

```

1 package problem1; /* 가장 큰 숫자 2개를 찾는 문제로,  $O(n)$ 으로
   해결할 수 있는 문제입니다. 모든  $i$ 와  $j$ 를 전수조사할
2  * 필요가 없다는 점을 캐치하는 것이 핵심입니다.*/
3 class Solution{public int solution(int[] nums){
4     int max1 = Integer.MIN_VALUE;
5     int max2 = Integer.MIN_VALUE;
6     for(int i=0;i<nums.length;i++){if(nums[i] >
   max1){max2 = max1;max1 = nums[i];}
7     else if(nums[i] > max2){max2 = nums[i];}}
8     return (max1-1)*(max2-1);}}
9 class Test{public static void main(String[] args){
10     int[] nums = {3, 5, 7, 5};System.out.println(
   new Solution().solution(nums));}}
11 package problem2;import java.util.PriorityQueue;
12 /* 그리디 문제입니다. 양 플레이어에게 서로 가치가 다르더라도, 돌을
   하나 선택함으로써 '내가 얻는 점수' +
13  * '상대가 잃는 점수'는 결국 같기 때문에, 동일한 기준으로 순서대로
   하나씩 뽑아서 결과를 확인하면 됩니다.*/
14 class Solution{public int solution(int[][] value){
15     PriorityQueue<Stone> pq = new PriorityQueue
   <>();int totalValue = 0;
16     // 양 점수의 합을 기준으로 큰 것 부터 뽑는 힙
17     for (int[] v: value) {pq.offer(new Stone(v[
   0], v[1]));}
18     int turn = 0;while (!pq.isEmpty()) {if (
   turn % 2 == 0) {totalValue += pq.poll().value1; //
   플레이어1 점수 획득
19     } else {totalValue -= pq.poll().value2
   ;} //플레이어2 점수 획득(플레이어1의 점수에서 깎음)turn++;
20     // 0이면 0, 0보다 크면 1, 0보다 작으면 -1 출력
21     } return totalValue == 0 ? 0 : (totalValue
   > 0 ? 1 : -1);}}
22 class Stone implements Comparable<Stone>{int value1
   ,value2;public Stone(int value1,int value2){
23     this.value1=value1;this.value2=value2;}
24     @Override public int compareTo(Stone o){return
25     (o.value1+o.value2)-(value1+value2);}}
26 class Test{public static void main(String[] args){
27     int[][] value = {{5, 3}, {6, 9}, {4, 5}, {6
   , 3}, {2, 8}, {5, 4}};System.out.println
28     (new Solution().solution(value));}}

```