DATE 09 03 2024

⊛ Nested loop : Pattern question.

1) Half Pyramid :

```
*
* *
* * *
* * * *
* * * * *
```

```
for (int i=1; i≤n; i++){
    for (j=1; j≤i; j++){
        System.out.print("*"+" ");
    }
    System.out.println();
}
```

2) Inverted half pyramid :

```
* * * * *
* * * *
* * *
* *
*
```

```
for (int i=1; i≤n; i++){
    for (int j=n; j≥i; j--){
        print("* ");
    }
    println();
}
```

3) Hollow Inverted half pyramid :

```
1  5 * 4 * 3 * 2 * 1 *  (1,1)
2  *         *  (2,2)
3  *       *  (3,3)
4  *   *  (4,4)
5  *  (5,5)
```

```
for (int i=1; i≤n; i++){
    for (int j=n; j≥i; j--){
        if (i==j || i=1 || j=n){
            print("* ");
        else{
            print(" ");
        }
    }
    println();
}
```

DATE

4) Hollow Square

```
1  *  *  *  *  *
2  *           *
3  *           *
4  *           *
5  *  *  *  *  *
```

```
for( int i=1 ; i≤n ; i++){
    for (j=1; j≤n; j++) {
        if ( i==1 || j==1 || i==n || j==n)
            print ("*");
        else {
            print (" ");
        }
    }
    println( );
}
```

5) Opposite pyramid :

```
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
```

```
for (int i=1; i≤n ; i++){
    for(int j=n ; j≥1 ; j--){
        if( i≥j ){
            print ("*");
        else {
            print (" ");
        }
    }
    println();
}
```

6) Inverted opposite pyramid :

```
*  *  *  *  *
   *  *  *  *
      *  *  *
         *  *
            *
```

```
i= 1 to n ++
j= 1 to n ++
if( i≤j ){
    print ("*");
else {
    print (" ");
}
```

DATE

7) Solid half diamond:

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

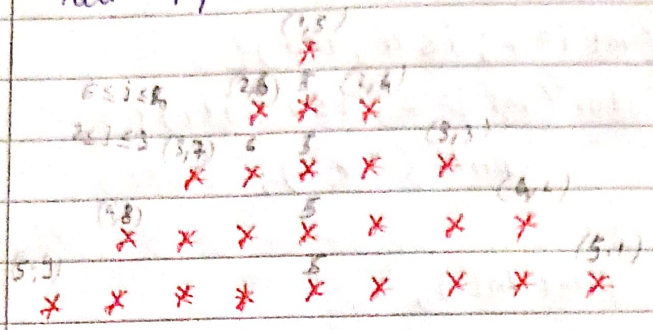
```
for (int i= 1; i≤ n; i++){
    for (int j= 1; j≤ i; j++){
        print ("*");
    }
    println();
}
for (int i=1; i≤n-1; i++){
    for (int j=n-1; j≥i; j--){
        print ("*");
    }
    println();
}
```

2nd method:
(for odd number):

```
for (int i= 1; i≤ n; i++){
    ~~for (int~~
    if (i≤ n/2) {
        for (int j= 1; j≤ i; j++) {
            print ("*");
        }
        system.out.println();
    }
    else {
        for (int j= n; j≥i; j--){
            print ("*");
        }
        println();
    }
}
```

DATE

7) full pyramid :



DATE

8) Solid half diamond :

```
        *
      *   *
    *   *   *
  *   *   *   *
*   *   *   *   *
  *   *   *   *
    *   *   *
      *   *
        *
```

n = 9

```
for( int i=1; i≤ n; i++ ){
    if(i≤5){
        for (int j=5; j≥1; j--){
            if (j≤i){
                print("*"+" ");
            }
            else {
                System.out.print("  ");
            } ~~print ln();~~
        } print ln();
    } else {
        for (int j=1; j≤5; j++){
            if (j≤(i-5)){
                print("  ");
            else {
                print("*"+" ");
            }
        } print ln();
    }
}
```


classmate

(*) Array :-

DATE

* Array :-

Syntax : type [ ] arrayName = new type[size];

type - int, float, String, boolean etc

eg: int [ ] marks = new int[4];

marks[0] = 30;

marks[1] = 27;

marks[2] = 25;

marks[3] = 20;

marks[4] = 21; ----> Shows error because length is 4.

In memory → | 30 | 27 | 25 | 20 |

- Index is start from 0.
- print the marks → System.out.println(marks[1]);

  o/p : 27.

∴ Suppose size of array is 3 and there type is int then total size of array in memory is 12

- Array is also defined by,

  int marks[ ] = {30, 27, 25, 20};

print the array by using for loop,

```
∴ for( int i=0; i< marks.length; i++){
        System.out.println(marks[i]);
    }
```

- User defined array & input element of array.

```
import java.util.Scanner;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array : ");
        int size = sc.nextInt();
        print(" Enter the elements : ");
        for( int i=0; i< size; i++){
            marks[i] = sc.nextInt();
        }
```

print the array by using for-each loop.

- ~~for(element :~~ for(int element : marks) {
  println(~~m~~ elements);
  }

O/P :

- In java, array is initialize at automatically
* with zero.

eg:
```
int [] arr = new int [3];
for( int ele : arr){
    println (ele);
}
```

O/P: 0
0
0

Note: print the array in reverse order, use the for loop and start from [array.length() - 1] to 0 & decreament by 1.

Q 1) Check the element present in the array

eg: 25

```
int [] marks = {30, 20, 19, 17, 25, 14};
for ( int ele : marks) {
    if ( ele == 25) {
        println( "present");
    }
}
```

O/P: present.

Swaping process is continuous for Interation is continue - Bubble swaping only one time for 2 interation - Selection.

DATE

Q2) Sorting the array (accending order):

eg: int [] arr = {7, 8, 3, 1, 2};

There are three type of sorting

1) Bubble sort
2) Selection sort
3) Insertion sort.

1) Bubble sort

- In bubble sort, they are compare two digits and which number is greater & sent in last. this process in n-1 times.

eg: given array | 7 | 8 | 3 | 1 | 2 |

Step 1 : Compare two-two elements.

| 7 | 8 | 3 | 1 | 2 |

compare

Step 2 : | 7 | 8 | 3 | 1 | 2 |

compare

Step 3 : | 7 | 3 | 8 | 1 | 2 |

Step 4 : | 7 | 3 | 1 | 8 | 2 |

complete : | 7 | 3 | 1 | 2 | 8 |

8 is maximum number go to the last.

--- this process is continuous ~~for~~ up to sorting the array

| 7 | 3 | 1 | 2 | 8 |

| 3 | 7 | 1 | 2 | 8 |

| 3 | 1 | 7 | 2 | 8 |

| 3 | 1 | 2 | 7 | 8 |

In final | 1 | 2 | 3 | 7 | 8 |

// Code //

```
public static void SortArray(int arr[]){
    for(int i=0; i<arr.length-1; i++){
        for (int j=0; j< arr.length-i-1; j++){
            if (arr[j] > arr[j+1]){
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
}
```

// then print the array by using loop

Note: Change the sign in if condition then array is arrange in descending order.

time complexity = $O(n^2)$

2] Selection sort:

- In selection sort algorithm, we search for lowest element and arrange it to the proper location. we swap current element with the next lowest element.
- one element is compare withe other all element and the swap with lowest element

eg: | 7 | 8 | 3 | 1 | 2 |

index.

step 1: store in smallest = 0[7]

then seven compare with 8, 7 is small; next 7 is compare with 3, 3 is small, hence value of smallest = 2[3].

then 3 is compare with 1, value of smallest = 3[1]

then 3[1] is compare 2 the smallest is 1,

∴ Smallest = 3[1] — they are swap by 0[7].

∴ newe array | 1 | 8 | 3 | 7 | 2 | -- this process is continue.

second step - process is start from the index of 1
third step process is start from the index of 2

DATE 10 03 2024

Step 2 : | 1 | 8 | 3 | 7 | 2 | ⇒ | 1 | 2 | 3 | 7 | 8 |

comparing with all and then swap

swap

step 3 : | 1 | 2 | 3 | 7 | 8 | ⇒ | 1 | 2 | 3 | 7 | 8 |

no swap but compare with index 3&4

Step 4 : | 1 | 2 | 3 | 7 | 8 | ⇒ | 1 | 2 | 3 | 7 | 8 |

no swap but compare with index 4

|| Code ||

```
for( int i=0 ; i< arr.length-1 ; i++){
        int smallest = i;
        for( int j= i+1 ; j< arr.length ; j++){
                if ( ~~arr[i]~~ arr[smallest] > arr[j]) {
                        smallest=j ;
                }
        }
        int temp = arr[smallest];
        arr[smallest] = arr[i];
        arr[i] = temp;
}
        // print the array by using for loop or for each loop.
```

time complexity = $O(n^2)$

3) Insertion sort :

- Insertion sort algorithm means, element is insert in a sorted array
- array is not sorted then assume the two part of the array & one part is sorted then applied the operation on array.

classmate

eg: arr → | 7 | 8 | 3 | 1 | 2 |

this array is divide in two parts & one part is so assume sorted & other part is unsorted

like,

| 7 | 8 | 3 | 1 | 2 |

↗ sorted ↑ unsorted.

// code //

```
for (int i= 1 ; i< arr.length ; i++) {
    int current = arr[i];
    int j= i-1;
    while ( j>=0 && current < arr[j]) {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = current ;
}
// print the array //
```

Explaination :

* Time Complexity

- Rel^n bet^n input size & running time (operation).

↳ ~~Big~~ Θ : Time complexity $\propto$ Input 'n'.

Best case : Ω
average case : Θ
worst case : O

eg: search for 1,

1) Numbers : 1, 2, 3, 4, 5 — only one operation ($n=1$).
$\therefore \Omega(1)$ - best case.

2) Numbers : 1, 2, 3, 4, 5
2, 3, 4, 1, 5 } avg. $\frac{1+2+3+4+5}{5}$

$\therefore$ In general, $\frac{1+2+3+\cdots n}{n} = \frac{n(n+1)}{2 \times n} = \frac{n+1}{2}$

$\therefore$ This are linear rel^n ; $\Theta\left(\frac{n+1}{2}\right)$

3) numbers : 4, 3, 2, 5, 1
5, 2, 4, 3, 1 } $n=5$ time operation
~~, 1~~

Suppose nuber numbers is 1 lakh then operation is 1 lakh; time $\propto n$ — worth case.
$\therefore O(n)$.

ex: 2]
```
for (int i=1; i<n; i++){
    System.out.println("onkar");
}
```

where n is input from user
$\therefore$ time complexity $= n$

eg 3] Nested loop

```
for (int i=0 ; i<n ; i++){
    for (int j=0 ; j<n ; j++){
        System.out.println(" Hellow");
    }
}
```

where n is input like ⇒ int n = sc.nextInt();

outer loop run n times, inner loops runs n times.

for i = 1 → j = 1, 2, 3 .... n

i = 2 → j = 1, 2, 3, ---- n

i = 3 → j = 1, 2, 3, ---- n

⋮

i = n → j = ---- n

∴ time complexity = $n \times n = n^2$

∴ $\boxed{O(n^2).}$

- simillary, two input 'n' & 'm'. Input of outer loop is n & Input of inner loop is m.
- outer loop run n times & inner loop run m times

∴ time complexity = $m \times n$

∴ $\boxed{O(m \times n).}$

eg 4] Normal loops.

```
for (int i=0 ; i<n ; i++){
    println(" Hi");
}
```

} run n time

```
for (int j=0 ; j<m ; j++){
    println(" Hellow");
}
```

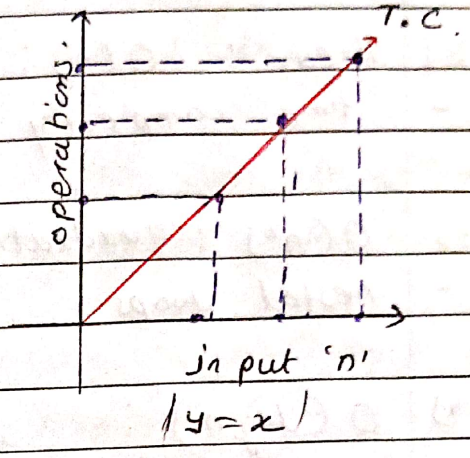
} run m time

time complexity = $m+n$ = $O(m+n)$

1] Big (O):

- denoted by O.
- time complexity is not actual the time of execute the code.
- time complexity is actual the operations.

ex 1]
```
for( int i=0; i<n; i++){
        print ln (arr[i]);
}
```



- input is increase then operation is increase.

In mathematically,

ex: $5n^2 + 4n + 20$

where n is the time complexity.

$\therefore O(n^2)$ — ans. because largest value of this expression is n^2, highly depend value this expression on n^2.

ex: 2]. $O(4n + 3n^4 + 40n^2)$:

Step 1 : Ignore all constant. → $n + n^4 + n^2$

Step 2 : big term of n & ignore all

Ans: $O(n^4)$.

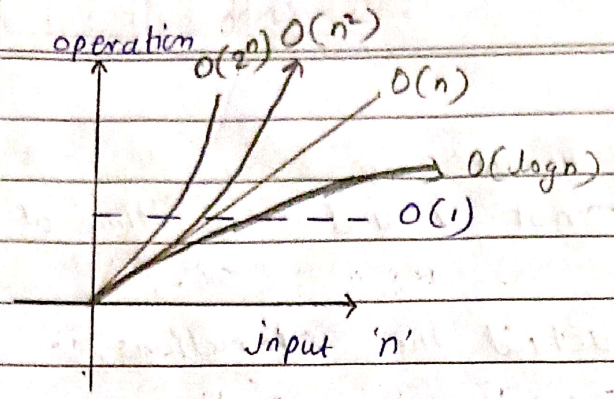
* Common complexity:

1] $O(1)$: constant

- In this, time is fix for operation

ex: input n; operation is square of the number.

classmate



$O(\log n) >> O(n)$

$O(n \log n) >> O(n^2)$

this sign is "much better"

2] linear : $O(n)$:

- time complexity is depend on n : example 1].

