1. Add Two Numbers

class Solution {

public int sum(int num1, int num2) {

int z = num1 + num2;

return z;

}

}

1. Smallest even multiple

class Solution {

    public int smallestEvenMultiple(int n) {

         int i,a,b,c;

       for(i=1;i<=1000000000;i++){

          if(i%n==0 && i%2==0)

          break;

       }

return I;

}

}

1. Sum mutiples

class Solution {

    public int sumOfMultiples(int n) {

        int sum=0;

        for(int i=1;i<=n;i++){

            if(i%3==0 || i%5==0 || i%7==0){

                sum+=i;

            }

        }

        return sum;

    }

}

1. Fibonacci number

class Solution {

public int fib(int n) {

if(n<=1)

return n;

int[] dp=new int[n+1];

dp[0]=0;

dp[1]=1;

for(int i=2;i<=n;i++){

dp[i]=dp[i-1]+dp[i-2];

}

return dp[n];

}

}

1. Reverse string

class Solution {

public void reverseString(char[] s) {

int l = 0;

int r = s.length -1;

while(l < r){

char temp = s[l];

s[l] = s[r];

s[r] = temp;

l++;

r--;

}

}

}

1. Power of two

class Solution {

public boolean isPowerOfTwo(int n) {

if(n == 1){

return true;

}

else if(n == 0){

return false;

}

else if(n%2 != 0){

return false;

}

else{

return isPowerOfTwo(n/2);

}

}

}

1. Subsets

class Solution {

public List<List<Integer>> subsets(int[] nums) {

Arrays.sort(nums);

List<List<Integer>>ans=new ArrayList<>();

solve(0,nums,new ArrayList<>(),ans);

return ans;

}

public void solve(int ind,int[]nums,ArrayList<Integer>temp,List<List<Integer>>ans){

ans.add(new ArrayList<>(temp));

for(int i=ind;i<nums.length;i++){

if(i!=ind && nums[i]==nums[i-1]) continue;//for duplicate checking

temp.add(nums[i]);

solve(i+1,nums,temp,ans);

temp.remove(temp.size()-1);

}

}

}

1. Combinations

class Solution {

public List<List<Integer>> combine(int n, int k) {

List<List<Integer>> allCombinations = new ArrayList<>();

f(1, n, k, allCombinations, new ArrayList<>());

return allCombinations;

}

private void f(int i, int n, int k, List<List<Integer>> allCombinations, ArrayList<Integer> list){

if (list.size() == k){

allCombinations.add(new ArrayList<>(list));

return;

}

for (int j = i; j <= n; j++){

list.add(j);

f(j + 1, n, k, allCombinations, list);

list.remove(list.size() - 1);

}

}

}

1. Combination sum

class Solution {

public List<List<Integer>> combinationSum(int[] candidates, int target) {

List<List<Integer>> result = new ArrayList<>();

getList(0, candidates, target, result, new ArrayList<Integer>());

return result;

}

public void getList(int i, int[] candidates, int target, List<List<Integer>> result, ArrayList<Integer> list)

{

if(target == 0)

{

result.add(new ArrayList<Integer>(list));

return;

}

if(i == candidates.length)

return;

int sub = target - candidates[i];

if(sub >= 0)

{

list.add(candidates[i]);

getList(i, candidates, target - candidates[i], result, list);

list.remove(list.size() - 1);

}

getList(i+1, candidates, target, result, list);

}

}

1. Permutations

class Solution {

public List<List<Integer>> combinationSum(int[] candidates, int target) {

List<List<Integer>> result = new ArrayList<>();

getList(0, candidates, target, result, new ArrayList<Integer>());

return result;

}

public void getList(int i, int[] candidates, int target, List<List<Integer>> result, ArrayList<Integer> list)

{

if(target == 0)

{

result.add(new ArrayList<Integer>(list));

return;

}

if(i == candidates.length)

return;

int sub = target - candidates[i];

if(sub >= 0)

{

list.add(candidates[i]);

getList(i, candidates, target - candidates[i], result, list);

list.remove(list.size() - 1);

}

getList(i+1, candidates, target, result, list);

}

}