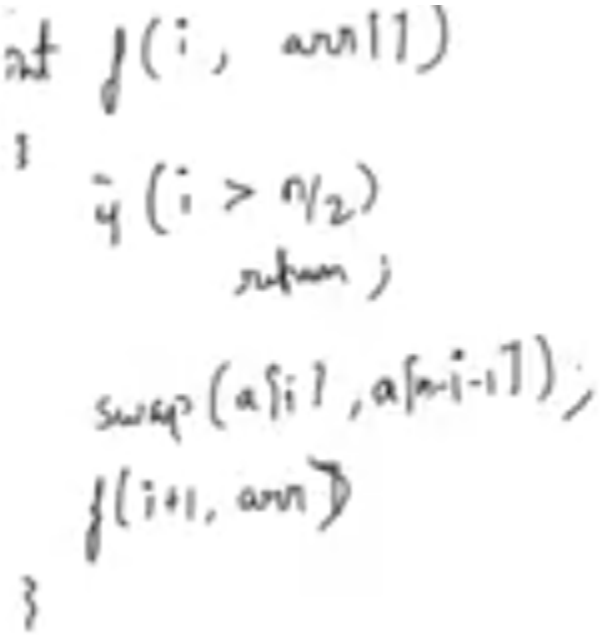
Time complexity for recursion: Total no. of calls=x^n
Where x=No. Of recursive calls inside function,n=depth of recursion tree
OR
Maximum possible no. of solutions

Space complexity of the recursive algorithm is proportional to the maximum depth of recursion tree generated. If each function call of recursive algorithm takes O(m) space and if the maximum depth of recursion tree is 'n' then space complexity of recursive algorithm would be O(nm).

To solve recursive problem,make a recursive tree i.e. all recursive calls when

Multiple recursion:A function is called recursively multiple times inside a function.
A multiple recursive function has three parts:

1. Base case
2. Function calls
3. Recurrence relation



Reverse array using recursion:

Check palindrome using recursion.

Print all possible subsequences using recursion(we had to use backtracking):

```
void func(int ind, vector<int> &ds, int arr[], int n) {
    if(ind == n) {
        // print statement
        for(auto it : ds) {
            cout << it << " ";
        }
        cout << endl;
        return;
    }

    ds.push_back(a[ind]);
    // pick it
    func(ind+1, ds, arr, n);
    ds.pop_back();

    // not pick
    func(ind+1, ds, arr, n);
}
```