3) $O(n^2)$: quadratic

- nested loops :

4) $O(\log n)$:

- This complexity is better than linear $O(n)$.
- It is found in generally binary searching.

5) $O(n \log n)$

- generally found in sorting.

6) $O(2^n)$: ~~exponential~~ exponential :

- generally found in recursion.
- This time complexity doing better by using dynamic programming.

7) $O(n!)$:

- generally found in permutation of strings.

$O(1) >> O(\log n) >> O(n) >> O(n \log n) >> O(n^2) >> O(2^n) >> O(n!)$

DATE

× by solving DSA question given the constrains :

- constrains is decide to which logic to write

1 s ≅ 10^8 operation

$n > 10^8$ — use $O(\log n)$ $O(1)$

$n \leq 10^8$ — $O(n)$ using 1 loop

$n \leq 10^6$ — $O(n \log n)$ sorting

$n \leq 10^4$ — $O(n^2)$ nested loop

$n \leq 500$ — $O(n^3)$

$n \leq 25$ — $O(2^n)$ - recursion, brute force

$n \leq 12$ — $O(n!)$ - permutation & combinatio

Q: questions for practice.

1)
```
int n = sc.nextInt();
int i = n;
while (i >= 1) {
    i /= 2;
}
```

1st time loop run for n

2nd time loop run for $\frac{n}{2}$

3rd time loop run for $n/4$

$n/8$

Cond^n $n \geq 1$ $\therefore \frac{n}{2^k} \geq 1 \Rightarrow n \geq 2^k$

→ time complexity = $O(\log n)$

2]

Q. question on array :

1) I/p in array :

→ Import Scanner library.

```
int [] arr = ~~sc.nextInt()~~; new int[input_element];
for ( int i=0 ; i< ~~arr~~ input_element ; i++){
        arr[i] = sc.nextInt();
}
  // print using loop.
```

2) copy the element of array.

→

```
int arr[] = { 1, 2, 3, 4, 5 };          // copy arr in arr2 //
int arr2[] = new int[arr.length];
for (int i=0 ; i< arr.length ; i++){
        arr2[i] = arr[i];
}
  // print the array using for each loop
```

3) sum of array element

-

```
int sum=0
for (int i=0 ; i< arr.length ; i++){
        sum= sum+ arr[i];
}   // print sum //
```

4) accending order (sorting) :

- arr → { 4, 7, 8, 2, 1, 6 };

```
for (int i=0 ; i< arr.length ; i++){
        for ( int j= i+1 ; j< arr.length-1; j++){
                if ( arr[j] > arr[j+1] ){
                        int temp= arr[j];
                        arr[j]= arr[j+1];
                        arr[j+1] = temp;
                }
}
```

5) max number in array:

→ int arr[] = { 4, 3, 2, 7, 19, 12 };

```
int max = arr[0];
for (int i=1; i<arr.length; i++){
        if (arr[i] > max){
                max= arr[i];
        }
}
System.out.println(max);
```

// also solve by sorting.

6) min value in array

→ Same code of q.5 but change the cond^n

arr[i] < min ⇒ min = arr[i].

7) Print the matrix using 2-D array.

→ ~~for~~ int [][] arr = new int [3][3];

row column

I/p the element of array of 3x3 matrix.

```
for (int i=0; i<3; i++){          — for row
    for (int j=0; j<3; j++){      — for column
        arr[i][j] = sc.nextInt();
    }
}
for (int i=0; i<3; i++){
    for (int j=0; j<3; j++){
        System.out.print(arr[i][j] + " ");
    }
    System.out.println();
}
```

8) Add two matrix using 2-D array:

→ ~~int [][] arr = new int [~~

```
// create the method for input array //
public static int[][] userArrayMatrix(int row, int column){
Scanner sc = new Scanner(System.in);
int [][] arr = new int[row][column];
System.out.println("Enter the elements of mitrix:");
for(int i=0; i<row; i++){
    for(int j=0; j<column; j++){
        arr[i][j] = sc.nextInt();
    }
}
return arr;
}
// method of sum of array //
public static ~~int~~ void sumOfArray (int row, int column, int arr1[][],
                                        int arr2[][]){
    System.out.println("Theddition is:");
    int sum[][] = new int[row][column];
    for(int i=0; i< ~~column~~ row; i++){
        for(int j=0; j<column; j++){
            sum[i][j] = arr1[i][j] + arr2[i][j];
        }
    }
    for(int i=0; i<row; i++){
        for(int j=0; j<column; j++){
            System.out.print( sum[i][j]);
            System.out.print(" ");
        }
        System.out.println();  }
}
```

```
public static void main(String [] args){
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter no. of row : ");
    int row = sc.nextInt();
    System.out.print("Enter no. of column : ");
    int column = sc.nextInt();
    int arr1[][] = userArrayMatrix(row, column);
    int arr2[][] = userArrayMatrix(row, column);
    sumOfArray( row, column, arr1 ; arr2);
}
```

method called & stored in object

calling the method

* Recursion :-

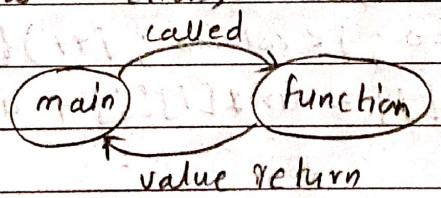
- it is use in tree & graph
- Recursion is brother of iteration.
- It is the function, the called by itself.

eg: $f(x) = x^2$

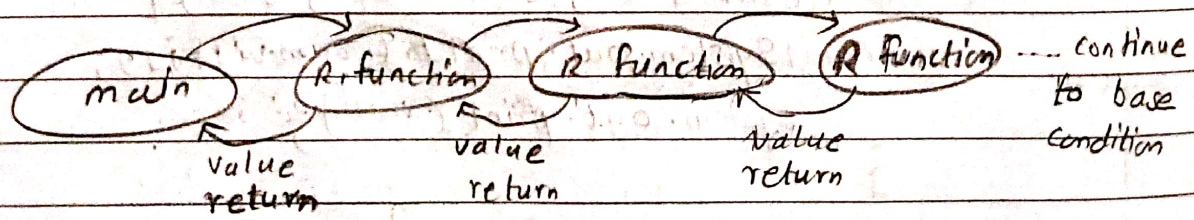
$f(f(x)) = f(x^2) = x^4$

recursion

- a normal function,



- recursion function



DATE

- It is work like a loop

Note: all iteration methods instead

* question for practice:

1) Print 'n' to '1' number by using the recursion

```
public static void printNum(int n){
        if(n==0){
            return; }
        System.out.println(n);
        printNum(n-1);
}
public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int num = sc.nextInt();
        printNum(num);
```

Note: In a recursion base condition is required; you does not write the base cond^n then recursion function called infinite times then the memory is full then show the error called stack error

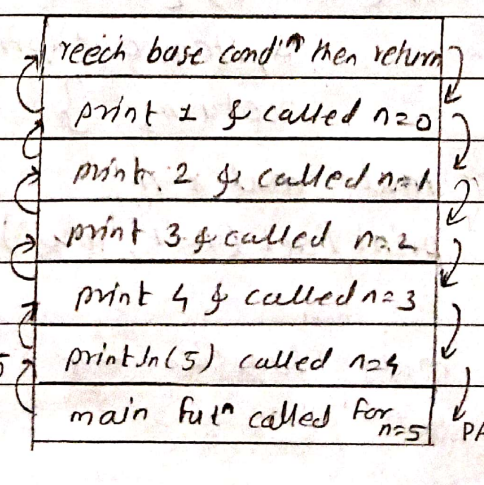
- for recursion 3 step are required

1) from main function pass the parameter

2] Base condition

3) code / work / logic etc. (काम)

In memory:



2) Print 1 to n number

```
public static void printNum(int n);
        if( n==given number ){
                return;
        System.out.println(n);
        printNum(n+1)
main {
        printNum(1);   }
```

3) Print sum of first N natural number.

```
public static ~~void~~ int sum=0;
public static void printSum(int n, int m){
        if(n==0){
                System.out.println(sum);
                return;
        }
        Sum= Sum+n;
        printSum(n-1, sum);
}
```

from main function → printSum(10, 1);

4) print factorial of number.

```
static int mul=1;
public static void printFact(int n, int m){
        if( n==1){
                System.out.println(mul);
                return;  }
        mul= mul * n;
        printfact(n-1, mul);
}
```

From main function → printFact(10, 1);

other method;

```
public static int printFact(int n);
        if (n==0 || n==1){
                return 1; }
        return n* printFact(n-1);
}
```

From main function → prin Int Fact = printFact(5);
System.out.println(Fact);

In memory:

base cond^ reech	return
" n=1	return 1
" n=2	return 2×1
called n=3	return 3×2×1
called n=4	" 4×3×2×1
called n=5	" 5×4×3×2×1
main call n=5	

Stack memory

Note: In return type, delete each stack block

5) Print fibonacci sequence:

Start with 0 & 1.

0 1 1 2 3 5 8 13 21 34 .....

```
public static void fibbonacci(int a, int b, int n){
        if (n==0){
                return; }
        int c=a+b;
        System.out.println(c);
        a=b; b=c;
        fibbonacci(a, b, n-1);
}
```

In main function → print a=0 & b=1 & called the fun^n
fibbonacci(a, b, c-2)

classmate

Note: c is the no. of term

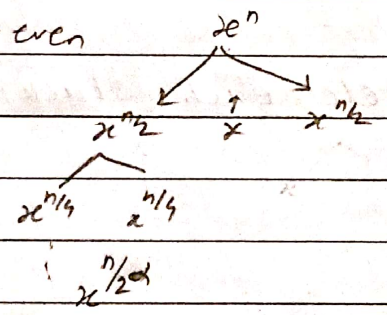
6) Calculate the power of n number : (Stack height n)

```
public static int Power (int num, int power){
        if (power==0){
            return 1;
        }
        int pow = num * Power(num, power-1);
        return pow;
}
```

main method → int num = 2
int power = 5
int ans = Power(num, power);
System.out.println(ans);

6) Stack hight = log n.

n - even



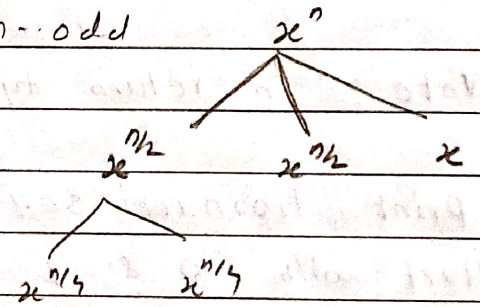
$\frac{n}{2^\alpha} = 1$

$n = 2^\alpha$

$\log_2 n = \alpha$

∴ time complexity = log n

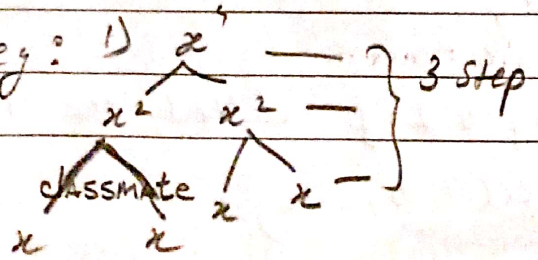
n - odd



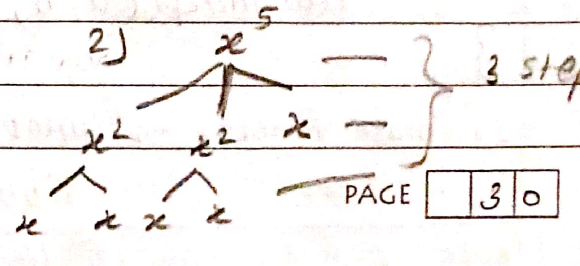
∴ (Same as n is even)

time complexity = log n

eg: 1)



2)



//code//

```
public static int calcPower(int x, int n) {
    if (n==0) {
        return 1; }
    if (x==0) {
        return 0; }
    // n is even
    if (n%2 == 0) {
        return calcPower(x, n/2) * calcPower(x, n/2);
    }else {    // n is odd
        return calcPower(x, n/2) * calcPower(x, n/2) * x;
    }
}
```

- Intermediate question on recursion :

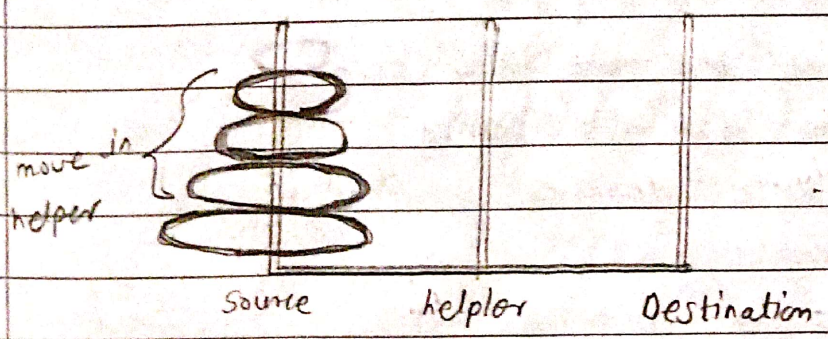
Q.1) Tower of Honoi :

rule : 1) 1 time move one disk

2) not allow to lower size disk below & greater sizer is upper

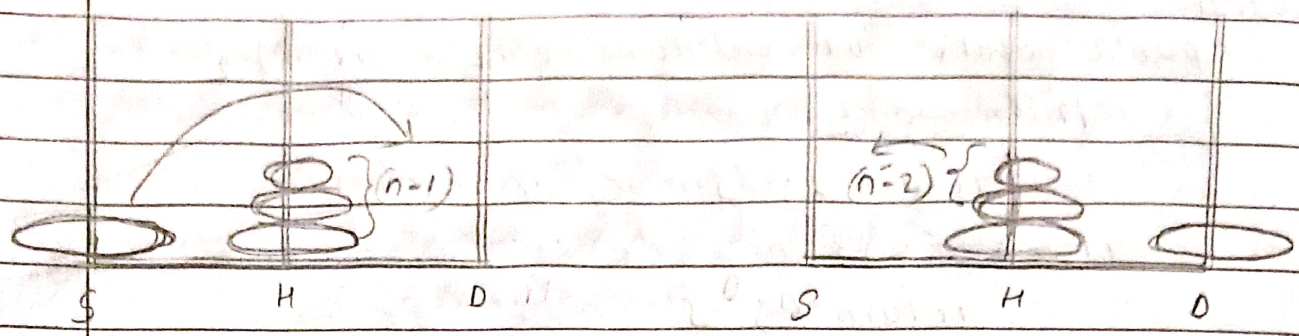
3) only allow lower sizer disk up side to max size disk

