Recursive Problems

Use only recursion to implement these Problems.

Numbers:

1. `double power (double a, int b)` that calculates the power a^b. You are allowed to use the multiplication operator *.
2. `int mul (int a, int b)` that computes the product of two integers a and b. The only arithmetic operation that you are allowed to use in this problem is addition +.
3. Write a recursive method to print the digits of a number right to left vertically from top to bottom. For example, if n = 1234 is passed to the function it prints 4, 3, 2 and 1 vertically.
4. Do the same thing as in the previous problem but in the opposite order, i.e., print the digits of a number from left to right vertically from top to bottom. For example, if n = 1234 is passed to the function it prints 1, 2, 3 and 4 vertically.
5. Write a recursive function that counts and return the number of zero digits in a number, `int countZeros (int n)` in if n = 10200 the it should return 3.
6. Write a recursive method that return the difference between the smallest and greatest digit of a number passed in input. For example, if n = 897714 was passed in input, the function should output 8, which is the difference between 1 and 9, etc.
7. Write a recursive method to convert a given number into its binary representation. For example, if the number n = 5 is passed to the function it should return 101, it n = 25 is passed to it, it should return 11001 etc. The 1's and 0's should be returned in a `vector<int>` (passed by reference).
8. Reverse Digits: Returns the positive integer that you get, when you reverse the digits of parameter n. For example, when called with the parameter 12345, this method would return 54321. (Do this with proper integer arithmetic instead of just simply converting to String)
   `int reverseDigits (int n)`
9. `void listNumbers (int start, int end)` that outputs the numbers from start to end to console. Write one version that outputs the numbers in ascending order, and another that outputs them in descending order.
10. `int sumOfDigits (int n)` that computes and returns the sum of digits of the positive integer n. For example, when called with the parameter 12345, this method would return 15.

Series:

1. Fibonacci numbers: that print first n Fibonacci numbers
   0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... where each number is the sum of the preceding two.
2. `double harmonicSum(int n)` that calculates and returns the sum 1 + ½ + 1/3 + ... + 1/n.

Arrays:

1. `int min(int[] a, int start, int end)` that returns the smallest element between the indices start and end in the parameter array a.
2. Even Odd: It will rearrange an array of int values so that all the even values appear before all the odd values. `void evenOdd (int * A, int size);`
3. Check sum: It will check if an array A of integers contains an integer A[i] that is the sum of two integers that appear earlier in A, that is, such that A[i] = A [j] + A[k] for j, k < i.
   `bool check_sum (int * A, int size);`
4. In a two-player game played with an array of numbers, the player whose turn it is to move can take a number from either the beginning or the end of the array, and gets that many points. The

game is over when the array becomes empty, and the player whose point total is greater wins. Write a method `int gameValue(int a[], int start, int end)` that computes the result for the game for the player whose turn it is to move, where the subarray of a from start to end contains the numbers that remain.

5. `bool subsetSum(int a[], int n, int goal)` that checks if it is possible to select some subset of the first n elements of the array a so that the selected elements add up exactly to goal.
6. `bool binarySearch(int a[], int start, int end, int x)` that uses the binary search algorithm to check whether the sorted array a contains the element x anywhere between indices start and end, inclusive.
7. `void shuffle (int a[], int start, int end, Random rng)` that shuffles the elements in the subarray from start to end, getting the random numbers it needs from the rng provided as parameter.

Strings:

1. Write a recursive function `bool isPalindrome(string s)`. It should return true if is s is a palindrome and false otherwise. Recall that a palindrome is a string that reads the same both ways, e.g. madam, civic, deified, 10101 etc. are all palindromes.
2. `String repeat (String s, int n)` that creates and returns a new string that is made by concatenating n copies of the parameter string s. For example, calling this method with the parameters "Hello" and 3 would return the string "HelloHelloHello". If n equals zero, the method should return the empty string.
3. `String mySubstring(String s, int start, int end)` that works like the substring method of the String class. You can only use the String methods charAt and +.
4. `int count (String s, char c)` that counts how many times the character c occurs inside string s. You can use the String methods charAt and substring to extract information from string s.
5. `String disemvowel(String s)` that creates and returns a string that is otherwise the same as the parameter string s, except that all vowels have been removed. (For simplicity, assume that you already have the method `bool isVowel(char c)` that tells whether the character c is a vowel.)
6. Write a recursive method to reverse the contents of a String. `String reverse (String s)`
   Sample input: hello
   Sample output: olleh
7. Write a recursive method `void print01(int k);` that prints all 0/1 strings of length k.
   For example,
   if k=1, the program should print 0 and 1.
   If k=2, it should print 00, 01, 10 and 11, etc.

