Recursion Question

Print Fibonacci Number Recursively

Approach

- In the **fib** method:
  - If n=1, it returns 0 (the first Fibonacci number).
  - If n=2, it returns 1 (the second Fibonacci number).
  - For n>2, it recursively calculates the nth Fibonacci number by adding the (n-1) and (n-2) Fibonacci numbers.

```
public class fibonnaci {
    public static void main(String[] args) {

        int n=7 ; // number

        int ans= fib(n); //functions

        System.out.println(ans);
    }

    public static int fib(int n){

        if(n==1) return 0; //base case if n becomes 1
        else if(n==2) return 1; //base case if n become 2

        int n1 =fib(n-1) ; // recursive call for n-1 term
        int n2 = fib(n-2); // recursive call for n-2 term

        int ans = n1 + n2; // calculate the next term
        return ans ;

    }
}
```

```
Output : 8
```

Print Factorial of given number Recursively

Approach

- The program calculates the factorial of 5.
- It uses a recursive method called **fact** to compute the factorial.
- In the **fact** method:
  - If **n** is less than 1, it returns 1 (base case).
  - For **n** greater than or equal to 1, it multiplies **n** with the factorial of (n-1) obtained through recursion.

```
public class factorial {
    public static void main(String[] args) {

        int n=5 ; //number

        int ans = fact(5); //function call

        System.out.println(ans);

    }

    public static int fact(int n){
        if(n<1){  //base case if n is less than 1.
            return 1;
        }

        int preFact = fact(n-1); //recursive call decrease n by 1 everytime functions call itself
        return n * preFact;  // return result
    }
}
```

```
Output : 120
```

Find power of given number Recursively

Approach

Suppose **n** is 2 and **x** is 3.

- First, the **power** method is called with **n=2** and **x=3**. Since **x** is not 1, it goes into the recursive case.
- It calls itself with **power(2, 2)**, and again, **x** is not 1, so it continues.
- It calls itself with **power(2, 1)**. Now, **x** is 1, and it returns **2** (base case).
- Now, **power(2, 2)** can calculate its result as **2 * 2 = 4**.
- Finally, **power(2, 3)** can calculate its result as **2 * 4 = 8**.

```
public class Power {
    public static void main(String[] args) {
        int n = 2; // number
        int x = 3; // exponent power

        int ans = power(n, x); //function
        System.out.println(ans);
    }

    public static int power(int n, int x) {
        if (x == 1) { //base case
            return n; // return n because any number raise to power 1 is number itself.
        }
        int pre= power(n,x-1); // recursive call
        int ans = n*pre; //calculate answer
        return ans ;
    }
}
```

```
Output : 8
```

Find k element in Array Recursively

Approach

Here our target is 8.

- Send **array** , an **index** number starting from **0** and **target** value.
- Set your **base** case , if index goes **out of bound** meaning greater than index value of array **return -1** , i.e, element was not present.
- If element is found then return it’s **index**.
- Else increase the index by 1, With each function call.

```
public class Main {
    public static void main(String[] args) {

        int[] arr = {1,2,3,4,5,6,7,8};
        int k=8; //target element

        int ans = find(arr, 0, k); //function call

        System.out.println(ans);

    }

    public static int find(int[] arr , int idx , int k){

        if(idx==arr.length){ // if idx goes out of bound meaning greater than array index
            return -1; // return -1 element was not present
        }
        else if (arr[idx]==k){  //if target is found return it's index
            return idx;
        }

        return find(arr,idx+1,k); // recursive function call increase idx by 1

    }
}
```

```
Output : 7
```

Find sum of digit Recursively

Approach

- The base case checks if **n** is a single-digit number (i.e., **n** is greater than or equal to 0 and less than or equal to 9). If **n** is a single digit, there is no need to further split it into digits, so it returns **n** itself.
- If **n** is not a single digit, the function proceeds to extract the last digit of **n** by taking the remainder when dividing by 10 (i.e., **int digit = n % 10**).
- The remaining part of the number (without the last digit) is calculated by dividing **n** by 10 (i.e., **int remSum = sumDigit(n / 10)**), and the **sumDigit** function is called recursively on

this remaining part.

- Finally, the function returns the sum of the last digit (**digit**) and the result of the recursive call (**remSum**).

```
public class Main {
    public static void main(String[] args) {

        int n = 123456; // number

        int ans = sumDigit(n); // functions call

        System.out.println(ans);

    }

    public static int sumDigit(int n){

        if(n>=0 && n<=9){ // base case if n is single digit we dont need to call our function again
            return n;
        }
        int digit = n%10; // get last digit
        int remSum = sumDigit(n/10); //remove the last digit from number and call the function
        return digit + remSum; //find sum and return result.
    }
}
```

```
Output : 21
```

Binary Search