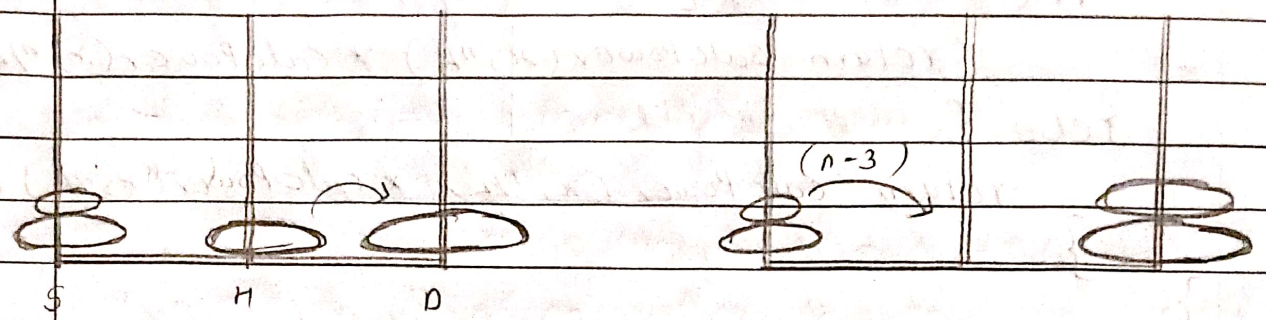
4) use only three tower Source, helper & destination.

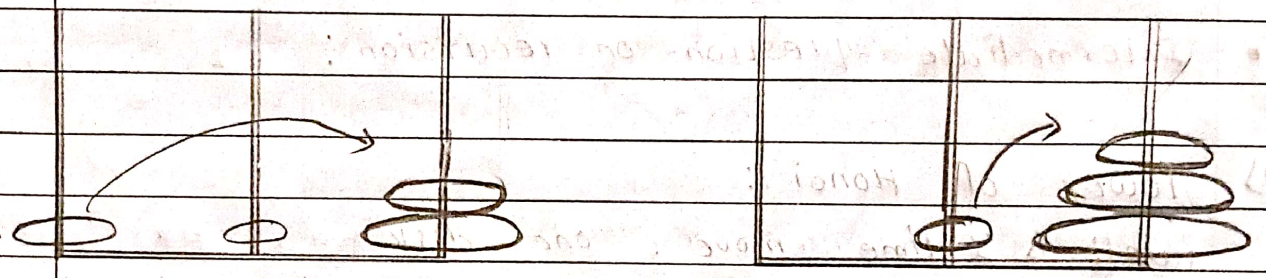


∵ work is divide in small part

∴ total n ring







for n ring => (n, source, helper, destination)

for (n-1) => (n-1, source, destination, helper)

for (n-2) => (n-1, helper, source, destination)

- take ring in rod → destination
- move ring from rod – source
- move ring but use another rod – helper

//code//

```
Public class recursion_of tower_of_hanoi{
    public static void towerOfHanoi(int n, String sor,
                                    string help, string dest){
      if(n==1){
          System.out.println("disc is move " + sor + "to" + dest);
          return;
      }
      towerOfHanoi(n-1, sor, dest, help);
      System.out.println("disc is move " + sor + "to" + dest);
      towerOfHanoi(n-1, help, sor, dest);
    }
    pubic static void main(String[] args){
        System.out.print("enter no. of disc : ");
        int n = sc.nextInt();
        String a, b, c;
        a = "Source"; b = "Helper"; c = "Destination";
        towerOfHanoi(n, a, b, c);
    }
}
```

o/p: enter the no. of disc : 3
disc is move Source to Destination
disc is move Source to Helper
disc is move Destination to Helper
disc is move Source to Destination
disc is move Helper to Source
disc is move Helper to Destination
disc is move Source to Destination

general formula of step : $2^n - 1$
$\therefore n \rightarrow$ no. of disc

$\therefore$ time complexity = $O(2^n)$.

Q 2) Print string in reverse :

- // method //

```
public static void reversString (String st, int index){
        if(index == 0){
                System.out.println(st.charAt(index));
                return;  }
        System.out.print(st.charAt(index));
        reversString(st, index - 1);
}
```

In main method ⇒ ~~reversString("onkar", s~~

```
String s = "onkar";
reversString(s, s.length()-1);
```

- time complexity depend on calling the recursion function and function called the string length time.
- hence time complexity depend on string length.

∴ time complexity = O(n)

Q 3) find the 1st & last occurance of an element in string.

eg: "abaachaefaah" element = 'a'.

index=0 (first occurance) index=10 (last occurance)

//code//

```
public static int first = -1; // initialise any -ve number
public static int last = -1; // ——
public static void occurenceOfEle(String st, int index, char ch){
        if( index == st.length()-1){
                System.out.println(first);
                System.out.println(last);
                return;
        }
        char currentChar = st.charAt(index);
```

```
        if(char == currentChar){
            if( first == -1){
                first = index; }
        }else {
                last = index;
            }
        }
        occurenceOfEle(st, index+1, ch);
    }
```

time complexity = O(n)

Q4) Check the given ~~sorte~~ array is sorted (strictly increase).

- eg: {1, 2, 3, 4}, {1, 3, 5, 9} ✓
  {1, 2, 2, 3}, {3, 2, 4, 5} ✗

//code//
```
public static boolean isSorted (int arr[], int index){
    if (index == arr.length-1){
            return true; }
    if (arr[index] < arr[index+1]){
            isSorted (arr, index+1);
    } else {
            return false; }
    return isSorted (arr, index+1);
}
```

//code// another way

```
                ↓ return type
public static boolean isSorted (int arr[], int idx){
    if (idx == arr.length-1){ return true; }
    if (arr[idx] < arr[idx+1]){
        return isSorted (arr, ~~ind~~ idx+1); }
    else { return false; }
}
```

return also write here →

// because any place return the function type //

time complexity = O(n) - (arr.length)

Q5) Move all 'x' in the end of the string.

//Code//

```
public static void moveAllX(String st, char ch, int idx, int count,
                            String newString){
        if(idx == st.length()) {
                System.out.print(newString);
                for(int i= 1; i≤count ; i++){
                        print(ch);      }
        } return; }
        if(ch==st.charAt(idx)){
                count++;
                moveAllX(st, ch, idx+1, count, newString);
        }else {
                newString += st.charAt(idx);
                moveAllX(st, ch, idx+1, count, newString);
        }
}
```

(part)

main method ⇒

```
String st = "onxxkxxar";
Char ch = 'x';
moveAllX(st, ch, 0, 0, " ");
```

time complexity = $O(n + n) = O(2n) = O(n)$

string length — loop for no. of x.

this part write also

```
if (idx == st.length()){
        for(int i=1; i≤count ; i++){
                newString += ch;
        }
        System.out.println(newString);
        return;
}
```

Q6) Remove duplicates in a string

eg: "abbccda" → "abcd"

```
//code// public static void removeDuplicate(String st, int idx,
                                        String newString){
        if( idx == Str.length()){
                System.out.println(newString);
                return;
        }
        Char currentChar = st.charAt(idx);
        if( map[currentChar - 'a'] ){
                removeDuplicate(st, idx+1, newString);
        }else {
                newString += currentChar;
                map[currentChar - 'a'] = true;
                removeDuplicate(st, idx+1, newString);
        }
}
```

→ public static boolean [] map = new boolean [26];
(this line is write to top of this code)

main Method ⇒ String st = "abbccda";
remove Duplicate(st, 0, " ");

time complexity = O(n) -(depend on string length
∴ Suppose length is n then called n time)

Imp

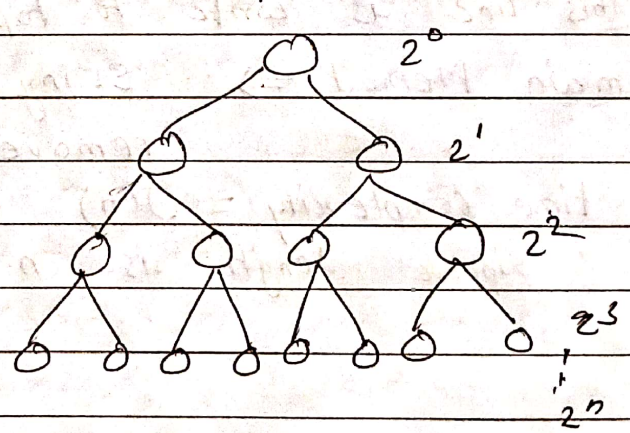
Q7) Print all the subsequences of a string

//code//

```
public static void subsequences(string st, int idx, string new){
    if( idx == st.length()) {
        System.out.println( new);
        return;
    }
    Char currentChar = st.CharAt(idx);
    // to be
    subsequences( st, idx+1, new + current-Char);
    // or not to be
    subsequences( st, idx+1, new);
}
public static void main(String [] args){
    String st = "abc";
    ~~Substrin~~
    subsequences(st, 0, " ");
```

o/p: abc, ab, ac, a, bc, b, c, ---- - blank } output

time complexity : $O(2^n)$

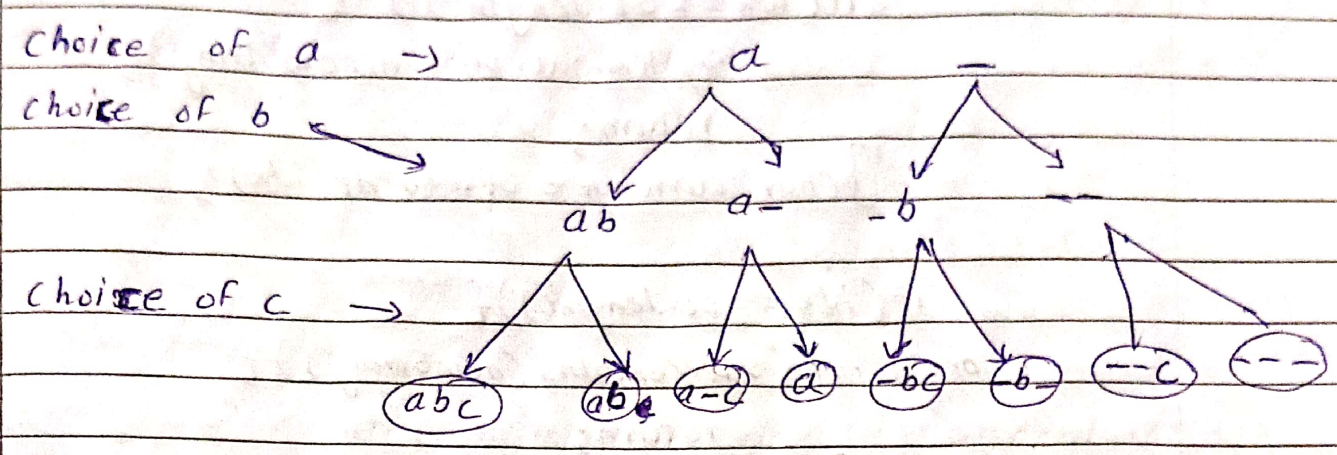


- this are represent the tree & each point are called the function (recursive tree).

DATE

Explaination: By using any string to make substring but sequence will not be change

- each character ~~co~~ choise present, they are add or not be add in any substrings.



⇒ abc, ab, ac, a, bc, b, c, -- ⇒ 8 subsequence

∴ In recursion called the two times because depend on character (choice) add or not add

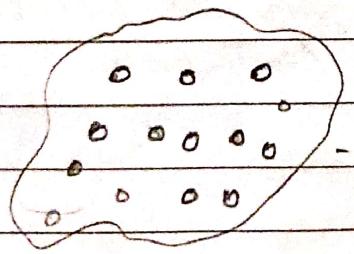
∴ called 1 : newstring + character
called 2 : newstring

Q8) Print all the unique subsequence of a string (not repeat same set of string).

~~Hashset~~ eg: "aaa" ⇒ aaa, aa, aa, aa, a, a, a
⇒ aaa, aa, a — without repeat

public ~~static~~ first import java.util.Hashset;

- In this code use hashset, they are use to all the value are store in the set with non ~~repeate~~ repeat



- hashset
- non repeate element store

//code//

```
import java.util.HashSet;
public class class_Name {
    public static void subSequence(String st, int idx,
                    String newstring, HashSet<String>){
        if(idx == st.length()) {
            System.out.println(newstring);
            return;
        }char currChar = st.charAt(idx);

        if(idx == st.length()){
            if (set.contains(newstring)) {
                return;
            } else {
                System.out.println(newstring);
                set.add(newstring);
                return;
            }
        }
        char currChar = st.charAt(idx);
        subsequence(st, idx+1, newstring + currChar, set);
        subSequence(st, idx+1, newstring, set);
    }
    public static void main(String [] args){
        String st = "aaa";
        HashSet <String> set = new HashSet<>();
        subSequence(st, 0, " ", set);
    }
}
```

DATE

Q 9) Print keypad combination

⊛ Operators & Binary number System :

1) operators : A symbol that can perform some operation.

eg: +, -, ×, ÷, %, ++, -- etc. (==, !=, <, >, <=).

- Arithmatic Operators

Binary	unary
+, -, *, /, %	++, --

← also called increament & decreament operator.

