★ Recursion — very very much important

Shrinath
Date
Page No.

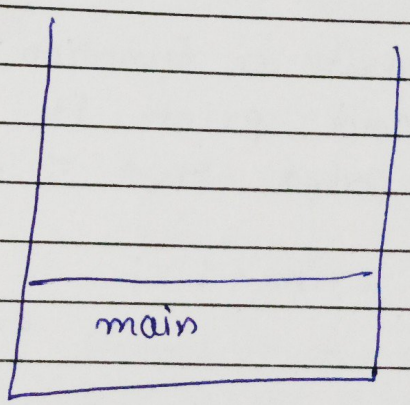
Recursion is confussing never Give up, because it is important for all the DP, and tree & ... problems.

1 week

pre requirests → 1. function
2. memory managment

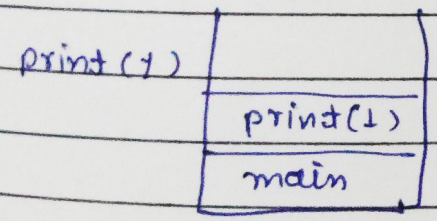
revice MysirG function

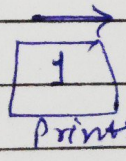
how function calls work in languages

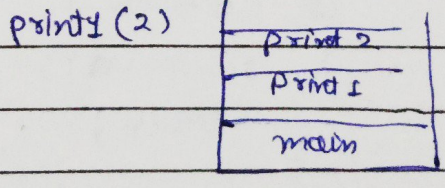


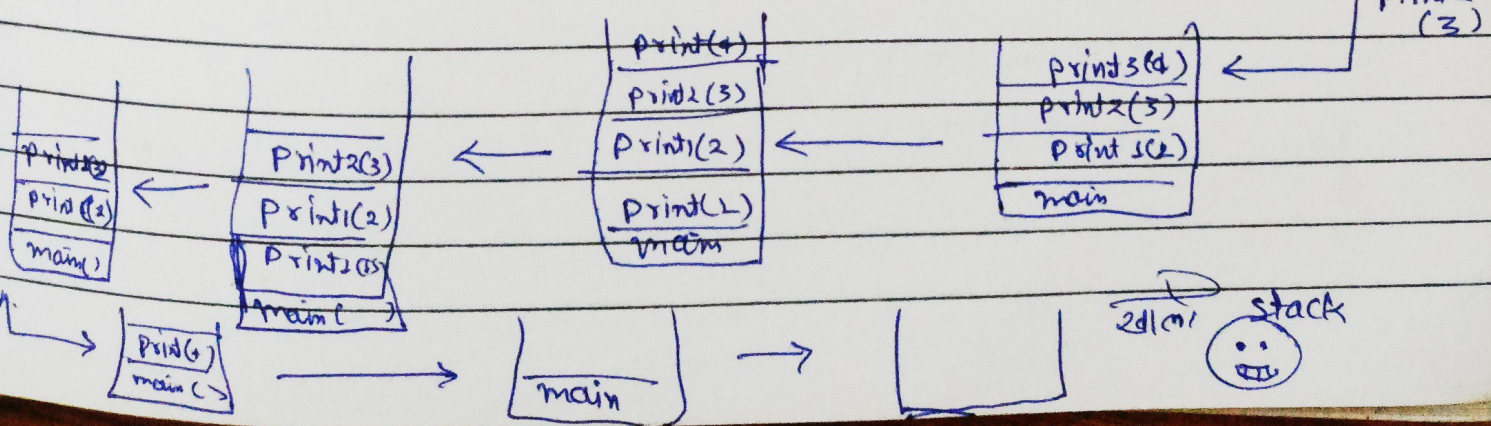
Stack memory

★ while the function is not finished executing, it will remain in stack









Shrinath
Date
Page No.

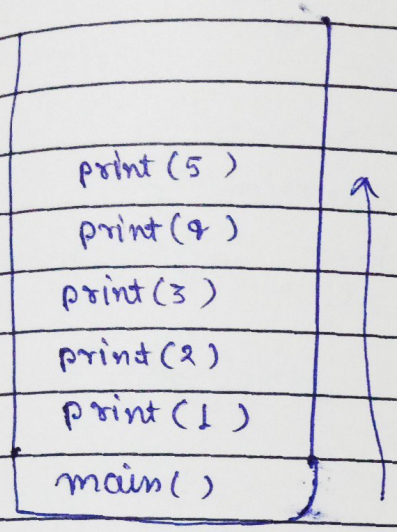
* when a function finishes executing, it is removed from stack and the flow of program is restored to where that function was called.

