Recursion in java

Assignment Solutions

PW SKILLS

Q1 : Given an integer, find out the sum of its digits using recursion.
Input: n= 1234
Output: 10
Explanation: 1+2+3+4=10

Solution :
Code : ASS_Code1.java

Output :

```
Enter the number
1234
 Sum of digits in 1234 is 10
```

Approach:
Let's define a function func which will take a number as input and will return the sum of all of its digits. now this function can we written in this way first digit + sum of remaining digit (the same way human do sum in left to right manner) but as extracting first digit is relatively tough thus we will start with last digit so the function func(num) can be written as func(num) = last_digit + func(num after removing last digit). Now to get the last digit we can use % operator and to remove the last digit we can use / operator.

Let's see the step-by-step approach for a better understanding of how the algorithm works.
Let the number be 1234.
Step 1-> 1234 % 10 which is equal-to 4 + (send 1234/10 to next step)
Step 2-> 123 % 10 which is equal-to 3 + (send 123/10 to next step)
Step 3-> 12 % 10 which is equal-to 2 + (send 12/10 to next step)
Step 4-> 1 % 10 which is equal-to 1 + (send 1/10 to next step)
Step 5-> 0 algorithm stops.

Q2: Given a number n. Find the sum of natural numbers till n but with alternate signs.
That means if n = 5 then you have to return 1-2+3-4+5 = 3 as your answer.
Constraints : 0<=n<=1e6
Input1 : n = 10
Output 1 : -5
Explanation : 1-2+3-4+5-6+7-8+9-10 = -5

Input 2 : n = 5
Output 2 : 3

Solution:
Code : ASS_Code2.java

Output :

```
5
The result is : -3
```

Approach :

- Here,we have created a go function of int type to get the sum of natural numbers with alternate signs till given n.
- We have passed 2 parameters ,n (the number till you want the alternating sequence sum) , a variable i (to keep track of odd and even numbers) .
- If i has reached n+1(base case condition) that means we have already calculated the sum of the sequence

till n so we returned from this function.

- If this is not the case(i.e i<=n) we can clearly observe from the sequence that every even number of this sequence is added with a negative polarity while every odd number is added with positive polarity.
- We have done the same as we observed in the pattern. For every 'i' we check if the if i is odd then we return the sum of (i) and value returned by go(n,i+1) and if i is even then we return the sum of (-i) and value returned by go(n,i+1). This is how we are recursively calling the go function.

Q3: Print the max value of the array [13, 1, -3, 22, 5].

Code ASS_Code3.java

Output :

```
The max value is : 22
```

Approach :

We start our function call from maxValue(arr, 5,0) which calls for maxValue(arr,5, 1) and so on. We will reach our base case at the last index of the array. In this case, it will be index 4. In the base case, since we have only one value, we can say that for the array portion starting from that index, the value itself is the max value that we can get.

MaxValue(arr,5, 4) will return arr[4] i.e. 5.

Using this the max value of the array part starting at index 3 will be calculated.

MaxValue(arr,5, 3)= max(arr[3], MaxValue(arr,5, 4))
= max(22, 5)
= 22
Similarly,
MaxValue(arr,5, 2)= max(arr[2], MaxValue(arr,5, 3))
= max(-3, 22)
= 22
MaxValue(arr,5, 1)= max(arr[1], MaxValue(arr,5, 2))
= max(1, 22)
= 22
MaxValue(arr,5, 0)= max(arr[0], MaxValue(arr,5, 1))
= max(13, 22)
= 22
At the end we have MaxValue(arr,5, 0)= 22, the desired output.

Q4 : Find the sum of the values of the array [92, 23, 15, -20, 10].

Code : ASS_Code4.java

Output :

```
The sum of all elements of the array is : 120
```

Explanation:

- The calculating sum of array elements using an iterative approach is simple and straightforward using loops. Let's discuss how to calculate the sum of array elements using recursion.
- To solve this problem recursively, I have created one function totalSum(). It takes two arguments, one is an array and the second is the length of an array and will return the sum of elements of the array. The method

totalSum() calls itself recursively until the base or terminating condition is not reached.

- The base condition is when the value of n is less than or equal to zero. Until the value of n is not reached zero it calls itself.
- If n i.e size of the array is greater than zero(i.e base condition is not satisfied) , then we will call the function totalSum(arr,n-1) and return the sum of values returned by totalSum(arr,n-1)and arr[n-1].

Q5. Given a number n. Print if it is an armstrong number or not.
An armstrong number is a number if the sum of every digit in that number raised to the power of total digits in that number is equal to the number.
Example : 153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153 hence 153 is an armstrong number. (Easy)

Input1 : 153
Output1 : Yes

Input 2 : 134
Output2 : No

Solution :

Code : ASS_Code5.java

Output :

```
Enter the number n:
153
yes
```

Approach :

- First, we calculated a power function which calculates the power of any number a raised to the power b.
- If b = 0, it means that the power to which the number needs to be raised is zero, then the value is equal to 1 and this would be our base case from where we need to terminate this code.
- If the power is even then the number can be splitted in half power.
- If the power is odd then the number can be splitted in half power and 1 additional 'a' value can be multiplied externally.
- Create another function where we pass two parameters n(the original number) and dig(number of digits in the number) . Now we need every single digit of the number raised to 'dig' power. To extract any digit we can do n%10 this will give us the last digit of the number. But since we only want sum and nature of sum is commutative, that is in any order we do the sum it is independent of the order.
- After extracting the last digit we do not need it any more so we will pass the value n/10 to the next recursive call and this will be repeated until n does not turn out to be 0. Once n = 0 that means we have touched the base case where it shows that no digit is left in the original number.
- In the main function we calculate the number of digits present in 'n'. The number of times we are required to divide n by 10 till it becomes zero indicates the number of digits in that number. Eg. let n = 1234