2) this operation is perform two number

this operation perform only one number

- Unary operators mostaly use in loops.
- there are two type of unary 1) Pre increament & post

	(++a) Pre increment	(a++) Post increment
-	fist change the value	- first use the value
-	then use the value	then change the value.
//code//	int a = 10;	int a = 10;
	int b = 0;	int b = 0;
	b = ++a;	b = a++;
	System.out.println(a);	System.out.println(a);
	System.out.println(b);	System.out.println(b);
	o/p : 11	o/p : 11
	11	10

- Relational operator

1) == → equal to
2) != → not equal
3) < → less than
4) > → great than

- logical operators

1) && (and) → T && T ⇒ T
2) || (OR) → F || F ⇒ F || T || F = T || F || T = T
3) ! (Not) → T → F & F → T

eg: a = 10, b = 2 ; a > b = T ; ! (a > b) = F

* Binary number System (Base 2)

- In this, use only two digit 0 & 1.
- Decimal digit → 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- All java code convert in binary then computer understant the code.

* decimal convert binary *

ex: 4

2	4	→ remainder 0
2	2	→ remainder 0
	1	→ remainder 1

$\therefore (4)_{10} = (100)_2$

decimal number ↑ binary number ↑

write remainder bottom to top ⇒ 100

eg: 15

2	15	→ 1
2	7	→ 1
2	3	→ 1
	1	

$(15)_{10} = (1111)_2$

* Binary convert into decimal *

eg: $(101)_2$

1 0 1 → This × This then add.

2^2 2^1 2^0

↓ ↓ ↓

$4 + 0 \times 2 + 1 \times 1 = 4 + 0 + 1 = 5$

$\therefore (101)_2 = (5)_{10}$

eg: (11101)

	1	1	1	0	1
multiply →	×	×	×	×	×
	2^4	2^3	2^2	2^1	2^0
equal to →	=	=	=	=	=

$16 + 8 + 4 + 0 + 1 = 29$

$\therefore (11101)_2 = (29)_{10}$

* other systems

- Octal – Base 8
- Hexadecimal number system – Base 16

– 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

– hexadecimal number system use to calculate the address of memory.

eg: address of array in memory.

* Bitwise Operators

– it is use to creat a short cut

– Bit manipulation is depend on it.

1) & (Bitwise and / Binary AND):

eg: A = 0101, B = 0110

1 – T
0 – F

$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \end{array} \quad 0100$$

$\therefore A \,\&\, B = 0100$,

2) | (Binary OR):

– $A | B = 0111$

$$\begin{array}{cccc} 0 & 1 & 0 & 1 \\ \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ 0 & 1 & 1 & 0 \end{array} \quad 0111$$

3) ^ (Binary XOR): (similar value → false, diff. v – T).

– like as negation of double implication.

$\therefore$ 0 1 0 1
↕ ↕ ↕ ↕
0 1 1 0 } 0011 $\therefore A \wedge B = 0011$

4) ~ (Binary one complement) : Symbol name - Tilde

$A = 0101 \Rightarrow \sim A = 1010$

$B = 0110 \Rightarrow \sim B = 1001$

5) << (Binary ~~Jest~~ Left shift):

$A = 0101$

... _ _ _ 0 1 0 1 _ _ _ ...

$\therefore A << 1 \Rightarrow$ ... _ _ _ 0 1 0 1 _ _ _ ...

$\therefore A << 1 \Rightarrow$ 1 _ _ 0 1 0 1 0 _ _ _

$\therefore$ finally $A << 1 = 1010$

$B = 0110$ I) _ _ 0 1 1 0 _ _ _

$B << 1 \rightarrow$ II _ 0 1 1 0 0 _ _ _

$B << 1 = 1100$

6) >> (Binary right shift):

$A = 0101$; $A >> 1 = 0010$

$B = 0110$; $A >> 1 = 0011$

* Assignment operator

$= \rightarrow A = B \Rightarrow A = B$

$+= \rightarrow A += B \Rightarrow A = A + B$

$-= \rightarrow A -= B \Rightarrow A = A - B$

$*= \rightarrow A *= B \Rightarrow A = A * B$

$/= \rightarrow A /= B \Rightarrow A = A / B$

DATE

* Bit Many Manipulation :

- they are four types
  1) Get bit - ~~Find~~ Check the nth position of bit (0 or 1)
  2) Set bit - Set the bit in n^{th} position (1)
  3) Clear bit - change the bit in 0
  4) updat bit - change all bit $1 \to 0$ & $0 \to 1$

- Bit Mask - apply the operation on bit like eg: get, set etc then change the bit this process called as bit mask

Q 1) Get the 3rd bit (position 2) of number n.
$n = (0101)$; Bit Mask = $1 << i$; operation = AND

→ apply Get check the 2nd position 0 and 1.

~~Bit~~ i = position - 2

0	1	0	1	0	1	0	1
~~4th~~	~~3rd~~	~~2nd~~	~~0th ← bit~~	3rd	2nd	1st	0th

$i = 2 \Rightarrow 1 \Rightarrow 3^{rd}$ bit

∴ 0 position ⇒ 1st bit

∴ 1 position ⇒ 2nd bit

∴ 3rd position ⇒ 4th bit

∴ Bit Mask = $1 << i$

$0001 << 2 = \underline{0100}$

given: 0101

∴ 0101 & 0100 = 0100

- In third position present 1

logic: $B = (1 \ \& \ A) \Rightarrow$ depend the value of A
A is 0 then value is 0 ; A is 1 value is 1
Similary logic applied on get, check the nth position bit is 0 & 1 then $(1 << i)$ 1 is move i^{th} position & apply operation AND with given example.

Same question but position: 3

i) $1 << 3$ = $0001 << 3$ = 1000

ii) $n = 0101$

$\therefore$ $1000 \,\&\, 0101 = 0000$

given bit is zero.

§ 2] Set bit; set the 2nd bit (position 1) of number n.

$n = 0101$.

$\therefore$ Bit Mask : $1 << i$

$\therefore$ operation : OR

$(0101)_2 = (5)_{10}$

$(0111)_2 = (7)_{10}$.

→ • $1 << i = 0001 << 1 \Rightarrow 0010$

• $n = 0101$; $0010 \mid \overset{0101}{\cancel{0100}} = 0111$

- using set bit replace 0 with one

• // code of Get Bit // (AND)

```
int n = 5;          // 0101
int position_of_bit = 2;
int BitMask = 1<< position_of_bit;   // 0001 → 0100
if((BitMask & n) == 0) {
        System.out.println("given Bit is 0");
} else {
        System.out.println("given Bit is 1");
}
```

• // code of Set Bit // (OR)

```
int n = 5
int pos = 1;
int BitMask = 1<< pos;
int newNum = BitMask | n;
System.out.println(newNum);
```

o/p: 7

DATE

Q 3) Clear Bit ; clear the 3rd Bit (position = 2) of a number n. (n = 0101).

Bit Mask = 1 << i

operation = AND with NOT of Bit Mask

→ 1<< 2 = 0001 << 2 = 0100

~(1<<2) = 1011

n = 0101 ; 0101 & ~~0100~~ 1011 = 0001

∴ $(5)_{10} = (1)_{10}$

// Code //

```
int n2 = 5;          // 0101
int pos = 2;
int BitMask = 1<<pos;      // 0001 → 0100
int newB = ~BitMask        // 1011
int ans = newB & n2;       // 1011 & 0101 → 0001
System.out.println(ans);   // 1  - in decimal
```

Q 4) Update Bit ; update 2nd bit (position 1) of number n to 1. (n = 0101).

for 0 ⇒ Bit Mask = 1<<i operation : AND with NOT of BM	For 1 : Bit Mask : 1<<i operation : OR

→ In these only extra apply the cond^n when convert 0 to 1 then use Set Bit & 1 to 0 use clear Bit.

- by user Input 'operator'

```
// code// Scanner sc = new Scanner(System.in);
int n3 = 5;
int pos1 = 1;
int BitMask3 = 1<<pos1;
int operator = sc.nextInt();
if (operator == 1){
        // use Set Bit //
    int newNum1 = BitMask3 | n3;
    System.out.println(newNum1);
} else {
        // use clear Bit //
    int neg = ~BitMask3;
    int newNum1 = neg & n3;
    System.out.println(newNum1);
}
```

Note: o/p in decimal number.

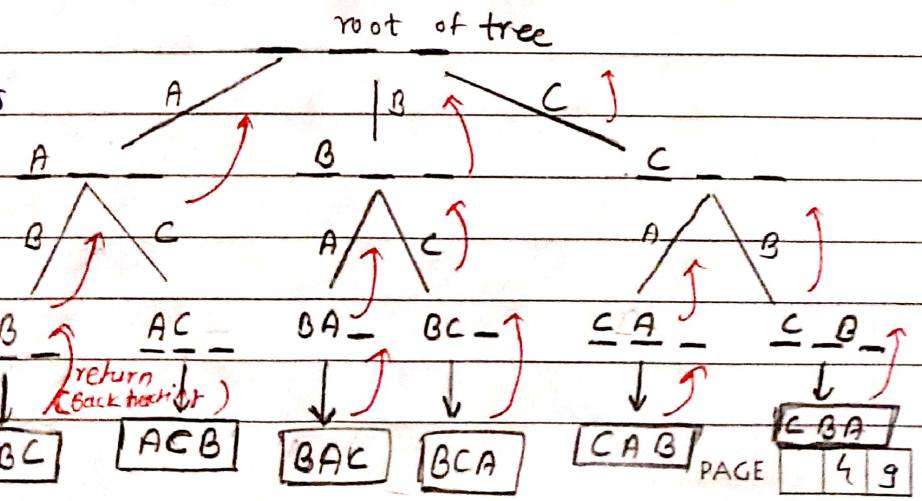
* Backtracking :

- It is like as permutation, arrange are all possible
- find all possible soln & use the one you want
- Back tracking is the backside work in the recursion

Q: Arrange possible arrangements of ABC.

→ total permutation = n! = 3! = 6

This structure like as tree.



// code //

```
Public static void printPermutation(String st, String Perm, int idx){
    if (st.length==0){
        System.out.println(perm);
        return;
    }
    for (int i=0; i< st.length; i++){
        char currChar = st.charAt(idx);
        String newst = st.subString(0,i) + st.subString(i+1);
        printpermutation(newst, perm+currChar, idx+1);
    }
}
```

main method ⇒ String st = "ABC";
printPermutation (st, " ", 0);

- time complexity = $O(n * n!)$ n – no. of steps
  n! – all possible combination, n! – time run programme.

9 N - queens

* Recursion problems

q 1) Print n numbers

// code //

a]

```
public static void printNum(int n, int m){
    if (n==m) {  System.out.println(n);
        return;
    } System.out.println(n);
    printNum(n+1, m);
}
```

eg: n=1, m=5 ⇒ o/p : 1, 2, 3, 4, 5.

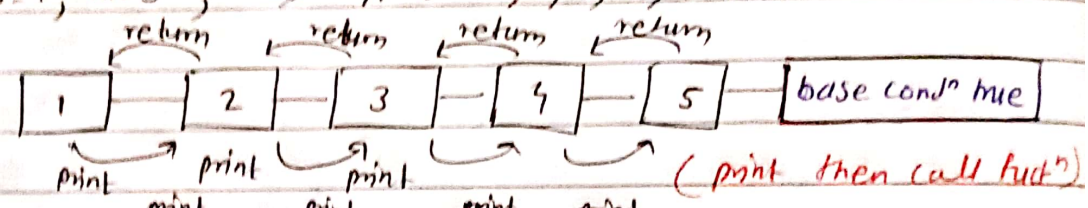
// logic change //

b]

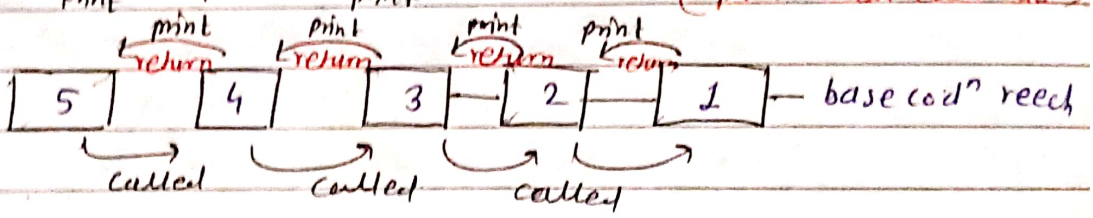
```
    if(n==m) {
        System.out.println(n);
        return;
    } System.out.
    printNum(n+1, m);
    System.out.println(n);
```

eg: n=1, m=5, ⇒ o/p : 5, 4, 3, 2, 1.

a]



b]



Note: In a recursion logic is write by considering assuming for step I & write code of the step II doing the recursion function automatic.

Q2] n^{th} fibonacci sequence number (leetcode)

→ 0 1 1 2 3 5 8 ....