Internal working of Recursive function

```
public class NumbersExampleRecursion {

    public static void main(String[] args)
    {
        // write a function that takes in a number
           and prints it
        // print first 5 numbers : 1 2 3 4 5
        print(1);
    }

    static void print(int n)
    {
        if (n==5)
        { s.o.pln(n);
          return;
        }

        S.o.pln(n);
        print(n+1);
    }
```

Shrinath
Date
Page No.

Base condition - in recursion base condition is condition where our recursion will stop making new calls



Stack memory

no base condition -
means function calls will keep happening, stack will be keep getting filled again & again and we know that every call of function takes memory.

→ memory of computer will exceed the limit →
Stack Overflow error

why recursion ?

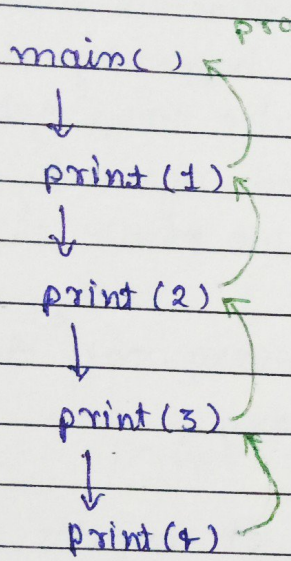
function calling itself is recursion

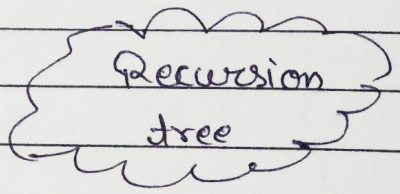
Why - * It helps us in solving bigger complex problems in a simple way.
* you can convert the recursion solution into iteration & vise versa

Shrinath
Date
Page No.

* Space complexity is not constant because of recursive calls.
* it helps us in breaking down bigger problems into smaller problems.

Visualising recursion





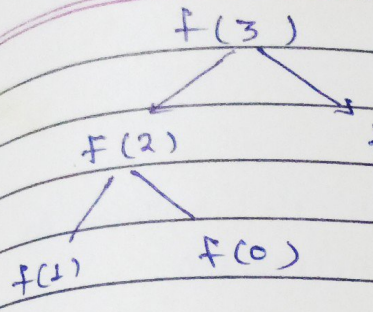
Fibonacci numbers series ⇒

0th	1st	2nd	3rd	4th	5th	6th	
0	1	1	2	3	5	8	....
				n-1	n	(n-1)+n	

Q. Find n^{th} fibonacii number

fib (N) = fib (N-1) + fib (N-2)

Shrinath
Date
Page No.



* Break it down into smaller problems
* The base condition is represented by answers we already have

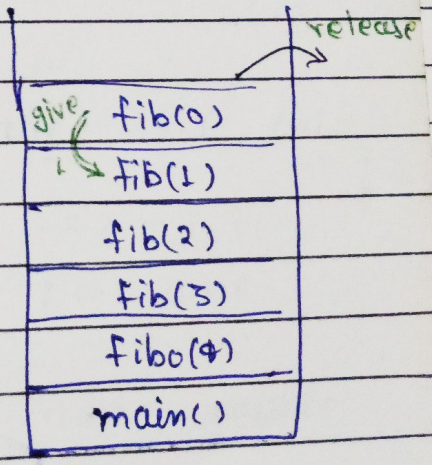
fib(N) = fib(N-1) + fib(N-2) → This is known as recurrence relation

↳ recursion in a formula is recurrence relation

in fibonacii e.g.

The base condition f(0) = 0
f(1) = 1

Base conditions are the answers that are already provided to you.




```
pulbic class F {

    p. s. v. main ( String[] args )
    {
        S.O.P ( fibo(3) );
    }

static int fibo (int n)
{
    if (n < 2)
    { return n ; }
    return fibo(n-1) + fibo(n-2) ;
```

~~Note~~ -

```
void print ( int n )
{
    if ( n == 5 )
    {
        s.o.pln ( n );
        return ;
    }
    s.o.pln ( n );
    print ( n+1 );
}
```

This is known as tail recursion

print(1) ने अपना काम कर दिया
+ print कर दिया
and
print(2); कॉल किया।

this is the last statement in the func.

