Sum triangle from array

Given an array of integers, print a sum triangle from it such that the first level has all array elements. From then, at each level number of elements is one less than the previous level and elements at the level is be the Sum of consecutive two elements in the previous level.

```
Input : A = {1, 2, 3, 4, 5}
Output : [48]
         [20, 28]
         [8, 12, 16]
         [3, 5, 7, 9]
         [1, 2, 3, 4, 5]

Explanation :
Here,   [48]
        [20, 28] -->(20 + 28 = 48)
        [8, 12, 16] -->(8 + 12 = 20, 12 + 16 = 28)
        [3, 5, 7, 9] -->(3 + 5 = 8, 5 + 7 = 12, 7 + 9 = 16)
        [1, 2, 3, 4, 5] -->(1 + 2 = 3, 2 + 3 = 5, 3 + 4 = 7, 4 + 5 = 9)
```

Write this function as a recursive function.

You are provided with the print function.

```
void print_array(int A[] , int n)
// EFFECTS: Print current array in the end so
// that smaller arrays are printed first
{
    for (int i = 0; i < n ; i++){
        if(i == n - 1)
            cout << A[i] << " ";
        else
            cout << A[i] << ", ";
    }
    cout << endl;
}
```

Solution:

```
void print_triangle(int A[] , int n) {
    // Base case
    if (n < 1)
        return;

    // Creating new array which contains the
    // Sum of consecutive elements in
    // the array passes as parameter.
    int temp[n - 1];
    for (int i = 0; i < n - 1; i++){
        int x = A[i] + A[i + 1];
        temp[i] = x;
    }
    // Make a recursive call and pass
    // the newly created array
    print_triangle(temp, n - 1);
    print_array(A, n);
}
```

Generate Binary Strings

Given a integer K. Task is to print all binary string of size K without consecutive 1’s.

Examples:

```
Input : K = 3
Output : 000 001 010 100 101

Input : K  = 4
Output : 0000 0001 0010 0100 0101 1000 1001 1010
```

Write this function as a recursive function. Hint: You may need a helper function.

Solution:

```
void generate_strings_helper(int K, char str[], int n) {
    // print binary string without consecutive 1's
    if (n == K){
        // terminate binary string
        str[n] = '\0' ;
        cout << str << " ";
        return ;
    }

    // if previous character is '1' then we put
    // only 0 at end of string
    //example str = "01" then new string be "010"
    if (str[n-1] == '1') {
        str[n] = '0';
        generate_strings_helper (K, str, n+1);
    }

    // if previous character is '0' than we put
    // both '1' and '0' at end of string
    // example str = "00" then new  string "001" and "000"
    if (str[n-1] == '0') {
        str[n] = '0';
        generate_strings_helper(K, str, n+1);
        str[n] = '1';
        generate_strings_helper(K, str, n+1) ;
    }
}

void generate_strings(int K) {
    // Base case
    if (K <= 0)
        return ;

    // One by one stores every binary string of length K
    char str[K];

    // Generate all Binary string starts with '0'
    str[0] = '0' ;
    generate_strings_helper (K, str, 1) ;

    // Generate all Binary string starts with '1'
    str[0] = '1' ;
    generate_strings_helper (K, str, 1);
}
```

Pseudo-perfect Numbers

A number `n` is called *pseudo-perfect* if it is equal to the sum of a subset of its proper divisors. The subset cannot contain duplicate divisors. For example, 12 is pseudo-perfect since it can be written as
2+4+6, where 2, 4, and 6 are three of its proper divisors.

Note: A proper divisor is a positive divisor of a number n, excluding n itself. For example, 1, 2 and 3 are proper divisors of 6, but 6 itself is not.

In this problem, you will write a program that takes an integer as its argument and checks whether it is a pseudo-perfect number. It outputs 1 if the number is and 0 otherwise.

For example, suppose that your program is named as `is_pseudoperfect`. If you run it as
`./is_pseudoperfect 12`
the output should be
`1`

Consider the `list_t` data structure defined in Lecture 7 with the following functions:

```
bool list_isEmpty(list_t list);
// EFFECTS: returns true if "list" is empty, false otherwise

list_t list_make();
// EFFECTS: returns an empty list

list_t list_make(int elt, list_t list);
// EFFECTS: given "list", make a new list consisting of
// the new element "elt" followed by the elements
// of the original "list"

int list_first(list_t list);
// REQUIRES: "list" is not empty
// EFFECTS: returns the first element of "list"

list_t list_rest(list_t list);
// REQUIRES: "list" is not empty
// EFFECTS: returns the list containing all but the first
// element of "list"
```

The only standard libraries you are allowed to use in this problem are and .

1. Implement `list_proper_divisors`, which takes a positive integer as input and returns a list of its proper divisors in ascending order. You are not required to implement this function recursively.

```
list_t list_proper_divisors(int num);
// REQUIRES: "num" is a positive integer
// EFFECTS: returns a list_t which contains all the proper
// divisors of "num" in ascending order
```

Solution:

```
list_t list_proper_divisors(int num) {
    list_t list = list_make();
    for (int i = num - 1; i > 0; i--) {
        if (num % i == 0) {
            list = list_make(i, list);
        }
    }
    return list;
}
```

2. Before writing the function `is_pseudoperfect`, implement a helper function first. It's up to you to decide the REQUIRES and EFFECTS of this function. Write this function as a recursive function.

```
bool is_pseudoperfect_helper(int num, list_t proper_divisors);
// REQUIRES: TODO
// EFFECTS: TODO
```

Solution:

```
bool is_pseudoperfect_helper(int num, list_t proper_divisors) {
    if (num == 0) {
        return true;
    }
    if (num < 0 || list_isEmpty(proper_divisors)) {
        return false;
    }
    return is_pseudoperfect_helper(num - list_first(proper_divisors),
list_rest(proper_divisors)) ||
           is_pseudoperfect_helper(num, list_rest(proper_divisors));
}

// Simpler solution
bool is_pseudoperfect_helper(int num, list_t proper_divisors) {
    if (list_isEmpty(proper_divisors)) return num == 0;
    return is_pseudoperfect_helper(num, list_rest(proper_divisors)) ||
           is_pseudoperfect_helper(num - list_first(proper_divisors),
list_rest(proper_divisors));
}
```

3. Please implement the function `is_pseudoperfect` which takes a positive integer as input and returns a Boolean type indicating whether the integer is pseudo-perfect. You may want to use the function `list_proper_divisors` and the helper function implemented above.

```
bool is_pseudoperfect(int num);
// REQUIRES: "num" is a positive integer
// EFFECTS: returns true if "num" is a pseudoperfect number,
// false otherwise
```

Solution:

```
bool is_pseudoperfect(int num) {
    list_t proper_divisors = list_proper_divisors(num);
    return is_pseudoperfect_helper(num, proper_divisors);
}
```