0th 1st 2nd 3rd

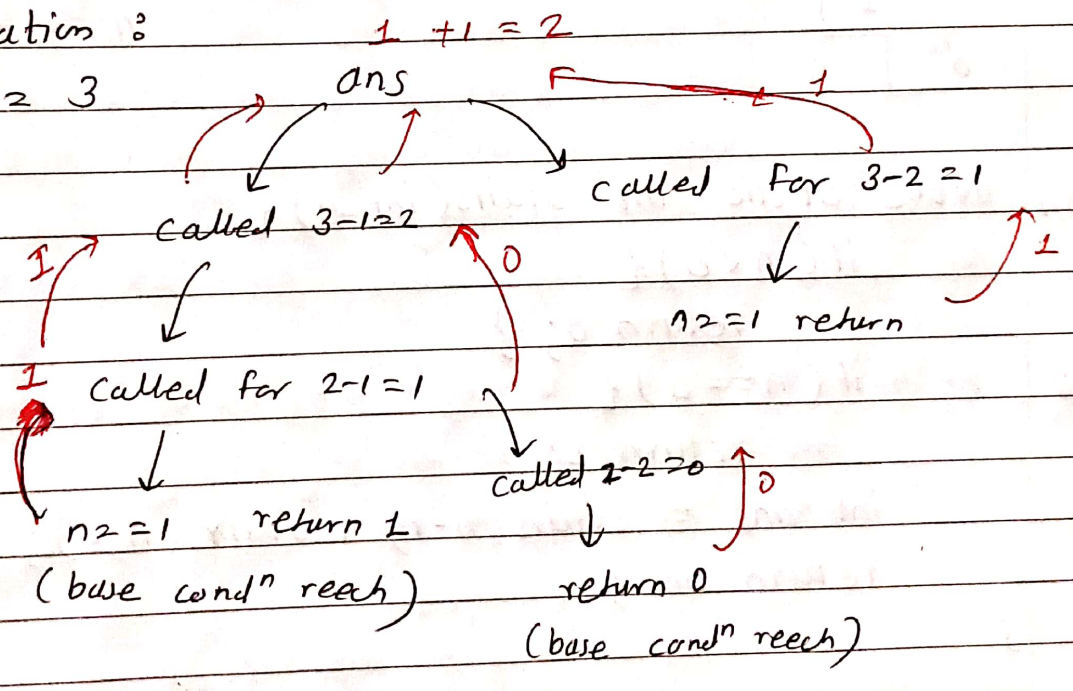
$f(0) = 0$, $f(1) = 1$, $f(2) = 1$ .. $f(3) = 2$

recursive function ∴ $f(n) = f(n-1) + f(n-2)$

//code//
```
public static int fibonacci(int n){
    if(n==0){
        return 0; }          // term is start with 0th.
    if(n==1){
        return 1; }
    int ans = fibonacci(n-1) + fibonacci(n-2);
    return ans;
}
```

Explaination :

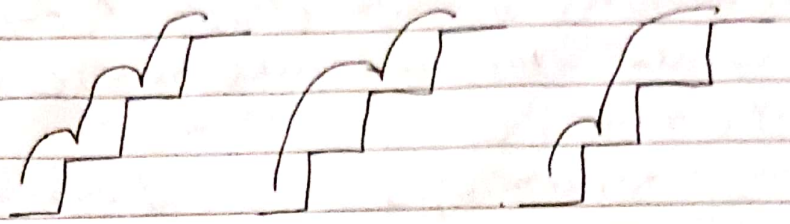
eg: n = 3



classmate

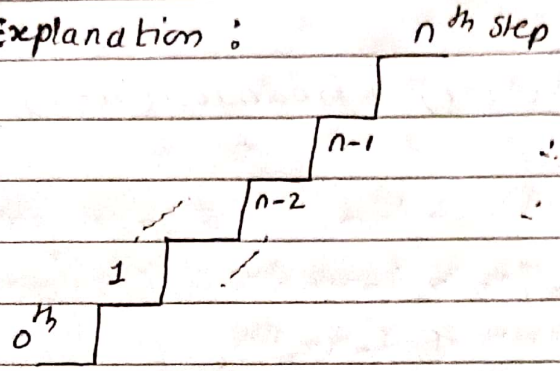
Q3] Counts ways to reach the Nth stairs by either climb one step or two steps. return the number of distict ways in which you can climb from 0th step to N^{th} step.

- eg: $n=3$



return 3

Explanation :



$\therefore f(n) = f(n-1) + f(n-2)$

$\therefore$ base case : $n<0 \rightarrow$ return 0;

$n==0 \rightarrow$ return 1;

```
//code// public static int stairs(int n){
        if(n<0){
            return 0; }
        if(n==0){
            return 1;
        int ans = stairs(n-1) + stairs(n-2);
        return ans;
    }
```

ex: ~~Sty~~ $n=4$ $\Rightarrow$ o/p : 5

Note: for a same logic climb 0th to n^{th} step by one step, two step & three step.

$f(n) = f(n-1) + f(n-2) + f(n-3)$

ex: 4
o/p: 7

} same as above code only add extra stairs(n-3)

DATE

Q4) Print the spelling of each digit by input any number

eg: 432 ⇒ four three two

```
//code// Static void wordOfDigit(int n){
        if(n==0){
            System return; }
        int a=n%10;
        n=(n-a)/10;
        div(n);
        System.out.print(arr[a]);
//code// public static void digitWord(int n){
        ~~if(n==0){~~
        String[] arr={"zero", "one", "two", "three", "four",
                "five", "six", "seven", "eight", "nine", "ten"};
        if(n==0){
            return; }
        int a = n%10;
        n = (n-a)/10;
        div(n);
        System.out.print(arr[a]+" ");
    }
```

* Binary Search

- Linear search : search any element in array and it is not required is sorted or not.
- Binary search : Binary search on only monotonic function (array is sorted)
  - Monotonic order - increasing / decreasing order

Step : 1st array is ~~s~~ sort then given element is compare with ~~middel~~ middle term in the array

- ~~elem~~ element < arr[middle] — go search only left side
- element > arr[middle] — search only right side.

0	1	2	3	4	5	6
1	2	3	4	5	6	7

length = 7 ; mid = (start + end)/2 ;

case 1 : element < arr[mid] ⇒ end = mid - 1

element > arr[mid] ⇒ start = mid + 1

element == arr[mid] ⇒ print (mid);

Start = 0 & end = arr.length - 1;

- this process is continuous for start == ends.

// code //

```
public static int binarySearch (int arr[] ; int element){
        int Start = 0 , end = arr.length - 1 ;
        int mid = start + ~~sta~~ (end - start) / 2 ;
        while (start ≤ end){
                if (element == arr[mid]){
                        return mid ;      }
                if (element < arr[mid]){
                        ~~return~~ end = mid - 1;
                else {
                        Start = mid + 1 ;  }
                mid = start + (end - start)/2 ;
        } retur -1 ;
}
```

eg: element = 6 ; arr = {1, 2, 3, 4, 5, 6, 7};

1	2	3	4	5	6	7
0	1	2	3	4	5	6

initially start = 0 ; end = arr.length - 1 = 7 - 1 = 6

mid = start + (end - start) / 2 ;

mid = 0 + (6/2) = 3

① ∴ element < arr[mid] ⟹ 6 < 4 ⟹ F

② ∴ element > arr[mid] ⟹ 6 > 4 ⟹ T

∴ start = mid + 1 = 3 + 1 = 4

mid = 4 + (6 - 4)/2 = 5

5	6	7

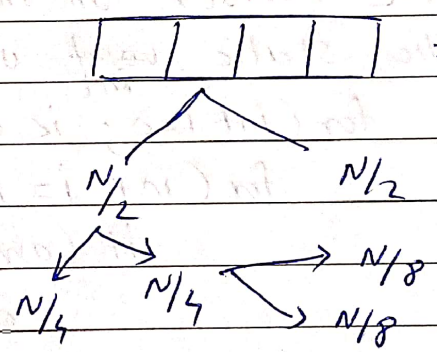
① ~~mid~~ arr[mid] == element ⟹ T

return mid ;

time complexity ⟹ ~~length~~ $\frac{}{2}$ ⟹ N size.

- initially $N = \frac{N}{2^0}$

2nd step ⟹ $\frac{N}{2^1}$; 3rd = $\frac{N}{2^2}$



∴ $\frac{N}{2^K} = 1$

$N = 2^K$ ⟹ $\log_2(N) = K$ ⟹ $O(\log N)$.

* Question on ~~Binary~~ array ~~Search~~ :

Q1) Search the pair, ~~wh~~ there sum is targeted number.

→ eg : arr = {1, 2, 3, 4, 5}

find the pair of ~~given~~ number there sum is 7

ans = 2, 5 & 3, 4 .

classmate

DATE

// code //

```
public static ~~void~~ int sumOfTarget (int[] arr, int target){
    int count = 0;
    for(int i=0; i<arr.length ; i++){
        for( int j= i+1 ; j<arr.length ; j++){
            if (arr[i] == target - arr[j]){
                System.out.println(i+", "+j);
                count++;
            }
        }
    }
    return count;
}
```

Q.2) Unique number in the array.

// code //

```
public static ~~void~~ int uniqueNum( int arr[]) {
    for (int i=0; i< arr.length ; i++) {
        for ( int j= i+1 ; j< arr.length ; j++) {
            if (arr[i] == arr[j] ){
                arr[i] = -1;
                arr[j] = -1; }
        }
    } int ans = -1;
    for( int i=0; i< arr.length ; i++){
        if ( arr[i] > 0){ ans = arr[i];
            ~~System.out.println(arr[i]+" is unique No.");~~
        }
    } return ans;
}
```

Integer.MIN_VALUE = ~~-27~~ -2147483648

DATE

Q 3) find the second largest element in array

//code//

```
public static int maxValue (int arr[]){
        int max = Integer.MIN_VALUE;
        for (int i=0; i< arr.length; i++){
                if ( arr[i] > max) {
                        max= arr[i];
                }
        }
        return max;
}
public static int secondMax (int arr[]){
        int max = maxValue (int arr[]);
        for (int i=0; i< arr.length; i++){
                if (arr[i] == max) {
                        ~~max~~ arr[i] = Integer.MIN_VALUE;
                }
        }
        int secondmaxvalue = maxValue (arr);
        return secondmaxvalue;
}
```

function

Q 4) first ~~reap~~ repeating element in array.

//code//

```
public static void firstRepeat (int arr[]){
        for (int i=0; i< arr.length; i++){
                for (int j=~~i+1~~; j< arr.length; j++){
                        if ( arr[i] == arr[j]){
                                return arr[i];
                        }
                }
        }
        return -1;
}
```

Note: Swapping without using tem variable

eg: a = 3, b = 9

∴ a = a+b; // a=12
∴ b = a-b; // b = 3
a = a-b // a=9

This is the logic of of swapping two number.

Note: 1) ~~logic~~ another logic of reverse of array.

eg: i = arr.length - 1
j = 0;
~~for (int i~~ while (i ≥ 0) {
newarr[j++] = arr[i--];
}

2) by using swapping

```
//code// public static void reverse(int arr[]){
        int j = arr.length-1;
        int i = 0;
        while (i<j){
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            j--; i++;
        }
        // code of print array by for loop //
}
```

Q5) Rotate the array

eg: arr[] = {1, 2, 3, 4, 5};
1 time rotate = {5, 1, 2, 3, 4};
2 time rotate = {4, 5, 1, 2, 3};
3 time rotate = {3, 4, 5, 1, 2};

//code//

```
public static void rotate (int arr[], int n) {
    int l = arr.length - 1;
    for (int i=0; i< n%l ; i++) {
        int temp = arr[l];
        for( int j = l ; j>0 ; j--) {
            arr[ j ] = arr[j-1] ;
            if ( j==1) {
                arr[0] = temp ;
            }
        }
    }
    // code of for print array //
}   // n is no. of rotation //
```

//code// Another method

```
public static void rotate2 (int arr[], int k) {
    int l = arr.length;
    int ans [] = new int [l];      // using extra space //
    int j = 0;
    int n = l-k;
    for (int i = n ; i < l ; i++) {
        ans[j++] = arr [i] ;
    }
    for (int p=0 ; p<n ; p++) {
        ans[k++] = arr [p] ;
    }
    // print array //
    for (int ele : ans) {
        System.out.print( ele + ", ");
    }
}
```

DATE

//Code// // Another method //

```
public static void reverse (int arr[] ; int i ; int j){
        while( i<j ) {
            ~~arr[i] = arr[j];~~
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++; j--;
        }
}
public static void rotate3 ( int arr[], int k ) {
        int j = arr.length;
        reverse ( arr, 0, j-k-1 );
        reverse ( arr, j-k, j-1 );
        reverse ( arr, 0, j-1 );
        for ( int ele : arr ) {
            System.out.print ( ele + " " );
        }
}
```

Q 5] Check more than one element present in the array.

→ 1) first create null array of size 10^5 or more

2) ~~In~~ these array, ~~assigne~~ called as frequency.

//code//

```
public static int [] frequency (int [] arr){
        int [] freq = new int [100000];
        for (int i=0; i< arr.length; i++){
            freq [arr[i]] ++;
        }
        return freq;
}
```

```
public static void checkElement (int arr[], int ele){
        int freq = frequency (arr);
        if ( freq[ele] > 0) {
                System.out.println(ele + " present");
        } else {
                System.out.println (ele + " Not present");
        }
}
```

* Two pointer : (In this use two point in the array)

- The array contains only zero & one

eg: arr = { 0, 1, 1, 0, 1, 0, 0 };

Q 7) Sort array consisting of only 0 & 1.

//code//

```
public static void sortZero-One ( int arr[]) {
        int count = 0;
        for ( int i = 0; i < arr.length ; i++) {
                if (arr[i] == 0) {
                        count++;
                }
        }
        for ( int j = 0 ; j < arr.length; j++) {
                if ( j < count ) {
                        arr [j] = 0;
                } else
                        arr[j] = 1;
                }
        }
}
```

//code 2// public static void sortZero_One2 (int arr[]){

```
int l = arr.length - 1;
int i = 0;
while (i < l) {
    if (arr[i] > arr[l]) {
        int temp = arr[i];
        arr[i] = arr[l];
        arr[l] = temp;   i++; l--;
    } if (arr[i] == 0){
        i++;  }
    if (arr[l] == 1) {
        l--;  }
}
}
```

Q 8] In given array, move all even integers at the beginning of the array followed by all the odd integer. The order of even & odd number not matt

sort array by parity

//code// public static void sortEvenOdd (int arr[]){

```
int right = 0; int left = arr.length - 1;
while (left < right){
    if (arr[left] % 2 == 0) {
        left ++;
    } else if (arr[right] % 2 == 0){
        int temp = arr[right];
        arr[right] = arr[left];
        arr[left] = temp;
        left ++; right --;
    } else {
        right --;
    }
}
}
```

eg: {1, 2, 3, 4, 5, 6};

o/p: {6, 2, 4, 3, 5, 1}

In these question for not write reverse method then change the logic ⇒ k = l-1 & replace k++ ⇒ k--

DATE 08 04 2024

Q 9) Given an integer array sorted in non-decreasing order, return an array of the square of each number sorted in non-decreasing order.