---

in Fibonacii

```
int fibo ( int n )
{
    if ( n < 2 )
    return n ;

    return fib(n-1) + fib(n-2) ;
}
```

This is not tail recursion

this is the last statement in the fun

Note - How to understand and approch a problem.

1. Identify if you can break down problem into smaller problem.

2. write the recurrence relation if needed

3. Draw the recursive tree.
   - See the flow of function, how they are getting in stack
   - identify & focus on left tree calls and right tree calls.
   - Draw the tree and pointers again and again using pen and peper.
   - use a debugger to see the flow

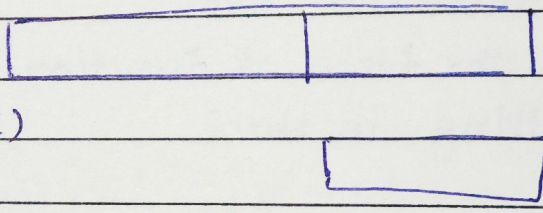
5. See how the values are returned at each step.
   see where the function call will come out off.
   in the end you will come out of the main function

Shrinath
Date
Page No.

Key areas to be focused for Recursion -

1. function calls
2. variables
   ① Argument
   ② return type
   ③ Body of function.

Binary search with recursion



① Comparing O(1)

② ① Dividing into two parts

$F(N) = O(1) + F\left(\frac{N}{2}\right)$ → recurrence relation

comparision — dividing

Types of recurrence relation -

1. Linear recurrence relation → e.g. fib. number
2. ① Divide & conquare recu. rela. → e.g. Binary search

Shrinath
Date
Page No.

L. x.	D. & C.
reducing it linearly	divide by a factor
inefficient	
Fibbonacii for 50 program stuck	
~~reasone~~ - function calls are being reapeated	
Dynamic Programming	

Tip - Do not overthink.

Argument → it is going to go in the next function call. आगे pass करना है काम के है तो argument में

~~stac~~ search (arr , target , s , e)

Body of function → specitic to that call

सिर्फ उसी func. में काम है var. का दूसरे fun call में नहीं है तो body में लिखो।

Make sure to return the result of a function call of the return type.

卐 Shrinath
Date
Page No.

```
public class BS {

    public static void main ( String[] args )
    {

    }

    int search ( int[] arr , int target , int s , int e )
    {
        if ( s > e )
            return -1 ;

        int m = s + (e - s) / 2 ;

        if (arr[m] == target )
            return m ;

        if ( target < arr[m] )
    return      search ( arr , target , s , m-1 ) ;

    return      search ( arr , target , m+1 , e ) ;
        if ( target
    }
}
```

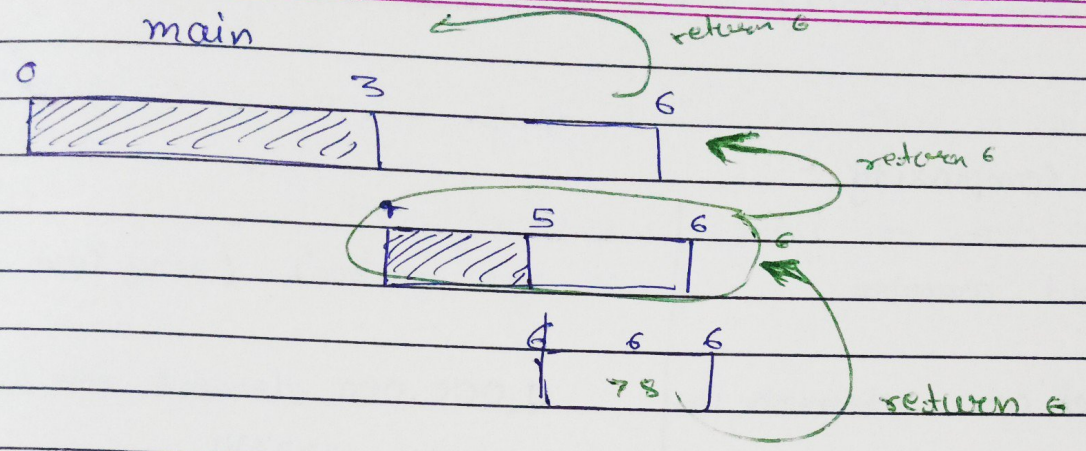
arr = { 1, 2, 3, 4, 55, 66, 78 }

target = 78

Shrinath

Date

Page No.



