40. Combination Sum II

link

这个必须sort 否则不好去除重复的, 去除重复的是sort之后 后一个跟前一个相等然后continue;

```
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        List<List<Integer>> res = new ArrayList();
        Arrays.sort(candidates);
        helper(candidates, target, new ArrayList(), res, 0, 0);
        return res;
    }
    public void helper(int[] candidates, int target, List tmp, List res, int index,
    int record){
        if(record > target) return;
        if(record == target){
            res.add(new ArrayList(tmp));
        }else{
            for(int i = index; i < candidates.length; ++i){
//这里得去除重复的.
                if(i > index && candidates[i] == candidates[i - 1]) continue;
                tmp.add(candidates[i]);
                helper(candidates, target, tmp, res, i + 1, record + candidates[i]);
                tmp.remove(tmp.size() - 1);
            }
        }
    }
```

39. Combination Sum

link

这里可以是任何顺序, 不用sort, 但是得注意传index否则每次从0开始, 后面的会把前面计算过的再算一次

```
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> res = new ArrayList();
        helper(candidates, target, new ArrayList(), 0, res, 0);
        return res;
    }
    public void helper(int[] candidates, int target, List tmp, int record, List res,
    int index){
        if(record > target) return;
        if(record == target){
            res.add(new ArrayList(tmp));
        }else{
            for(int i = index; i < candidates.length; ++i){
                tmp.add(candidates[i]);
//这里得传入一个index 要不然会有重复.
                helper(candidates, target, tmp, record + candidates[i], res, i);
                tmp.remove(tmp.size() - 1);
            }
        }
    }
```