**Recursion Assignment**

**1.Factorial Calculation**

**public static int fact(int n){**

**if(n==1){**

**return 1;**

**}**

**else{**

**return n\*fact(n-1);**

**}**

**}**

**Input: 8**

**Output: 40320**

**Explanation**: The recursive call starts and fact(n-1) is called as many as times base condition is not full filled. When the base condition becomes true, it returns 1 which is basically the fact of 1. Then onwards on each step we return a value like in first return case 1 is returned which gets multiplied with n that, at that call stack which is 2. So, hence 1\*2 which is now passed to the previous calling function. It in turn multiplies 3 into the return value 1\*2. The process goes on until the main method is called.

**2.** **Fibonacci Series**

**public static int fib(int n){**

**if(n<=1){**

**return n;**

**}**

**else{**

**return fib(n-1)+fib(n-2);**

**}**

**}**

**Input: 5**

**Output: 5**

**Explanation**: Here the one call made to fib() is actually two calls made to one fib(n-1) and other to fib(n-2) like in Tower of Hanoi problem. We implement using a recursive tree where the value of child nodes like fib(1) will give 1 and fib(0) will give 0. So, tracing backwards we will reach parent node and assign values accordingly.

**3.Sum of Digits**

**public static int sum(int n){**

**if(n==0)**

**return 0;**

**return (n%10 + sum(n/10));**

**}**

**Input: 644**

**Output: 14**

**Explanation**: The last recursive call where the base condition is satisfied returns 0. Then, the previous calls receive 0 plus the n%10 which signifies the last digit left in the number which here is 6. Then 0+6 is returned to previous call then the same process goes on.

**4. Reverse a String**

**public static String TocharArray(String s) {**

**char[] a = s.toCharArray();**

**rev(a, 0, a.length - 1);**

**return new String(a);**

**}**

**public static void rev(char[] s, int f, int l) {**

**if (f >= l)**

**return;**

**char temp = s[f];**

**s[f] = s[l];**

**s[l] = temp;**

**rev(s, f + 1, l - 1);**

**}**

**Input: Hello**

**Output: olleH**

**Explanation**: In the above code the first function tocharArray uses a predefined function toCharArrays() func to convert string to array. Then on it calls another function to reverse this array. The functions reverse array by swapping elements between first and last index elements. In that function we recursively call the function by increasing the value of the variable f and decreasing l. When f>l that is f has crossed l then we implement base condition. After this the array is already reversed in the memory. Then a new string is generated from this array and returned to main.

**5. Power Function**

**public static int power(int a, int b){**

**if(b==1){**

**return a;**

**}**

**return a\*power(a,b-1);**

**}**

**Input: (2, 5)**

**Output: 32**

**Explanation:** The function power recursively calls itself but decreasing the value of b which is the power and keeping a same. When the base condition is fulfilled any value to the power of 1 is the number itself. Then basically a is returned to previous call. Then a is returned and a\*a is returned to previous call and so on.

**7. Check Palindrome**

**public static void palin(String s){**

**char arr[] = s.toCharArray();**

**palindrome(arr,0,arr.length-1);**

**String res = new String(arr);**

**if(res.compareTo(s)==0){**

**System.out.println("true");**

**}**

**else{**

**System.out.println("false");**

**}**

**}**

**public static void palindrome(char a[], int f, int l){**

**if(f>=l){**

**return ;**

**}**

**char swap = a[f];**

**a[f]= a[l];**

**a[l]= swap;**

**palindrome(a,f+1,l-1);**

**}**

**Input: dad**

**Output: true**

**Explanation:** The above codeworks same as before then reverse is just compared with original string. If same then true is returned.

**8. Binary Search**

**public static int binarySearch(int arr[], int i, int index){**

**if(index==arr.length){**

**return -1;**

**}**

**else{**

**if(arr[index]==i){**

**return index;**

**}**

**return binarySearch(arr,i,index+1);**

**}**

**}**

**Input: {10,22,44,66,5,0},5**

**Output: 4**

**Explanation:** The code recursively calls itself also increasing the index of the array thus decreasing the search area. Then at last if the index is equal to array length that is index is out of array index then it will return -1 which signifies element not found in array.

**9. Tower of Hanoi**

**public static void toh(int n, int s, int inter,int d){**

**if(n == 1){**

**System.out.println("Disk from "+s+" to "+d);**

**}**

**else{**

**toh(n-1,s,d,inter);**

**System.out.println("Disk from "+s+" to "+d);**

**toh(n-1,inter,s,d);**

**}**

**}**

**Input: n = 3**

**Output:**

**Disk from 1 to 3**

**Disk from 1 to 2**

**Disk from 3 to 2**

**Disk from 1 to 3**

**Disk from 2 to 1**

**Disk from 2 to 3**

**Disk from 1 to 3**

**Explanation**: The code here iscalling itself recursively in one call twice. For one iteration it will run two recursive calls until n becomes zero. When it becomes zero the code goes back to again running from first recursive call with n-1 disk. After that two recursive call with n-1-1 until it becomes 0.