Shrinath
Date
Page No.

Complaxity

Time complaxity -

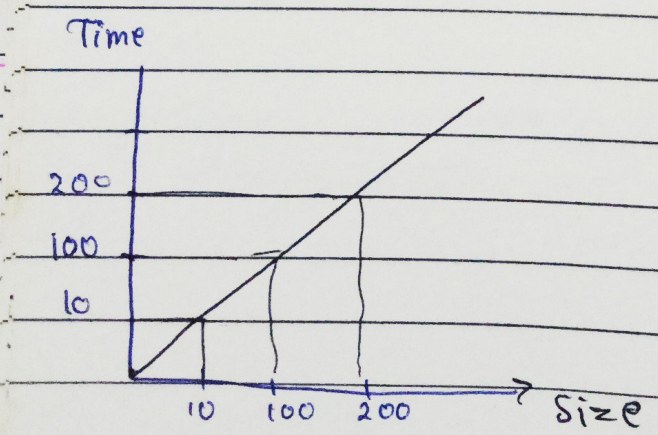
old computer	my macbook (very fast)
1. data 1,000,000 elements in an array	1,000,000 element in an array
2. Algo - linear search for target that does not exist on the array	
3. Time taken - 10 sec.	Time sec. 1 sec.

* Both machines have same time complexity

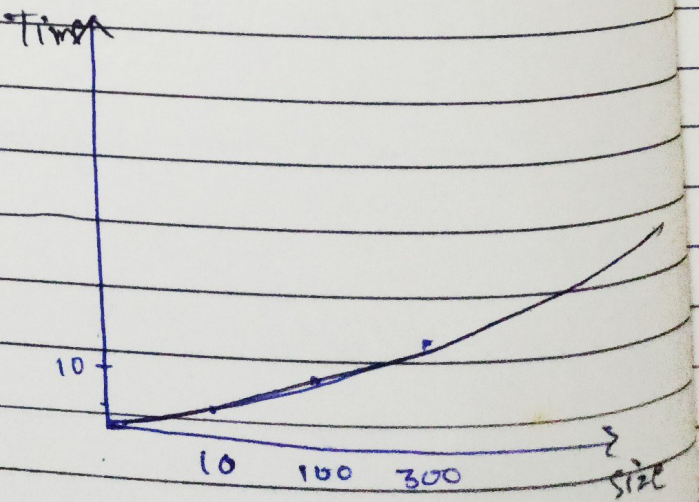
Time complexity != Time taken

4.

Old machine



My macbook



卐 Shrinath
Date
Page No.

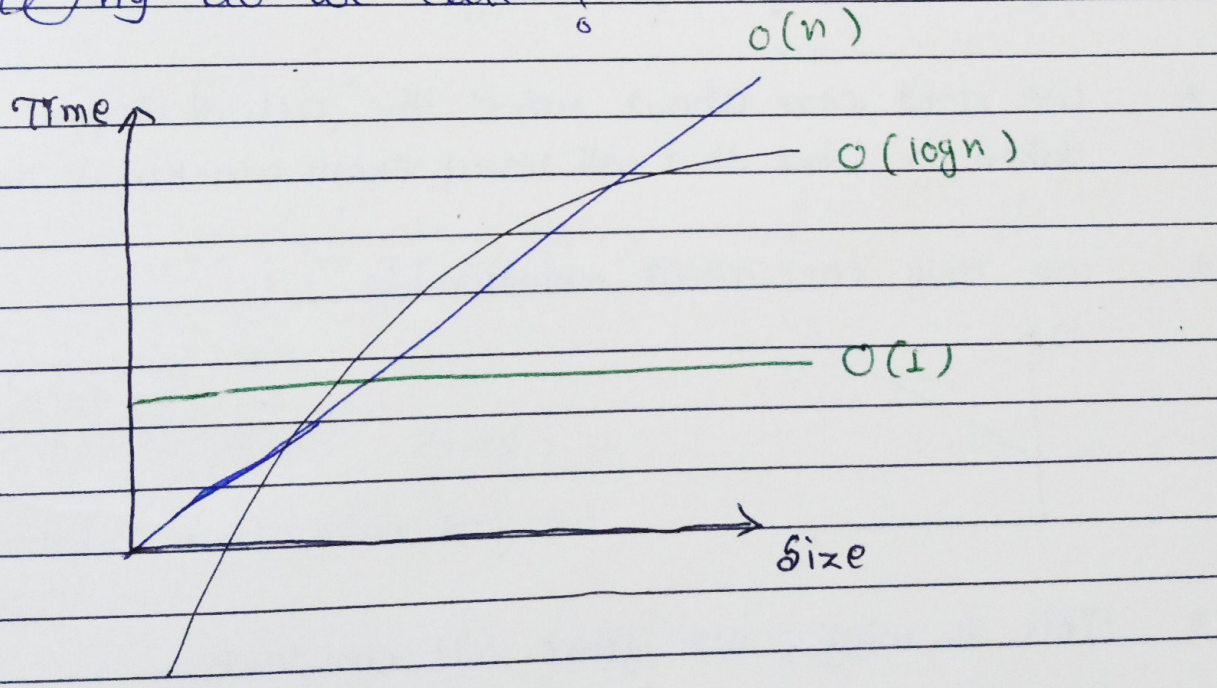
Time complexity - it is a function that tells us how the time is going to grow as the input grows.

