4th June 2025 :

Problem 1: Subarray GCD

Approaches:

- Map from GCD - frequency for all subarrays ending at each index.
- for each position i, track all possible GCDs of subarrays ending at i.
- for each gcdval, update gcd(gcdval, A[i]).
- if any such GCD becomes k, add its frequency to result.

pseudo code:

```
count = 0
for(int i=0; i<n; i++)
{   Map<Integer, Integer> currGcdMap = new HashMap<>
 for(Map.Entry<Integer, Integer> entry :
    prevGcdMap.entrySet()) {
      int gcdVal = gcd(entry.getKey(), A[i]);
 currGcdMap.put(gcdVal, currGcdMap.getOrDefault(gcdVal,
  0) + entry.getValue());
    }
  currGcdMap.put(A[i], currGcdMap.getDefault
   (A[i],0) + 1);
  if (currGcdMap.containsKey(k)){
       count += currGcdMap.get(k);
  }
     prevGcdMap = currGcdMap;
  }
     return count;
}
```

Dry run: A = [2, 4, 6, 2, 8] , k = 2

[2] → 2
[2,4] → 2
[4] → 4
GCD = 2

[4,6] → 2
[6] → 6
[6,2] → 2

[2] → 2
[2,8] → 2
[8] → 8

count → total = 7

Problem 2:

Approaches:- move through array by keeping track of the farthest place you can reach and when to jump.

pseudo code:
```
int n = J.length;
    if (n==1) return 0;
    if (J[0]==0) return -1;
  int jumps = 0;
  int currend = 0; farthest = 0;
for(int i=0; i<n-1; i++){
  farthest = Math.max(farthest, i + J[i]);
  if (i == currend){
      jumps++
        currend = farthest;
    if( currend <= i) return -1;
  }
}
  return currend >= n-1 ? jumps: -1;
}
```

Dry run: Input: [2,3,1,1,4,2]
Output: 3

? index 0: jump to 1 or 2 → choose 1 (jump1)
" 1: can go to 4 → jump to 4 (jump2)
" 4: jump to 5 (last index) → jump 3