

```

1 package problem1;import java.util.Arrays;
2 import java.util.Collections;
3 /* 그리디 문제입니다! 가장 많이 먹는 머리 순으로 순서대로 주면,
4 최대한 양손을 많이 쓸 수 있습니다. 양손으로 먹이를 주는 것이 끝나면
5 나머지는 한손으로 주어야 합니다. 한 손으로 주는 것이 최소가 되게
6 하는것이 핵심!*/
7 class Solution{public int solution(int[]food){
8     int[]sortedFood=Arrays.stream(food).boxed
9     ().sorted(Collections.reverseOrder()).mapToInt(i->i
10    ).toArray();
11     int count = 0; // 급여 횟수
12     int currFood = sortedFood[0]; // 현재 급여중인
13     머리에 급여해야 할 양
14     for (int i = 1; i < sortedFood.length; i
15     ++){
16         int newFood = sortedFood[i]; // 또 다른
17         머리에 급여해야 할 양
18         if (newFood < currFood) { // 급여 중 다른
19         머리의 식사가 모두 끝나면
20             currFood -= newFood; // 현재 급여중인
21             머리에 해당 횟수만큼 급여 완료
22             count += newFood; // 급여 횟수는
23             다른 머리의 식사횟수 만큼이 된다.
24             } else { // 현재 급여중인 머리의 급여가 끝나면
25             count += currFood; // 급여 횟수만큼
26             추가하고
27             currFood = newFood - currFood; //
28             다른 머리가 이제 '현재 급여중인 머리'가 된다.
29             }count += currFood; // 양손으로 줄 수 있는
30             급여를 마치고, 한 손 급여만 가능한 케이스
31         return count;}}class Test {
32     public static void main(String[] args){
33         int[] food = {6, 3, 4, 5};System.out.
34         println(new Solution().solution(food));}}
35 package problem2;/* DP/수학문제입니다. Tabulation 방식으로
36 계산하면서, 현재까지 구한 값 중 작은 값에
37 * 2, *3, *5한 값을 추가해 나갑니다. 값을 추가한 이후에는 2,3,
38 5를 곱할 대상을 더 큰 값으로 바꾸어 줍니다. */
39 class Solution{public int solution(int n){
40     int mult2 = 0, mult3 = 0, mult5 = 0; // 2배, 3배
41     , 5배할 값의 인덱스 초기화

```

```

25     int[] dp = new int[n];dp[0] = 1;for(int i = 1;
    i < n; i++) {int valMult2 = 2*dp[mult2];
26         int valMult3 = 3*dp[mult3];int valMult5 = 5
    *dp[mult5];
27         // 2곱한거 3곱한거 5곱한것 중 가장 작은 것을 i번째 숫자로
    확정
28         dp[i] = Math.min(valMult2, Math.min(
    valMult3, valMult5));
29         // 아래 조건은 else if로 하면 안됩니다. 왜냐하면 여러
    경우에 동시에 일치할 수 있기 때문!
30         // ex) dp[mult2] = 6, dp[mult3] = 4, dp[
    mult5] = 3인 경우,
31         // 2*dp[mult2]와 3*dp[mult3]가 같은 값이므로, 둘
    다 인덱스가 변경되어야 함
32         if(dp[i] == valMult2) // dp[mult2]의 2배와
    일치하는 값이라면, mult2를 다음 값으로 업데이트 mult2++;
33         if(dp[i] == valMult3) // dp[mult3]의
    3배와 일치하는 값이라면, mult3를 다음 값으로 업데이트 mult3++;
34         if(dp[i] == valMult5) // dp[mult5]
    의 5배와 일치하는 값이라면, mult5를 다음 값으로 업데이트 mult5++;
35         return dp[n-1];}}class Test{
36     public static void main(String[] args){int n =
    15;System.out.println(new Solution().solution(n
    ));}}
37 package problem3;/* CCW 알고리즘입니다. (Counter-
    ClockWise) 외적을 이용하여 점의 진행 방향을 알아낼 수 있습니다.
38     * 이러한 알고리즘은 단독으로 출제되기 보다는, 더 큰 문제를
    풀고자 할 때 일부분으로 사용되기도 합니다. 그럴 때 이 알고리즘을 알고
    있는지
39     * 아닌지에 따라 구현 속도와 정확도에 차이가 있겠지요! CCW
    알고리즘에 대한 자세한 내용은 원본 문제와 그 답안을 참고해 주세요.
40     * 문제 출처 : https://www.acmicpc.net/problem/11758*/
41     class Solution { public String solution(int
    [][] points) {
42         int x1 = points[0][0], y1 = points[0][1];
43         int x2 = points[1][0], y2 = points[1][1];
44         int x3 = points[2][0], y3 = points[2][1];
45         int val = (x1 * y2 + x2 * y3 + x3 * y1) - (
    x2 * y1 + x3 * y2 + x1 * y3);
46         if (val == 0) {return "LINE";} else if (val

```

```
46 > 0) {return "CCW";} else {return "CW";}}}  
47     class Test{public static void main(String[]  
    args){  
48         int[][] points={{0,0},{0,10},{10,5}};System  
        .out.println(new Solution().solution(points));}}
```