old machine — new mac

time is growing linearly == time is growing linearly

* Function that gives us the relationship about how the time will grow as the input grows.

Why do we care ?



for large size data

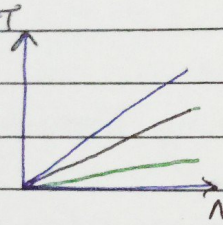
time complexity

$O(1) < O(\log n) < O(N)$

Shrinath
Date
Page No.

What do we consider when thinking about complexity

1. Always look for worst case complexity
2. Always look at complexity for large/∞ data



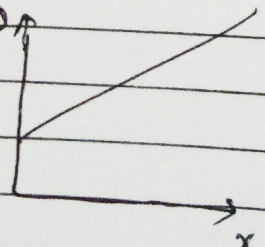
O(N)

$y = x$
$y = 2x$
$y = 9x$

* Even though value of actual time is different they are all growing linearly.

* We don't care about what the actual time taken is (bcz that will vary from machin to machin)

* We only care about relationship



$y = 3x + 5$

We just care about relationship

* This is why, we ignore all constants

4. $O(N^3 + \log N)$ from point no ②

imagine
it da
comp
4.
ex.
by
by
Time
for large
O(1) <

卐 Shrinath
Date
Page No.

imagine
if data N = 1 million

complexity ⇒ $(1 \text{ mill.})^3 + \log(1 \text{ mill.})$

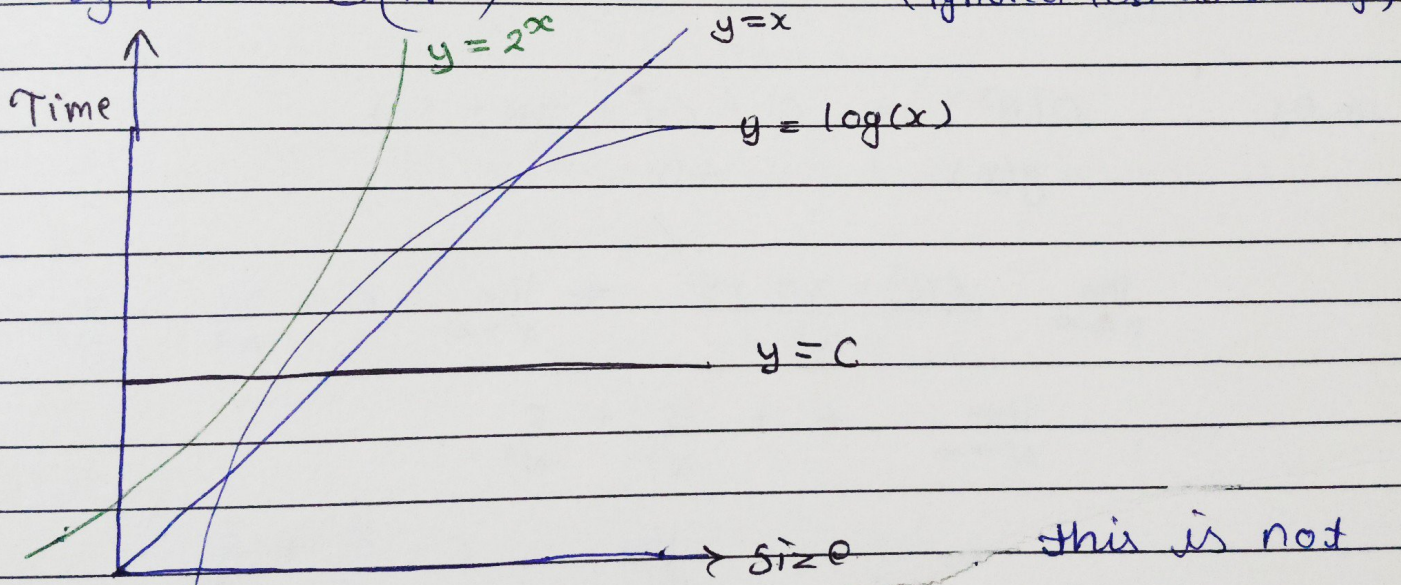
⇒ $(1 \text{ mill.})^3 + \underbrace{6 \text{ sec.}}_{\text{very small - ignore}}$ hence

