printf("After Swapping A=%d  B=%d ",a,b);




}
```

Q2. Explain o(n)?

Ans--> The time complexity of a loop is considered as O(n) in the loop variable, incremented/decremented, by a constant amount for example following program have O(n) time complexity

//

```
#include<stdio.h>

void main(){


int i,n=23; //constant O(1)

                    for(i=1;i<=n;i++){

   printf("\n %d ",i);//statement--------->n time

}


}

// O(n)+O(1)=O(n)
```

Q3. Explain $O(n^2)$?

Ans--> The time complexity of a nested loop is considered as $O(n^2)$ in the loop variable, incremented/decremented, by a constant amount for example following program have $O(n^2)$ time complexity

```
//

#include<stdio.h>

void main(){

int n=10;

int i,j;

for(i=1;i<=n;i++){ //outer loop n times

   for(j=i;j<=n;j++){//inner loop n times


   }

}

}

// n*n=O(n^2)
```

```
---------------------------------------------------------------------------------------------------------------------------
--------------
//
#include<stdio.h>
void main(){
int n,i;
printf("Enter Any Number : ");
scanf("%d",&n);
if(n==0){  //O(1)
   printf("This is 0");
}
else{

   for(i=1;i<=n;i++){
      printf("\n%d ",i); // O(n)
   }
}



}

//O(n)+O(1)=O(n)
---------------------------------------------------------------------------------------------------------------------------
//
#include<stdio.h>
void main(){
int i,j,n=23;
for(i=1;i<=n;i++){
```

```
   printf("\n%d ",i);  //n times
}

for(j=1;j<=n;j++){
   printf("\n %d ",j); //n times
}

}

//O(n)+O(n)=O(n)
```

Q1. What is Logarithm?

Ans-->

$\log_2(8)=3$

The above algorithm says "how many times 2 has been multiplied by itself in order to obtain value 8"

Logarithmic time complexity is achieved when the problem size is cut down by fraction

iter 1 Initial i=1 $2^0=1$

iter 2 i=2 2^1

iter 3 i=4 2^2

iter 4 i=8 2^3

iter 5 i=16 2^4

iter k $i=n=2^{k-1}$

$n=2^{k-1}$

$k-1=\log_2 n$

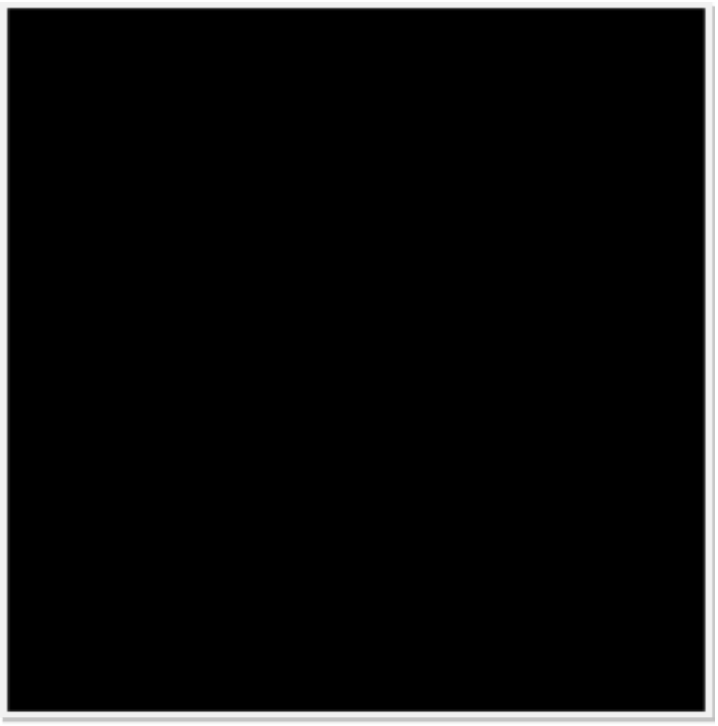
$k=\log_2 32 +1$

$k=5+1$

$=6$

if out program as a consecutive statement

```
int x=2;

int i;

x=x+1;

for(i=1;i<=n;i++){

//statement

} //O(n)
```



```
void abc(int n){

if(n<=1){

return;

}

int i;

for(i=1;i<=n;i++){

        printf("\nHello");

}


}
```

}

How Many Times the inner loop will get executed

I=1	I=2	I=3						
Inner loop only 1 time	Inner loop only 1 time	Inner loop only 1 time						

Q1. How to calculate time complexity of recursive function?

Ans-->

}

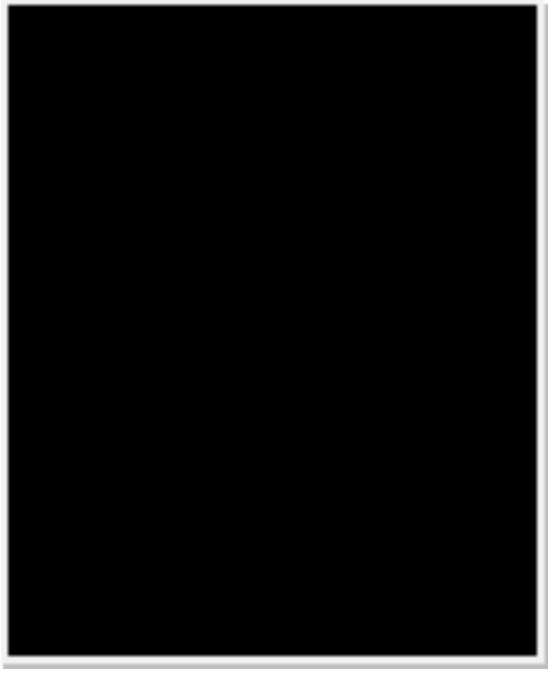
Q2. Time Complexity (n log n)?

Ans--> Any algorithm that repeatedly divides a set of data in half and then processes those halves independently with a sub algorithm that has a time complexity of O(n), will have an overall time complexity of O(n logn).

Q3. what will be time complexity of the below program

Q4. what will be time complexity of the below program

Q5. what will be time complexity of the below program



Q5. what will be time complexity of the below program