→ eg: ~~2, 3,~~ 5, 4, 3, 2, 1 → decreasing order

5, 4, 4, 3, 2, 2, 1 → non-decreasing order.

//code//

```
public static int [] sortOfSquareArray (int arr[]){
    int l = arr.length;
    int start = 0;
    int end = l-1; int k = 0;
    int ans[] = new int [l];
    while( start < end ) {
        if( Math.abs(arr[start]) > Math.abs(arr[end])){
            ans [k++] = arr[start] * arr[start];
            start ++;
        } else {
            ans [k++] = arr[end] * arr[end];
            end --;
        }
    }
    return ans;
}
public static void main(String [] args){
    int [] arr = {-5, -2, 2, 3, 4};
    int [] arr2 = sortOfSquareArray(arr);
    for ( int ele : arr2){
        System.out.print(ele + " ");
    }
}
```

o/p : 25, 16, 9, 4, 4 — decreasing order

Correction : write the reverse code for the array in sortOfSquarArray() method at bottom

classmate

DATE

Q.9 Check array is monotonic (Leetcode)

//Code//
```
public static boolean isMonotonic(int [] arr){
        boolean increasing = true;
        boolean decreasing = true;
        for( int i=1 ; i<arr.length ; i++){
            if(arr[i-1] < arr[i]){
                decreasing = false;
            }else if (arr[i-1] > arr[i]){
                increasing = false;
            }
        }
        return increasing || decreasing;
}
```

Q 10) Prefix sum of array

⇒ eg: arr = {2, 4, 5, 6, 8}; ⇒ {2, 6, 11, 17, 25};

//Logic//
```
public static void preffixSum( int [] arr){
        int j = 0;
        for( int i=0 ; i<arr.length ; i++){
            arr[i] = arr[i] + j;
            j = arr[i];
        }
}
```

Q 11) Sum of m to n index number in array

Note: ~~arry~~ array is 1 index base

⇒ Normally array is zero index base but in these question by user input assumed array start 1st index.

i.e. i is index of array ; m from user

i = m-1; i < n.

DATE

//logic//
```
public static void prefixSum2(int arr[], int m, int n){
        if (m ≤ n){
            int j=0;
            for(int i=m-1; i<n; i++){
                arr[i] = arr[i] + j;
                j = arr[i];
            }
            System.out.println(j);
        }
        else{
            int j=0;
            for(int i=n-1; i<m; i++){
                arr[i] = arr[i] + j;
                j = arr[i];
            }
            System.out.println(j);
        }   // write code to print array //
}
```

Q 12) Suffix sum of arry.

→ Sum of element by ~~bee~~ ending of the array.

eg: arr = {1, 2, 3, 4, 5}; ⟹ {15, 14, 12, 9, 5};

//logic//
```
public static void suffixSum(int arr[]){
    int l = arr.length-1;
    int j = 0;
    for(int i=l; i>0; i--){
        arr[i] = arr[i] + j;
        j = arr[i];
    }
    // print array by using for-each loop//
}
```

Q 13) Check if we can partition into subarrays with equal sum. (equal partition).

⟹ hint : equate prefix & suffix sum.

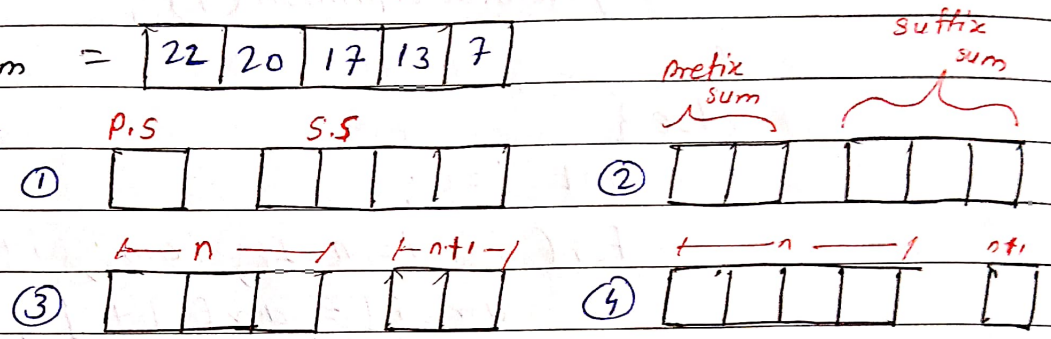
Explaination :

eg : arr = | 2 | 3 | 4 | 6 | 7 |

Prefix sum = | 2 | 5 | 9 | 15 | 22 |

suffix sum = | 22 | 20 | 17 | 13 | 7 |

Possibilities



∴ Preffix sum up to 'n' & suffix sum up to n-1

equate ⇒ P.S[n] == S.S[n-1].

2nd logic : | 2 | 5 | 9 | 15 | 22 | Total sum = j

j↑ cond^n ⇒ arr[i] = total − arr[i]

~~j = arr[~~

| 2 | 5 | | 9 | 15 | 22 |

sum → arr[1] j = arr[1] (suffix sum).

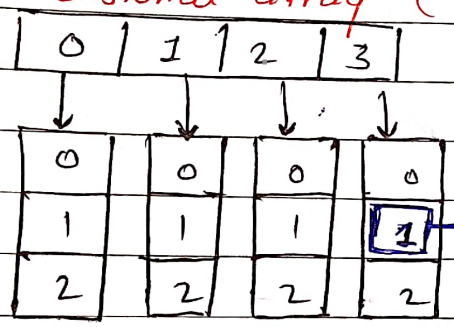
```
//code//  public static void prefixSuffixSum2(int arr[]){
            int l = arr.length;
            int j = 0;
            for(int i=0; i<l; i++){
                arr[i] = arr[i] + j;
                j = arr[i];
            }
```

```
for( int k=0 ; k< l. ; k++){
        if ( arr[k] == j - arr[k] ){
                System.out.print("equal : 1 to " + (k+1));
                System.out.println(" and " + (k+2) + " to " + l);
        }
    }
}
```

Note : // another method code in drive //.
// method name is "prefixSuffixSum" in class //.

(*) Two - Dimiensional array (Multidimen tion array) ;

- arr =



- array is present in array by each ~~ele~~ index.

representation :

rows \ columns	0	1	2	3
0	0,0	0,1	0,2	0,3
1	1,0	1,1	1,2	1,3
2	2,0	2,1	2,2	2,3
3	3,0	3,1	3,2	3,3

∴ row = m
∴ column = n
i.e arr [m] [n].

declaration : int [][] arr = new int[m][n];

classmate

Q 1) addition of two matrix by input.

//code//

```
public static void print2DArray (int arr[][]){
        for (int i=0; i<arr.length; i++){
                for (int j=0; j< arr[i].length; j++){
                        System.out.print (arr[i][j] +" ");
                }
                System.out.println();
        }
}
public static int[][] input (int row, int col){
        Scanner sc = new Scanner (System.in);
        int arr[][] = new int [row][col];
        System.out.println("Enter the element : ");
        for (int i=0; i< row; i++){
                for ( int j=0; j< col; j++){
                        arr[i][j]= sc.nextInt();
                }
        }
        return arr;
}
public static void addTwoMatrix ( int nums[][], int r1,
                int c1, int nums2[][], int r2, int c2){
        if ( r1 ≠ r2 || c1 ≠ c2 ){
                System.out.println(" addition is not possible");
                return;
        }
        int [][] sum = new int[r1][c1];
        for ( int i=0; i< nums1.length; i++){
                for (int j=0; j< nums1[i].length; j++){
                        sum[i][j]= nums1[i][j] + nums2[i][j];
                }
        }
```

```
        System.out.println(" The addition of two matrix");
        print2DArray (sum);
}

public static void main (String[] args){
        // r1, c1, r2, c2 are input from user //
        int arr1[][] = input (r1, c1);
        int arr2[][] = input (r2, c2);
        addTwoMatrix (int arr1, r1, c1, arr2, r2, c2);
}
```

Q 2) multiplication of two matrix.

// code //

```
public static void mulOfTwoMatrix (Int nums1[][]; int r1, int c1,
                                   int nums2[][]; int r2, int c2) {
        if (c1 ≠ r2) {
                System.out.println("Multiplication not possible");
                return;
        }
        int mul[][] = new int[r1][c2];
        for (int i=0; i<r1; i++) {
                for (int j=0; j<c2; j++) {
                        for (int k=0; k<c1; k++) {
                                mul[i][j] += (nums1[i][k] * nums2[k][j]);
                        }
                }
        }
        System.out.println (" The multiplication of two matrix");
        print2DArray (mul);
}
```

Q 3) transpose of matrix

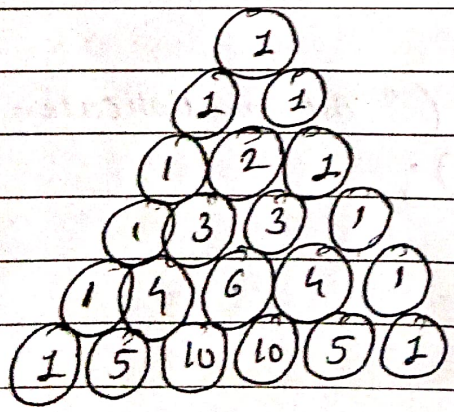
//code//
```
public static void transposeMat (int arr[][]; int r, int c){
        int [][] arr2 = new int [c][r];
        for(int i=0; i<c ; i++){
            for ( int j=0; j<r ; j++){
                arr2[i][j]= arr[j][i];
            }
        }
    }
```

Q 4) Rotate matrix in 90 degree in clockwise.

//code//
```
public static void rotate_90_Degree(int arr[][]; int r,
                                    int c) {
        int arr2[][] = new int[c][r];
        for (int i=0; i<c ; i++) {
            int k=r-1;
            for ( int j=0 ; j<r ; j++ ) {
                arr2[i][j] = arr[k--][i];
            }
        }
        // print array 2 //
    }
```

Q 5) Creat pascal triangle by using 2D array.

/ →



1				
1	1			
1	2	1		
1	3	3	1	
1	4	6	4	1

DATE

```
//code//  public static int [][] pascalTriangle (int n) {
            int [][] arr = new int [n][];
            for (int i=0 ; i< n ; i++) {
                int arr[i] = new int [i+1];
                arr[i][0] = arr[i][i] = 1;
                for ( int j = 1 ; j < i ; j++) {
                    arr[i][j] = arr[i-1][j-1] + arr[i-1][j];
                }
            }
            return arr;
        }
```

new topic start : Arraylist

DATE 17 04 2024

```
            for(int i= bottomRow; i>= topRow ; i--){
                System.out.print (arr[i][leftCol]+" ");
                totalEle ++;
            }
            leftCol++;
        }
    }
```

//Note: In given code correction, write the cond^n (totalEle < r*c) in all loops by &&

eg: for(--- ; i<= --- && totalEle < r*c ; i++);

this position write in all loop.

* Array list class :

Syntax: ArrayList <type> = new ArrayList<>();

type = Integer, String, Float, Boolean, Short etc

ex: ArrayList<Integer> = new ArrayList<>();

* Advantage : size is variable, not ~~definit~~ definite

- In arraylist store primitive data type & object
- array stored in a heap
- ArrayList <Integer> l1 = new ArrayList<>();

ex: l1 → | 2 | 3 | 4 | 5 |

* methods of Arraylist

1) for add element in arraylist ⇒ l1.add(n,m)

n = index , m - element

ex: l1.add(10) ⇒ | 2 | 3 | 4 | 5 | 10 |

l1.add(1,10) ⇒ | 2 | 10 | 3 | 4 | 5 |

DATE

Note: first → import java.util. ArrayList;

2) Print Arraylist : System.out.println(l1)
o/p : [2, 3, 4, 5]

3) Print any specific element : l1.get();
System.out.println(l1.get(index-no.));

4) delete any element : l1.remove(index_num);

5) Check the size of Array List : l1.size();

6) Iteration of ArrayList by using for loop
```
for (int i=0; i< l1.size(); i++) {
    System.out.println(l1.get(i));
}
```

7) add another Array List : l1.addAll(l2);

8) Check the element present in the Array List:
l1. contains(element);

9) Check the index number of element :
l1.indexOf(element);

Note: also allow ⇒ ArrayList l2 = new ArrayList
ex: l2.add("onkar");
l2.add(23);
l2.add(true);
print(l2)
o/p: [onkar, 23, true]

In this ArrayList add all type of primitive data type

DATE

Questions on Arraylist

Q 1) Print revers of Array List :

→ use two pointer method

//code//
```
public static void revers(ArrayList<Integer> l1){
    int i=0, j=l1.size()-1;
    while(i<j){
        Integer temp = new Integer(l1.get(i));
        l1.set(i, l1.get(j));
        l1.set(j, temp);
        i++; j--;
    }
    System.out.println(l1);
}
```

//Note// This question also solve by inbuilt function

first import java.util.Collections;

Collections.reverse(list); // list ⇒ l1 //

* Inbuilt function :

1) Collections.sort(list); → increasing order

~~2) Collections.reversOrder()~~

2) Collections.sort(list, Collections.reverseOrder()); → decreasing

3) Collections.reverse(list);

* Linked List

- The data store combinelly connect to each other.
- difference betn array, ArrayList & Linked List

Array : In array, definied initially size is fixed then add extra element over the size, it is not possible

eg: arr[] = | 1 | 2 | 3 | — size 3

adding in | 4 | — not allow

// This problem solve by ArrayList //

ArrayList : They are allow to add extra element after adding element then ArrayList can be resize automatically.