★ 4. Always ignore less dominating term.

ex. $O(3N^3 + 7N^2 + 5N + 6)$

by point 3 ⇒ $O(N^3 + N^2 + N)$ (ignore constants)

by p. 4 ⇒ $O(N^3)$ (ignored less dominating)



for large data

this is not upper bound

$O(1) < O(\log(N)) < O(N) < O(C)$

$O(N \log N) < O(N^2 \log N)$

Big - oh notation

deff. ⇒ upper bound ⇒ your algo. will not exid this complexity.

worst case

1. $f(n) = O(g(n))$

2. $\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$

always look for large value

इससे कम complexity हो सकता है पर इससे ज्यादा कभी नहीं होगी।

3. in. e.g. $O(N^3) = O(6N^3 + 3N + 5)$

$g(n)$ $f(n)$

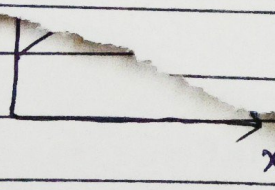
$$\lim_{n \to \infty} \frac{6N^3 + 3N + 5}{N^3} \Rightarrow \lim_{N \to \infty} 6 + \frac{3N}{N^3} + \frac{5}{N^3}$$

4. $$\lim_{N \to \infty} 6 + \frac{3}{N^2} + \frac{5}{N^3}$$

$$\lim_{N \to \infty} 6 + \frac{3}{\infty} + \frac{5}{\infty}$$

$6 < \infty$

finite value



★ This is why, we ignore

4. $O(N^3 + \log N)$ from pair

Shrinath
Date
Page No.

Big Omega - oppsite of Big oh

lower bound

best case

$$\lim_{n \to \infty} \frac{f(x)}{g(x)} > 0$$

Question - what if an algorithm has lower bound and upper bound ~~of~~ as $\Theta(N^2)$

$O(N^2)$ & $\Omega(N^2)$

Thetha notation $\Theta(N^2)$

Both upper bound & lower bound is equal to

$$0 < \lim_{N \to \infty} \frac{f(n)}{g(n)} < \infty$$

little O notation

* This is also giving upper bound, this is not a strict upper bound

Words : loose upper bound

Shrinath
Date
Page No.

```
for ( i=1 ; i≤N ; )
{
  for ( j=1 ; j ≤ K ; j++ )
  {
      // some operation that
      // take time t
  }
    i = i+K ;
}
```

inner loop is running K times and taking t time
$O(Kt)$ time

Ans ⇒ $O($ Kt + times outer loop is running ?$)$

$i = 1, 1+K, \underbrace{1+2K}_{1+K+K}, \underbrace{1+3K}_{1+K+K+K} \cdots\cdots 1+xK$