Random Problems:

1. When you have a set of n people, how many ways are there to choose a committee of k members, that is, a k-element subset? The order in which you choose the people who end up in the committee doesn't matter: that is, first choosing Alice, then Bob and then Carol is the same as first choosing Carol, then Bob and then Alice. The answer is given by the recursive equation with base cases C(0,k) = 0 and C(n,1) = n, and recursion C(n,k) = C(n-1,k-1) + C(n-1,k). Write this function as a recursive method `int choose (int n, int k).`

2. **Count Paths:** You are standing at the point (n,m) of the grid of positive integers and want to go to origin (0,0) by taking steps either to left or down: that is, from each point (n,m) you are allowed to move either to the point (n-1,m) or the point (n,m-1). Write a method countPaths that counts the number of different paths from the point (n, m) to the origin.
   `int countPaths(int n, int m);`
3. Find all Subsets of Size k for a given Set.
   The function `int** ComputeSubsets(int*a, int size int k)` will take a dynamic array (Set S), size of array and an integer k (where 1 ≥ k ≤ Set-Size) as parameters and will return a 2-D dynamic array, which contain all subsets of set s of size k.
   For example, let S = {1, 2, 3, 4, 6} and k=3
   then your function must return all subsets of size 3 which will be
   {1,2,3}, {1,2,4}, {1,2,6}, {1,3,4}, {1,3,6}, {1,4,6}, {2,3,4}, {2,3,6}, {2,4,6}, {3,4,6}.

Link List:

Note: Use singly linked list implementation for the following question. Use only recursion to implement these operations.

1. Implement a recursive member function `insertATHead` which recursively insert element at the head of the linked list. `void insertATHead (T a)`
2. Implement a recursive member function recursive which recursively reverses the linked list.
   `void recursiveReverse ()`
3. Implement a recursive member function isEqual which is passed a linked list object as parameter. The function then recursively checks whether the contents (data) of the two linked lists are equal or not. `bool isEqual (List const& obj)`
4. Write a method to print a singly linked list in reverse using recursion.
5. Write a method to print the list in such a way that the odd numbered nodes, i.e., the first, third and so on, are printed in the straight order first, followed by the even numbered ones printed in the reverse order. Write a single recursive method to do this.
   For example: If linked list contains the following elements: 1->2->3->4->5->6
   Output is: 1-3>5->6->4>2
6. Implement a recursive member function Merge which is passed a linked list object as parameter. The function then merge the two sorted lists so as to produce a combined sorted list.
   `List* Merge (List const& h1)`
   Input : list1: 1->3->5->7 , List2: 2->4
   Output : 1->2->3->4->5->7
7. Implement a recursive member function print, which print the list recursively. `void print ()`
8. Create a main function with following instructions:
   a. Insert at head of your singly linked list: 0, 6, 7, 8.
   b. Insert at head of your singly linked list: 1, 2, 9.
   c. Now merge lists created in step (a) and (b) and print result.
   d. Reverse the linked list created in step (a) and print it.
   e. Now check whether the linked lists created in step (a) and (b) are equal or not.

Binary Search Tree:

The algorithms for binary search trees can often naturally be written recursively, since a binary search tree itself is a recursively defined structure: a BST is either empty (this is the base case of structural recursion), or it consists of a root node whose left and right children are themselves BST's. Assume that the class Node has fields int key, Node left and Node right.

1. Write a recursive method `Node maximum (Node root)` that returns a reference to the node that contains the maximum value in the tree that starts from the given root node.
2. Write a recursive method `Node contains (Node root, int key)` that returns a reference to the node that contains the given key, and returns null if there is no such node.
3. Write a recursive method `void add (Node root, int key).` You can assume here that the tree is initially nonempty. However, if the tree already contains the given key, this method should not add another one (so our tree is a set, not a multiset).
4. Write a recursive method `void rangeSearch (Node root, int min, int max)` that outputs all the keys in the tree whose value is between min and max, inclusive. Don't just recursively walk through the whole tree, but examine only those branches of the tree that you really need to examine.
5. The empty tree has a height of zero, and otherwise the tree height is one plus the length of the path from the root to the farthest leaf node. Write a recursive method `int height(Node root)` to compute the height of the given tree.
6. Write a method `Node createTree (int a [], int start, int end)` that creates and returns a BST that contains the elements of the array a between the indices start and end, inclusive. You can assume that the parameter array a is already sorted in ascending order.