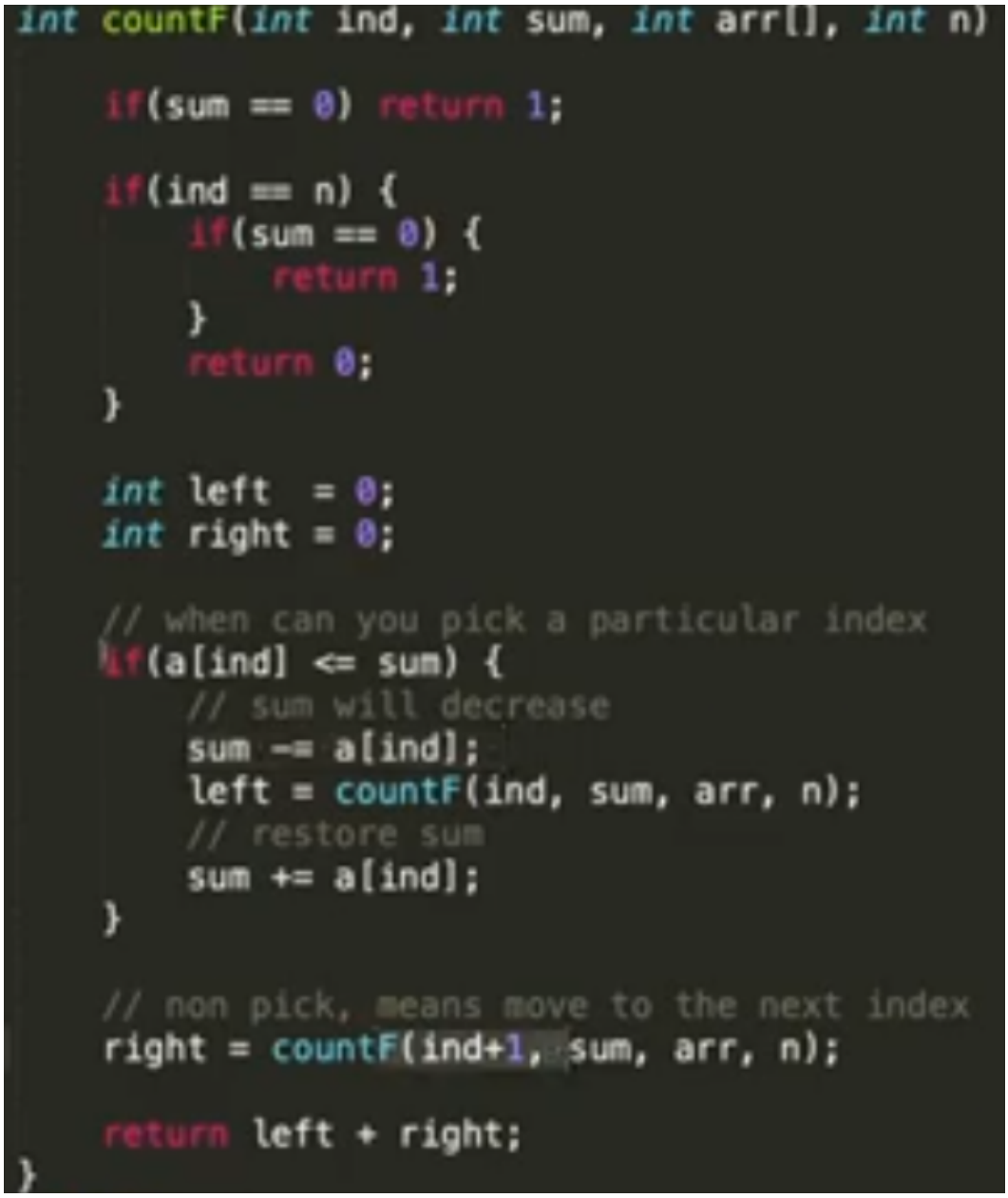
Print all subsets with sum divisible by k.

Count all subsets with sum divisible by k without using count as global/local variable.
Always use this technique when counting.

```
f( 0, sum)
{
  if(ind == n)
  {
    if(sum-1.k==0)
      return 1;
    else return 0;
  }
  sum += a[ind];
  l = f(1, sum);
  sum -= a[ind];
  r = f(1, sum);
  return l+r;
}
```

To solve array using recursion, solve by index based.

Find all unique subsets with sum equal to S. A no. can be selected more than once.



```
int countF(int ind, int sum, int arr[], int n)

    if(sum == 0) return 1;

    if(ind == n) {
        if(sum == 0) {
            return 1;
        }
        return 0;
    }

    int left  = 0;
    int right = 0;

    // when can you pick a particular index
    if(a[ind] <= sum) {
        // sum will decrease
        sum -= a[ind];
        left = countF(ind, sum, arr, n);
        // restore sum
        sum += a[ind];
    }

    // non pick, means move to the next index
    right = countF(ind+1, sum, arr, n);

    return left + right;
}
```

//boolean recursive function is when only one solution has to printed and
//void recursive solution is when all solutions are to be printed

Print all permutations of a string

Striver solution:

https://leetcode.com/problems/subsets-ii/

To avoid adding duplicate subsets,let say there are n duplicates of a no.,we can add 0 or 1 or 2…or n elements of this to our subset.Out of n elements,if we have picked kth element,we can pick either k+1th element or not pick k+1th element,but if we have not picked kth element,we can't pick k+1th element;