$1 + xK \leq N$

$xK \leq N-1$

$x = \frac{N-1}{K}$

Ans ⇒ $O\left(Kt + \frac{N-1}{K}\right)$

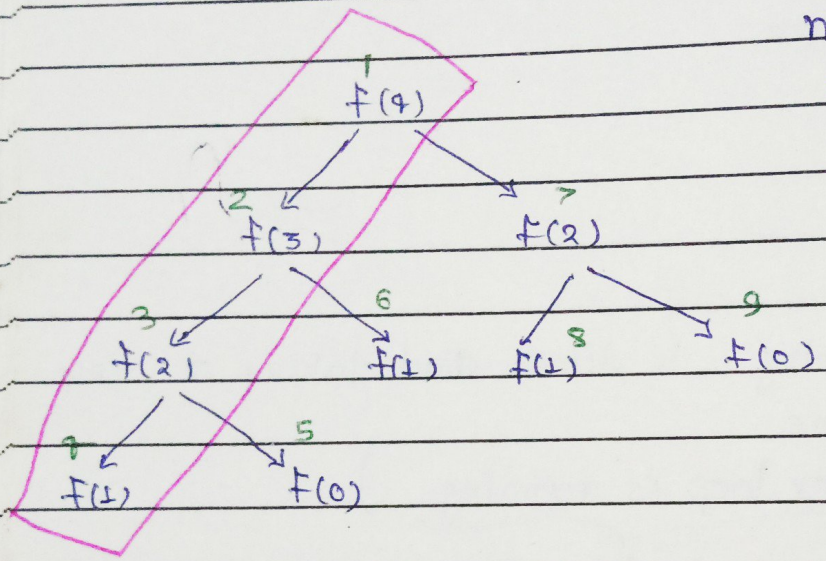
Bubble sort, Selection sort, Insertion sort

watch notes kunal's

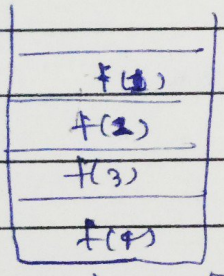
Shrinath
Date
Page No.

Recursive algorithms

in fibonacci ex. -



Space complexity -
not constant because in recursion function calls are stored in stacks, and these fun. calls take some memory in stacks,
hence recursive programs do not have constant space complexity



flow of recursive program

a fun call can not be in the stack at the same time,

longest chain starting from the root till the leaf

Space complexity = high of the tree (path)

O(N)

Shrinath
Date
Page No.

2 types of recursions

① Linear (i.e. fibonacci)

$f(N) = f(N-1) + f(N-2)$

② Divid & conquer (e.g. binary search)

$f(N) = f\left(\frac{N}{2}\right) + O(1)$

Recurrence Relations

Divide & conquare recurrence – How to identity wether an equation is of div & con or not

Form :

$$T(x) = a_1 T(b_1 x + \xi_1(x)) + a_2 T(b_2 x + \xi_2(x)) + \cdots + a_k T(b_k x + \xi_k(x)) + g(x)$$

for $x \geq x_0$ ↳ some constant

e.g.
B.C. binary $T(N) = T\left(\frac{N}{2}\right) + C$

by comparing

$a_1 = 1$

$b_1 = \frac{1}{2}$

$\xi_1(x) = 0$

$g(x) = C$

$T(N) = 9\,T\left(\frac{N}{3}\right) + \frac{9}{3}\,T\left(\frac{5N}{6}\right) + 7N^3$

a_1 b_1 a_2 b_2 $g(N)$

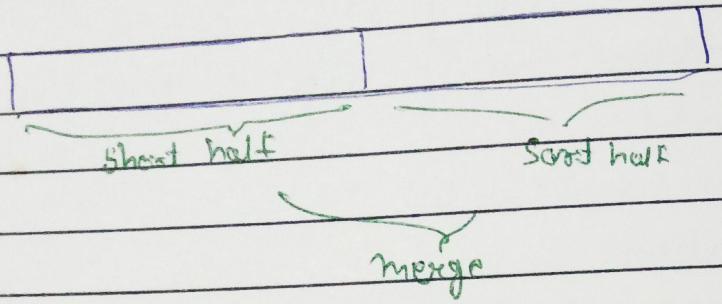
$g(x)$ / $g(N)$ means what ?

$T(N) = \underbrace{2\,T\left(\frac{N}{2}\right)}_{a_1 \quad b_1} + \underbrace{(N-1)}_{g(N)}$

→ whe you get ans from this & what you are doing with answer takes how much time

Shrinath
Date
Page No.

in merge sort



$$T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{2}\right) + \underbrace{(N-1)}_{\text{merging takes}}$$

$$= 2T\left(\frac{N}{2}\right) + \underbrace{(N-1)}_{g(N)}$$

How to actually solve the to get complexity

(1) Plug & chug.

$$F(N) = F\left(\frac{N}{2}\right) + C$$ <west of time>

(2) Masters Theorem <no need to do>

(3) ★★★ Akra-Bazzi (1996)

Shrinath
Date
Page No.