- In this not madnatory to pass the size by defined initially.

ex: l1 → 4 — 5 — 6 — 7

l1.add(1, 10); then

4 — 10 — 5 — 6 — 7

// but in case of ArrayList size is 2000-3000 or much greater the time complexity is increase because after adding element betn two element push elements one by one like this

creating blank memory

l1.add(1,10) → 4 — 5 — 6 — 7 — [ ]

then, push element then

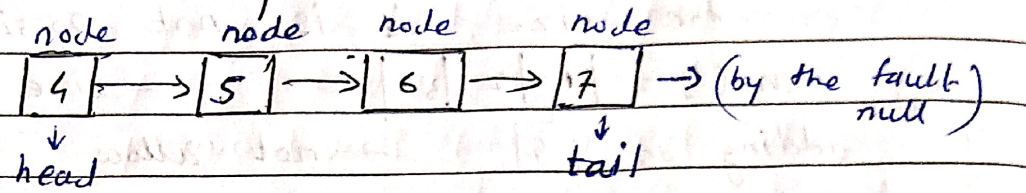
4 — 10 — 5 — 6 — 7

// This ~~bto~~ problem solve by LinkedList //

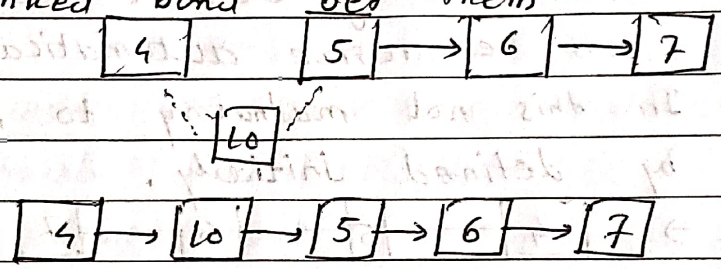
LinkedList : all the feature of Arraylist present in the LinkedList, but it is work to easy add element betn any two element.

- In LinkedList Node are connected to each other by Node address

ex:



address [5] this node store in [4] node. by adding 10 betn 4 & 5 then break the linked bond betn them



Note: all the methods in Arraylist are valid for Linked List.

- less time complexity for insert element.

* Creating Linked list.

```
//code//  public static void main(String[] args){
            Node a = new Node(5);
            Node b = new Node(7);
            Node c = new Node(9);
            Node d = new Node(10);
```

- first create a Node class
- create objects, like integer, string etc.
- Node next; in a class
- In main class create like this
  Node obj = new Node();
- also creating constructor in class then also write as, → Node obj = new Node (argument);

for example: create the linkedList;

```
//code// public class LinkedList_01 {
            public static class Node {
                int data;
                Node next;
                Node (int data) {        // constructor //
                    this.data = data;
                }
            }
            public static void main (String [] args) {
                Node a = new Node (5);
                Node b = new Node (6);
                Node c = new Node (7);
                Node d = new Node (8);
            // this are creating like this 5 6 7 8
                not linked //
                a.next = b;        // 5→6 7 8
            // a is linked with b; that means
                address of b stored in Node a //
                b.next = c;        // 5→6→7 8 //
                c.next = d;        // 5→6→7→8 //

            // It is print the following //
```

```
System.out.println(a.data);
System.out.println(b.data);
System.out.println(c.data);
                   (d.data);
```

o/p: 5, 6, 7, 8

```
// only use head //
System.out.println(a.data);
System.out.println(a.next.data);
System.out.println(a.next.next.data);
System.out.println(a.next.next.next.data);
```

o/p: 5, 6, 7, 8

```
    }
}
```

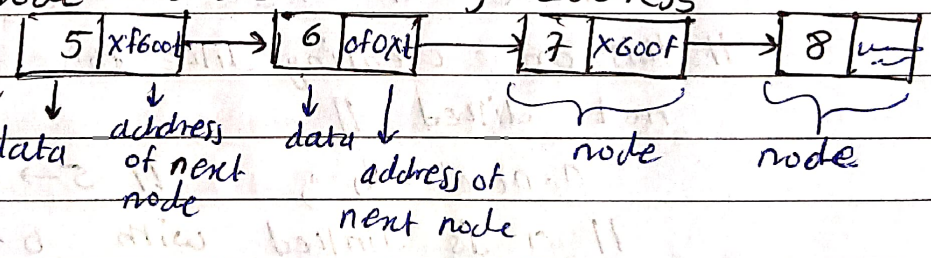
* Types of Linked List :

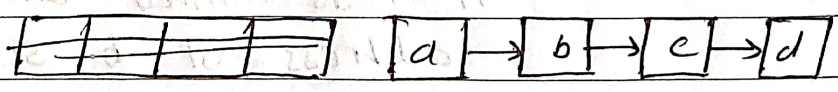
1) Singly Linked List
2) Doubly Linked List
3) Circular Linked List

1) Singly LinkedList

- each node stored data & address

ex:



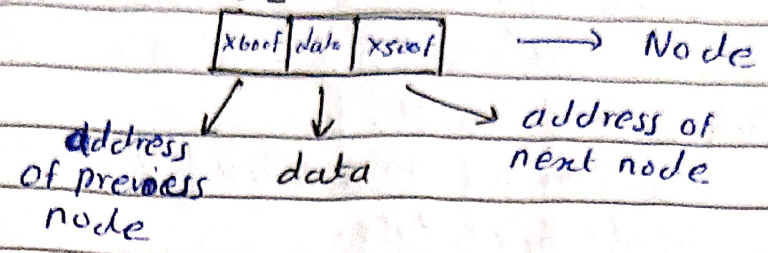
- LinkedList → 

by using 'b' node to find the c & D node but not find a node or data by using node 'b'. it is not allow.

2) Doubly linked list :

- The node stored the address of previous ~~previes~~ node & next node

ex:



3) Circular LinkedList :

- In this linked list not present tain, the last node is stored the address of head.

Q. Print LinkedList by using for loop.

```
//code//  Node temp = a;
            for(int i=
            public static void display(Node a){
                if(a=p=null
                if(a=+
            public istatic
//code//  public static void display (Node head){
                while(head != null){
                    System.out.print(head.data);
                    head = head.next;
                }
```

O/P : 5 6 7 8 → after calling.

// by using recursive function //.

```
//code// public static void display2(Node head){
            if(head == null){
                return; }
            System.out.prints(head.data+" ");
            display2(head.next);
        }
```

// Note // public Static void display (Node head){
while (head ≠ null){
System.out.print(head.data + " ");
head = head.next;
} while (head ≠ null) {
System.out.print (head.data+" ");
head = head.data;
}

In main function → display (a);
O/p: 5 6 7 8

- because in first while loop print all data up to the value of head is null, then loop is end.
- after first loop running second while loop but value of head is null, hence this loop is not run.

Q. Length of a list.

// code // public Static void lengthOfNode (Node head, int n){
if (head == null){
System.out.println("The length is :"+n);
return;
}
lengthOfNode (head.next , n+1);
}

DATE

* Create linkedlist by user defined data structure

```
//code// public class userDefineStructure {
        Node head = null;
        Node tail = null;
O(1) →  void addNode(int val){                    // method //
            Node temp = new Node(val);
            if( ~~tem~~ head == null){
                head = temp;
            } else {
                tain.next = temp
            } tail = temp;
        }
        void display () {                         // method //
            Node temp = head;
            while (temp ≠ null) {
                System.out.print( temp.data + " ");
                temp = tem.next;
            } System.out.println();
        }
O(n) →  ~~void~~ int size() {
            int count = 0; Node temp = head;
            while ( temp ≠ null) {
                count ++;
                temp = temp.next;
            }
            ~~System.out.println(count);~~ return count;
        }
```

O(1) →

```
void addFront (int val) {
    Node temp = new Node(val);
    if( head == null){
        head = temp;
    }else {
        temp.next = head;
    }
    head= temp;
}
```

O(n) →

```
void addNodeAt (int idx, int ele) {
    Node temp1= head;
    Node temp2= new Node(ele);
    if( idx ==0) {
        addFront (ele);
    } if( idx == size()) {
        addNode (ele);
    } for( int i= 0; i< idx-1; i++) {
        temp1 = temp1.next;
    }
    temp2.next = temp1.next;
    temp1.next = temp2;
}
```

O(n) →

```
void getNode (int idx) {
    Node temp = head;
    for( int i=0; i< idx; i++) {
        temp= temp.next;
    }
    System.out.println( temp.data );
}
```

DATE

// only head of list given & method is outside of linkedlist class & question is add element in tail //

//code// public static void insertAtEnd (Node head, int ele){

```
        Node temp = head;
        Node t = new Node (ele);
        while ( temp.next ≠ null){
                temp = temp.next;
        }
        temp.next = t;
}
```

// delete the element from list //
// this method write in linkedlist class //

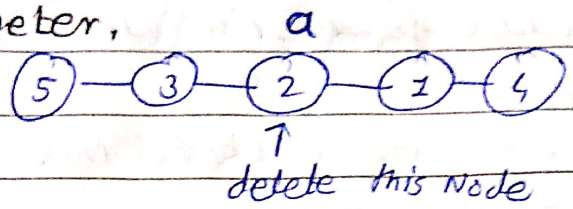
```
void deletAt( int idx ){
        Node temp = head;
        if( idx == 0){
                head = head.next;
                return;
        } for ( int i=0; i< idx-1; i++){
                temp = temp.next;
        }
        temp.next = temp.next.next;
        tail = temp;
}
```

* Interview question

Q 1) Can we delete the node given the node itself as parameter.

leetcode

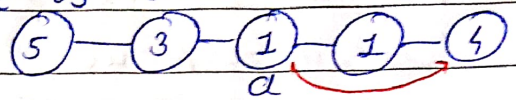
→ ex:



logic:

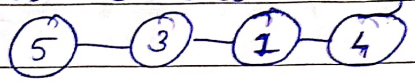
~~first, which node is delete~~ d.data = d.next.data.

new list →



logic ⇒ d.next = d.next.next;

new list →

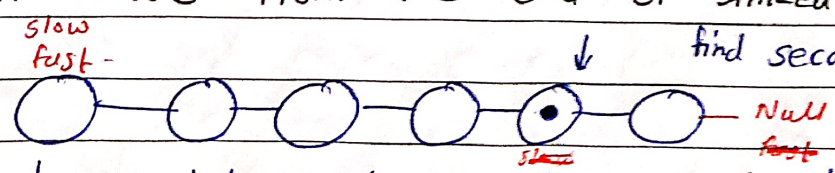



```
//code// public class ListNode{
            int val;
            ListNode next;
            ListNode (int x){
                val = x;
            }
        }
        class Solution {
            public void deleteNode(ListNode node){
                node.val = node.next.data;
                node.next = node.next.next;
            }
        }
```

Note: This logic not valid for given node is last node.

Q 2) finding N^{th} Node from the end of linked List

→ ex:



create two pointe slow & fast and initialise by head.

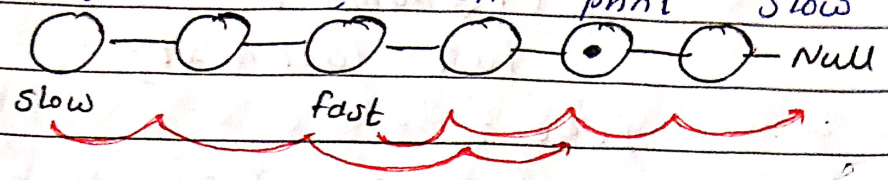
∴ fast is move next node by n times.

(n is given number, which can find)

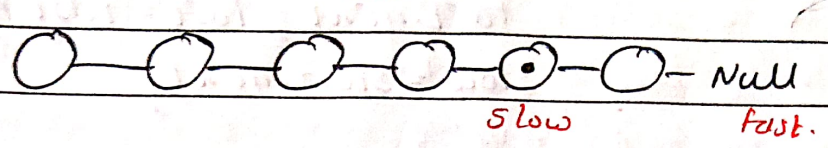
Note: First create Node class.

DATE 24 04 2024

then move slow & fast in next node simultaneously up to fast == null; then print slow node data.



then



//code//
```
public static Node NthNode(Node head, int num){
    Node slow = head;
    Node fast = head;
    ~~int~~ si
    for(int i=1; fast ≠ null; i++){
        if (i > num) {
            fast = fast.next;
            slow = slow.next;
        } else {
            fast = fast.next;
        }
    }
    return slow;
}
```

In main function → Node node = NthNode(a, N);
System.out.println(node.data);

Q3) delete ~~the~~ node from the end of the list.
- Same logic of previes question;

//code//
```
public static ~~void~~ Node deleteNodeFromEnd(Node head, int num)
{
    Node slow = head;
    Node fast = head;
    int i = 1;
```

```
while (fast.next ≠ null){
        if (i ≤ num) {
            fast = fast.next;
        }else {
            fast.next; fast = fast.next;
            slow = slow.next;
        }
        i++;
    }
    slow.next = slow.next.next;    return head;
}
```

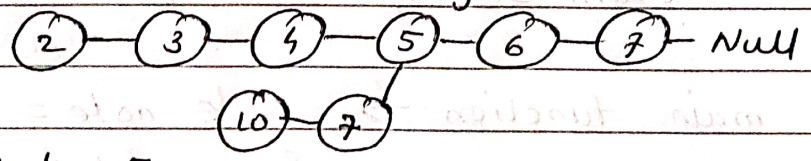
// code// This code is write in upper square.

```
if (slow == head) {
    head = head.next;
    return head;
}
```

→ In main function : a = deleteNodeFromEnd(a, N);
display (a);

Q. 4 Intersection of two list & return this Node

→ eg



intersecting Node 5

Step : 1) find size of this two list

2) compare this size

3) which size is greater, this list head node move (m-n) to next node

m - large size, n - small size.

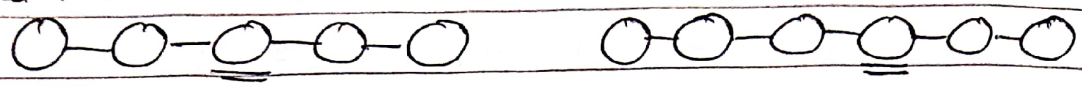
4) then move node in next node simultaneously

5) then two node is equal then return this node.

