Program 1.

https://leetcode.com/problems/build-array-from-permutation/

**Accepted Solution :**

```
class Solution {
        public int[] buildArray(int[] nums) {

        int[] array = new int[nums.length];

        for(int itr=0;itr<nums.length;itr++){

        array[itr]=nums[nums[itr]];
        }

        return array;
        }
}
```

Program 2.

https://leetcode.com/problems/can-place-flowers/

**Tried solution 1 :**

```
class Solution {

        public boolean canPlaceFlowers(int[] flowerbed, int n) {

        int plantedCount = 0;

        for(int itr=0 ; itr<flowerbed.length; itr++){

                if(flowerbed[itr]==1){
```

```
                plantedCount+=1;

            }
        }

        int totalpossiblePlant = flowerbed.length % 2==0 ? flowerbed.length/2 :
(flowerbed.length/2)+1;
        int possiblePlant = totalpossiblePlant - plantedCount;

        if(possiblePlant>=0 && n<=possiblePlant){

                return true;


        }else{

                return false;

        }

        }

}
```

**Accepted Solution :**

```
class Solution {

        public boolean canPlaceFlowers(int[] flowerbed, int n) {

        List<Integer> possible  = new ArrayList<Integer>();
        int possibleCount = 0;
        for(int itr = 0 ; itr<flowerbed.length ; itr++){
```

```
        // at first position check forward
        if(itr==0){
        if(flowerbed[itr]!=1){
                try{
                if(flowerbed[itr+1]==0){

                possible.add(itr);
                possibleCount++;
                }
                }catch(Exception e){
                possibleCount++;
                }

        }
        // at middle posistion check both side
        }else if(itr>0 && itr<flowerbed.length-1){
        if(flowerbed[itr]!=1){
                if(flowerbed[itr+1]==0 && flowerbed[itr-1]==0){
                if(possible.isEmpty()){
                        possible.add(itr);
                        possibleCount++;
                }else if(!possible.contains(itr+1) &&
!possible.contains(itr-1)){
                        possible.add(itr);
                        possibleCount++;
                }
                }

        }
        // at last position check backward
        }else{
        if(flowerbed[itr]!=1){
        if(flowerbed[itr-1]==0)
                if(!possible.contains(itr-1)){
```

```
                possibleCount++;
                }

            }
            }
        }

        if(possibleCount>=n){

        return true;

        }else{

        return false;
        }


        }

}
```

Program 3.

https://leetcode.com/problems/maximum-subarray/

**Try 1 : (Time exceeded)**

```
class Solution {

        public int maxSubArray(int[] nums) {

        int max = nums[0];
        int sum = 0;
        for(int itr=0;itr<nums.length;itr++){

        sum = nums[itr];

        if(sum>max){

                max = sum;
        }

        for(int jtr = itr+1 ; jtr<nums.length ; jtr++){

                sum = sum +  nums[jtr];

                if(sum>max){
                max=sum;
                }

        }

        }

        return max;


        }
}
```

**Accepted Solution :**

```
class Solution {

        public int maxSubArray(int[] nums) {

        int max = Integer.MIN_VALUE;
        int sum = 0;
        for(int itr=0;itr<nums.length;itr++){

        sum = sum + nums[itr];

        if(sum>max){

                max = sum;
        }

        if(sum<0){

                sum = 0;
        }

        }

        return max;
        }
}
```

Program 4.

[https://leetcode.com/problems/set-mismatch/](https://leetcode.com/problems/set-mismatch/)

Try 1:

```
class Solution {
        public int[] findErrorNums(int[] nums) {

        for(int itr = 0 ; itr<nums.length ; itr++){

        for(int jtr = itr+1 ; jtr<nums.length; jtr++){

                if(nums[itr]==nums[jtr]){

                int [] arr = new int[]{nums[itr],nums[itr]+1};
                return arr;

                }

        }

        }

        return new int[0];
        }
}
```

Accepted Solution :

```
class Solution {

        public int[] findErrorNums(int[] nums) {

        int [] hash = new int[nums.length];
        int duplicate = 0;
        for(int itr = 0 ; itr<nums.length ; itr++){

        /// hash function
        int ans = nums[itr]-1;
        if(hash[ans]==nums[itr]){
                duplicate = nums[itr];
        }
        hash[ans]=nums[itr];



        }

        for(int itr = 0;itr<hash.length;itr++){

        if(hash[itr]==0){
            return new int[]{duplicate,itr+1};

        }

        }

        return new int[0];
        }
}
```

Program 5.

https://leetcode.com/problems/pascals-triangle/

Try 1 :

```
class Solution {

        public List<List<Integer>> generate(int numRows) {

        List<List<Integer>> returnList = new ArrayList<List<Integer>>();
        List<Integer> firstNumber = new ArrayList<>();
        firstNumber.add(1);
        returnList.add(firstNumber);
        int product = 1;
        for(int itr = 1 ; itr<numRows ; itr++){

        List<Integer> number = new ArrayList<>();
        product = product*11;

        int temp = product;
        while(temp!=0){

                int rem = temp%10;

                number.add(rem);

                temp = temp/10;
        }

        Collections.reverse(number);
        returnList.add(number);

        }

        return returnList;

        }
}
```

Accepted Solution :

```
class Solution {

        public List<List<Integer>> generate(int numRows) {

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



        for(int itr = 0 ; itr < numRows ; itr++ ){

        List<Integer> innerList = new ArrayList<Integer>();
        int innerLength = itr+1;

        innerList.add(1);
        if(innerLength>2){

                List<Integer> prevList = returnList.get(itr-1);
                int prevListLength = prevList.size();
                int prevItr = 0;
                int currentIndex = 1;
                while(currentIndex<innerLength-1){


innerList.add(currentIndex,prevList.get(prevItr)+prevList.get(prevItr+1));
                currentIndex++;
                prevItr++;
                }
        }
        if(itr!=0)
                innerList.add(innerLength-1,1);
        returnList.add(innerList);

        }


        return returnList;

        }
}
```