- print 1, 2, 3, 4, 5
- print n 1
- factorial of a number
- Sum of n to 1 } same
- Product of n to 1
- ~~Sum of digits~~
- product of digits
- Reverse a number using recursion

- palindrom
- count zeros
- count steps

```
class A
{
    p.s.v.m( ~~ )
    {   fun(5);
    }

static void fun ( int 5 )
{
    if (n == 0)
        return;
    System.out.println(n);
        fun(n--);
}
```

// n-- v/s --n both are n = n - 1

but n-- post prefix

पहले काम होगा फिर -- होगा

तो हर बार 5 pass होगा।

Shrinath
Date
Page No.

Reverse of a number

```
int rev = 0

void fun (n)
{
    if (n==0)
        return ;

    int rem = n %10;
    rev = rev * 10 + rem;
    fun (n/10) ;
}
```

```
static boolean Palind (int n)
{
    return n == rev2(n);
}
```

```
static void rev2 ( int n )
{ // sometimes you might need some additional variables in the argument
  in that case make another function(helper)

    int digits = (int)(Math.log10(n)) +1 ;
    return helper (n, digits);
}

static int helper ( int n , int digit)
{
    if (n%10 == n)
        return;

    int rem = n %10;

return rem*(int)Math.pow(10 , digits-1) + helper ( n/10 , digits-1);
}
```

Shrinath
Date
Page No.

Important question for count

pattern have to return same vallie to above fun call

```
static int count (int n)
{
    return helper(n, 0);
}

private static int helper( int n , int c )
{
    if (n==0)
        return c;

    int rem = n %10
    if (rem==0)
        return helper( n/10 , c+1);

    return helper( n/10 , c);
}
```

c को increase करते गये, last में c को return किया

Leetcode easy

Number of steps to reduce a Number to Zero

(argument) → में उतनी ले जाना जिनकी जरूरत है।

Shrinath
Date
Page No.

Recursion → Arrays

Q.1 Sorted Array
Q.2. Linear search
Q.3. Linear search (on Multiple occurrences)
Q.4 Return an ArrayList
Q.5 Return the list without passing the argument.
Q.6 Rotated Binary Search

Prefer Kunal's notes

(1) is the array sorted or not cheak by recursion
⇒ try by yourself if fail you can coatch vidio again
&&

(2) Linear search with recursion.
⇒ try by youreself
||

```
static boolean find (int[] arr, int target, int index
{   if (index == arr.length)
        { return false; }
    return arr[index] == target || find (arr, target, index+
}
```

(3) Multiple occurence (use list) ArrayList

(4)

```
static Arraylist findAllIndex( int[] arr, int target, int index,
                                Arraylist<Integer> list )
{ if ( index = arr.length )
      { return list }
  if (arr[index] == target)
      { list.add (index); }
  findAllIndex ( arr, target, index+1, list )
```

Shrinath
Date
Page No.

(5) return the list without passing it in argument

return type → ArrayList

Problem - every call will have new arraylist

argument → (arr, target, 0)

// not optimised but sometime we may need

```
static ArrayList<Integer> findAllIndex(int[] arr, int target, int index)
{
    ArrayList<Integer> list = new ArrayList<>();

    if (index == arr.length)
    { return list; }

    // this will contain answer for that function call only

    if (arr[index] == target)
    { list.add(index); }

    ArrayList<Integer> ansFromBelowCalls = findAllIndex(arr, target, index+1);
    list.addAll(ansFromBelowCalls);

    return list;
}
```

(watch again & revise rotated binary search)

Shrinath
Date
Page No.

Rotated Binary Search

```
static int search (int[] arr, int target, int s, int e)
{
    if (s > e) {return -1;}
int m = s + (e - s)/2;
if (arr[m] == target) {return m}

if (arr[s] <= target arr[m])
{
  if (target >= arr[s] && target <= arr[mid])
    return search (arr, target, s, m-1);
  else return search (arr, target, m+1, e);
}

if (target >= arr[m] && target <= arr[e])
return search (arr, target, m+1, e);
return search (arr, target, s, m-1);
```