DATE

```
//node// public static Node intersectionOfTwoList(Node head1, Node head2){
        int size1 = 0; int size2 = 0;
        Node temp1 = head1; Node temp2 = head2;
        while (temp1 ≠ null){
                size1++;
                temp1 = temp1.next;
        }
        while (temp2 ≠ null){
                size2++;
                temp2 = temp2.next;
        }
        temp1 = head1; temp2 = head2;
        if (size1 > size2){
                for(int i=1; i≤ size1-size2; i++){
                        temp1 = temp1.next;
                }
        } else {
                for(int i=1; i≤ size2-size1; i++){
                        temp2 = temp2.next;
                }
        }
        while(temp1 ≠ temp2){
                temp1 = temp1.next;
                temp2 = temp2.next;
        }
        return temp2;
}
```

Q 5) ~~fin~~ find middle term in the list ?

→ 

//code//
```
public static Node middleNode(Node head) {
    Node slow = head;
    Node fast = head;
    while( fast ≠ null) {
        if( fast.next = null) {
            return slow;
        }
        slow=slow.next;
        fast = ~~fast.next~~; fast.next.next;
    }
    return slow;
}
```

Q 6) Check the list is cyclic or not return true otherwise return false

→ eg:

//code//
```
public static ~~void~~ boolean cycliclist(Node head) {
    if(head ==null) {
        return false;
    } if( head.next == null) {
        return false;
    }
    Node slow = head;
    Node fast = head;
    while ( fast ≠null) {
        if( slow==null) {
            return false;
```

```
            slow = slow.next;
            if ( fast.next == null ) {
                return false;
            }
            fast = fast.next.next;
            if ( fast == slow ) {
                return true;
            }
        }
        return false;
    }
```

Q7) detect Node which create cycle, return this node otherwise return null

eg:

//code//

```
public static Node deleteCyclicNode (Node head) {
    if( head == null || head.next == null ) {
        return null;
    }
    Node slow = head;
    Node fast = head;
    while ( fast ≠ null ) {
        slow = slow.next;
        if ( fast.next == null) {
            return null;
        }
        fast = fast.next.next;
        if ( fast == slow ) {
            break;
        }
    }
```

```
        if( fast == null) {
                return null;
        }
        Node temp = head;
        while ( temp ≠ slow) {
                temp = temp.next;
                slow = slow.next;
        }
        return slow;
}
```

Note: binary algorithms is not feasible to implement in a linked List.

Q8) Merge two sorted list

eg: l1 : 1 — 3 — 5 — 7 — null

l2 : 2 — 4 — 6 — 8 — null

merge → 1 — 2 — 3 — 4 — 5 — 6 — 7 — 8 — null

Leetcode

//code//

```
public static Node mergeTwoList (Node head1, Node head2){
        Node temp1 = head1
        Node temp2 = head2;
        Node head = new Node(7);
        Node temp = head;
        while ( temp1 ≠ null && temp2 ≠ null) {
                if( temp1.val < temp2.val ) {
                        ~~tem~~ Node a = new Node (temp1.val);
                        temp.next = a
                        temp = a
                        temp1 = temp1.next;
                }
```

```
15.         else {
14                  Node a = new Node (temp2.val);
15                  temp.next = a;
16                  temp = a
17                  temp2 = temp2.next;
18          }
19      }
20      if(temp1 == null){
21              temp.next = temp2;
22      } else {
23              temp.next = temp1;
24      }
25      return head.next;
26. }
```

// code is also write like given //
// line no- 6 to no.- 18 //

```
while( temp1 ≠ null && temp2 ≠ null){
        if( temp1.val < temp2.val){
                temp.next = temp1;
                temp = temp1;
                temp1 = temp1.next;
        }
        else{
                temp.next = temp2;
                temp = temp2;
                temp2 = temp2.next;
        }
```

99) Arrange odd-even number in the LinkedList,
Note: arrange odd numbers 1st then arrange all even numbers.

→ eg: 2 — 4 — 6 — 3 — 8 — 5 — null

Ans: 3 — 5 — 2 — 4 — 6 — 8 — null

//code//

```
public static Node oddEvenNumList(Node head){
        Node temp = head;
        Node t2 = new Node(2);
        Node even = t2;
        Node t3 = new Node(1);
        Node odd = t3;
        while (temp ≠ null){
                if( temp.data % 2 ==0 ){
                        even.next = temp;
                        even = temp;
                } else {
                        odd.next = temp;
                        odd = temp;
                }
                temp = temp.next;
        }
        even.next = null;
        odd.next = t2.next;
        return t3.next;
}
```

In main function {

```
        Node temp = oddEvenNumList(A);
        display(temp);
}
```

LeetCode (medium);

DATE

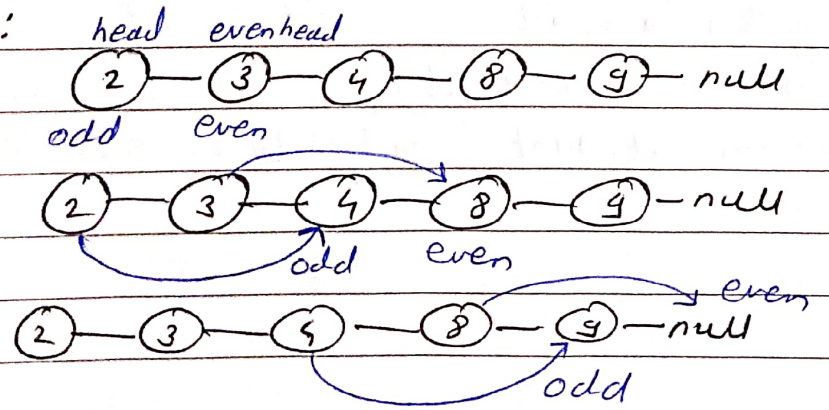
Q 10) Arrange the number based on even odd index of LinkedList, arrange odd index based number first.

→ eg: I1: 2 — 3 — 5 — 4 — 8 — 9 — null

Ans: 2 — 5 — 8 — 3 — 4 — 9 — null

```
//code// public static Node oddEvenIdxeList(Node temp){
        if( head ==null){
                return head;
        }
        Node odd = head;
        Node evenhead = head.next;
        Node even = evenhead;
        while (odd ≠ null && odd.next ≠null && even.next
                                                ≠null ) {
                odd.next = odd.next.next;
                odd = odd.next;
                even.next = even.next.next;
                even = even.next;
        }
        odd.next = evenhead;
        return head;
}
```

working:



In the while loop cond^n is reach then connect odd list in evenhead and return head.

space complexity = O(1) ; time complexity = O(n)

Q. 11) delete duplicate from linkedli sorted linked list.

→ ex: 1 - 1 - 2 - 2 - 3 - 3 - 4 - 4 - null

Ans 1 - 2 - 3 - 4 - null

//code//

```
public static Node deletDuplicates (Node head){
        Node temp = head;
        while( temp ≠ null && temp.next ≠ null){
                if(temp.data = temp.next.data){
                        temp.next = temp.next.next;
                }else{
                        temp= temp.next;
                }
        }
        return head;
}
```

Q.12 Print the reverse linkedlist by using recursion function.

//code//

```
public static void displayR(Node head){
        if( head == null){
                return;
        }
        System.out.print
        displayR(head.next);
        System.out.print (head.data + " ");
}
```

time complexity - O(n)
Space complexity - O(n)

DATE 06 05 2024

leetcode

* Reverse the linkedlist by recursively.

Ex: 3 - 5 - 7 - 9 - 11 - null

Reverse: 11 - 9 - 7 - 5 - 3 - null

```
//code// public static Node reverseList( Node head) {
        if( head == null) { return null;}
        if ( head.next == null) return head ;
        Node temp = reverselist (head.next);
        head.next.next = head;
        head.next = null;
        return temp ;
}
```

eg: 1 → 2 → 3
h h.next
h.next.next = h
3 → = 2 || 3 → 2

working: 3 - 5 - 1 - 2 - 4

①

p.s.N. reverseList(3){
false if(head.next == null) r. h.

Node t = reverselist (5);
head.next.next = head
head.next = null
return t;

3 → 5 ← 1 ← 2 ← 4
null
null ← 3 ← 5 ← 1 ← 2 ← 4

②

p.s.N. reverseList(5) {
false if(head.next == null) r. h

Node t = reverse list (1);
head.next.next = head
head.next = null ;
return t;

5 - 1 ← 2 ← 4
null
null 5 ← 1 ← 2 ← 4

④

p.s.N. reverselist(2);
if(head.next == null) r.h

Node t = reverse list(4);
head.next.next = head
head.next = null;
return t;

2 - 4 - null

null 2 ← 4

③

p.s.N. reverse List (1);
if(head.next == null);

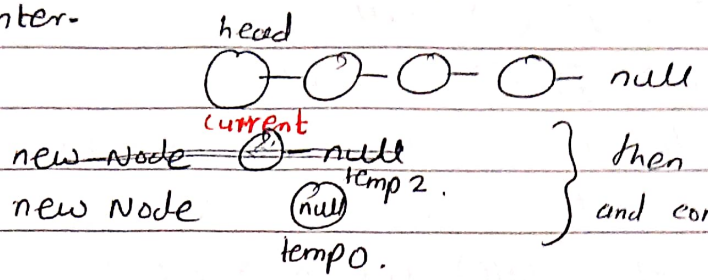
Node t = reverselist(2);
head.next.next = head;
head.next = null;
return t; 4 → 2 → (null)

1 → 2 ← 4
null

1 ← 2 ← 4
null

Q 13) Reverse LinkedList by iterative method, using three pointer.

→



then temp2 moves current.next and connect current to temp0.

```
//code// public Static Node reverseLinkedList(Node head){
        Node current = head;
        Node temp0 = null;
        Node temp2 = null;
        while (current ≠ null){
            temp 2 = current.next;
            current.next = temp0;
            temp0 = current;
            current = temp2;
        }
        return temp0;
    }
```

Leetcode Q 14) Palidrome of List.

ex: 1) 1 — 2 — 3 — 2 — 1 — null

2) 1 — 2 — 2 — 1 — null

Logic: 1) find middle term by using ~~slow~~ two pointer (slow & fast).

2) reverse remaining list (after slow Node List)

3) then compare value ~~head~~ node of before middle and node of after middle (value compare).

4) then return true or false.

5) Note: given node is 1 then return true.

DATE

//code//
```
public static Node isPalidrome (Node head){
        if(head.next == null){
                return true;
        }
        ~~List~~ Node slow = head;
        Node fast = head;
        ~~boolean~~
        while (fast.next ≠null && fast.next.next = null){
                slow = slow.next;
                fast = fast.next.next;
        }
        ~~li~~ Node rev = reverselist(slow.next);        //method call//
        slow.next = rev;
        ~~List~~ Node p1 = head;
        ~~List~~ Node p2 = slow.next;
        while ( p1 ≠ p2 ){ ~~return false~~;
                if( p1.val ≠ p2.val )  return false;
                p1 = p1.next;
                p2 = p2.next;
        }
        return true;
}
```

Q15) Create the deep copy of linked List (create duplicate List).

//code//
```
public static Node duplicateList (Node head){
        Node temp1 = head.next;
        Node temp2 = new Node (head.val);
        Node head1 = temp2;
```

→

```
        while ( temp1 ≠ null) {
                Node nex = new Node(temp2.val);
                temp2.next = nex;
                temp2 = temp2.next;
                temp1 = temp1.next;
        }
        return head1;
}
```

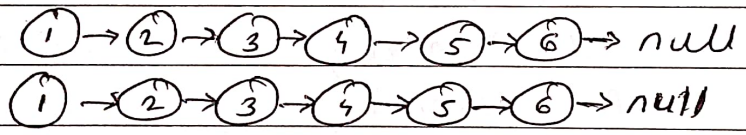
Leetcode

Q15) Copy List with random pointer.

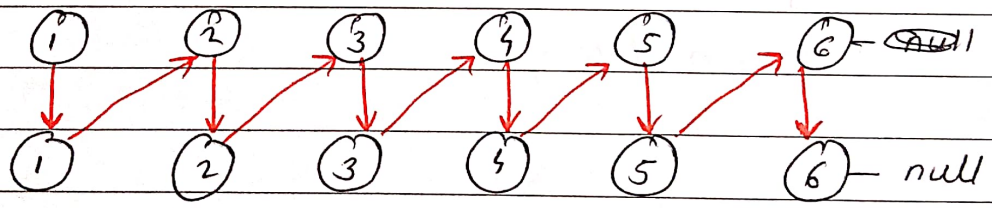
→ Steps :
1) Create deep copy of linked list.
2) Create alternate connections.
3) Assigne randon pointers.
4) Seperating the list

→ I/p : 1 → 2 → 3 → 4 → 5 → 6 – null

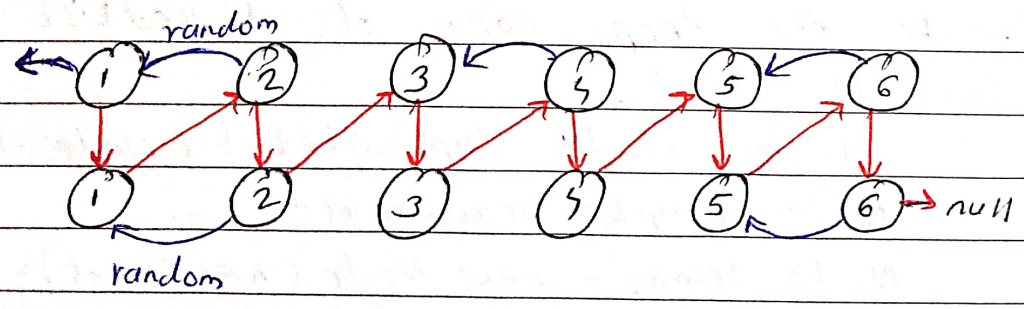
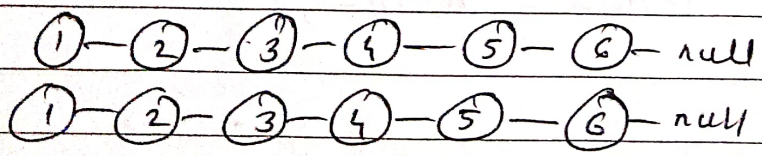
1)



2)



3)



4)