Recursion Problem

1. M Apples are placed in N Plates.

We want M apples to be placed in N plates. It allows empty plates. How many choices to distribute apples to plates? And it allows repeatable combinations. For example, M=4, N=3, (1, 1, 2), (1, 2, 1) and (2, 1, 1) are repeatable combinations, they are regarded as three choices.

```
int fun(int m, int n){
     if(m <= 0)
          return 1;
     if(n <= 0)
          return 0;
     int num=0;
     for(int i=0;i<=m; ++i){
          num += fun(m-i, n-1);
     }
     return num;
}
```

2. 4 0~9 Numbers form 24 points with +-/*

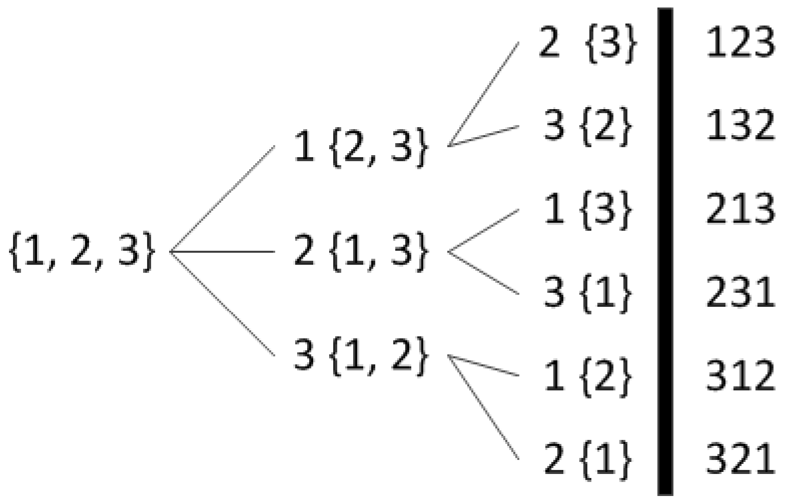
We want 4 decimal numbers which ranges from 0~9 to form 24 point with operations +-*/. For example, 4 numbers: 7, 3, 2, 4. Then (7+3)*2+4=24.

The issue can be divided into two parts:

1) Four numbers in various orders, totally 4*3*2*1 = 24 choices.

2) For each order, justify whether it can form 24 or not.

For 1),



```
SET_COMB[MAX][4];

Set_cnt=0;

Permute(set, start, end){

    If (start==end){

        Memcpy(SET_COMB[set_cnt], set, 4 * sizeof(int)); // save the final result

        Set_cnt++;

    }

    For (i=start;i<end;i++){

        Swap(set, start, i); // exchange set[start] and set[i]

        Permute(set, start+1, end);

        Swap(set, start, i); // recover the set

    }

}
```

比如说，有{1，2，3，4}, 先固定 1， 然后用 1， 2， 3，4 去替换 1， 得到 4 个并行的集合

{1, x, x, x},

{2, 1, x, x},

{3, x, 1, x},

{4, x, x, 1}

然后分别对单个集合做第二层的处理，第一个元素不用管了，因为数值不同，所以后面不会产生重复的组合。

{1, x, x, x} = {1, 2, 3, 4} => {2, 3, 4}, 固定 2， 用后面的元素替换(包括自己)， 得到以下集合

{2, x, x},

{3, 2, x},

{4, x, 2}

接着又可以延申到第三层。。。

For 2)

```
int func(int result, int* nums, int len){
    int i;
    int success;
```

```
        int temp_result;

        if ((len == 1) && (result == nums[0]))
        {
                return 1;
        }

        if(len == 1){
                return 0;
        }

        success = 0;
        for (i=0;i<4;i++){
                switch (i){
                        case 0: temp_result = result - nums[len-1]; break;
                        case 1: temp_result = result + nums[len-1]; break;
                        case 2: temp_result = result * nums[len-1]; break;
                        case 3:
                                if ((result%nums[len-1])==0){
                                        temp_result = result / nums[len-1];
                                }else{
                                        return success;
                                }
                                break;
                }

                success += func(temp_result, nums, len-1);
        }

        return success;
}
```