```java
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;
import java.util.stream.IntStream;

class GaleShapley {
    public static void main(String[] args){

        Random rnd = new Random();
        rnd.setSeed(3141592);

        for (int i = 500; i <= 10000; i += 500){
            int[][] pefm = generatePef(i, rnd);
            int[][] pefw = generatePef(i, rnd);
            int[] goodness = goodness(pefm, pefw);
            System.out.printf("when n is %d, goodness for M is %d," +
                              "goodness for W is %d. \n",
                               i, goodness[0], goodness[1]);
        }
    }

    public static int[][] galeShapley(int[][] M, int[][] W){
        int[] pairM = new int [M.length];
        int[] pairW = new int [W.length];
        int[] rankM = new int [M.length];
        int[] rankW = new int [W.length];

        Arrays.fill(pairM, -1);
        Arrays.fill(pairW, -1);

        // using Queue to track free m
        Queue<Integer>freeM = new LinkedList<Integer>();
        for (int i = 0; i <  W.length; i++){
            freeM.offer(i);
        }

        while (!freeM.isEmpty()){
            int m = freeM.poll();
            // start searching qualified w from last unmatch w index + 1
            for(int i = rankM[m]; i < W.length; i++){
                int w = M[m][i];
                if (pairW[w] == -1){
                    // if w is free, match (m, w)
                    pairM[m] = w;
                    pairW[w] = m;

                    // note down rank of current match
                    rankM[m] = i + 1;
                    rankW[w] = findIndex(W[w], m) + 1;
                    break;

                }else{
                    for(int j = 0; j < W.length; j++){
                        if (W[w][j] == m){
                            // unmatch w with pairW[w]
                            freeM.offer(pairW[w]);

                            // match w with m
                            pairM[m] = w;
                            pairW[w] = m;

                            // note down rank of current match
                            rankM[m] = findIndex(M[m], w) + 1;
                            rankW[w] = j + 1;
                            i = W.length;
                            break;
                        }
                        if (W[w][j]  == pairW[w]){
                            break;
                        }
                    }
                }
            }
        }
        int[][] ret = {pairM, pairW, rankM, rankW};
        return ret;
    }

    public static int[] permutation(int n, Random rand) {
        int[] arr = IntStream.range(0, n).toArray();
        for (int i = 0; i < n; i++) {
            int j = rand.nextInt(i + 1);
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
        return arr;
    }

    public static int[][] generatePef(int n, Random rand){
        int[][] pef = new int[n][];
        for(int i = 0; i < n; i++){
            int[] cur = permutation(n, rand);
            pef[i] = cur;
        }
        return pef;
    }

    public static int[] goodness(int[][]  M, int[][] W){
        int[][] res = galeShapley(M,W);
        int[] mRank = res[2];
        int[] wRank = res[3];
        int[] ret = {Arrays.stream(mRank).sum(), Arrays.stream(wRank).sum()};
        return ret;
    }

    public static int findIndex(int[] arr, int t) {
        if (arr == null) return -1;
        int len = arr.length;
        int i = 0;
        while (i < len) {
            if (arr[i] == t){
                return i;
            }
            else{
                i++;
            }
        }
        return -1;
    }
}

/*
when n is 500, goodness for M is 3065, goodness for W is 39225.
when n is 1000, goodness for M is 9036, goodness for W is 114134.
when n is 1500, goodness for M is 14274, goodness for W is 242898.
when n is 2000, goodness for M is 17396, goodness for W is 459985.
when n is 2500, goodness for M is 22651, goodness for W is 674116.
when n is 3000, goodness for M is 28297, goodness for W is 947377.
when n is 3500, goodness for M is 32899, goodness for W is 1298732.
when n is 4000, goodness for M is 28437, goodness for W is 2206001.
when n is 4500, goodness for M is 35469, goodness for W is 2509572.
when n is 5000, goodness for M is 49494, goodness for W is 2519637.
when n is 5500, goodness for M is 58113, goodness for W is 2899666.
when n is 6000, goodness for M is 53436, goodness for W is 4048064.
when n is 6500, goodness for M is 64774, goodness for W is 4146344.
when n is 7000, goodness for M is 59914, goodness for W is 5761363.
when n is 7500, goodness for M is 59234, goodness for W is 7175208.
when n is 8000, goodness for M is 70080, goodness for W is 7359127.
when n is 8500, goodness for M is 92816, goodness for W is 6717973.
when n is 9000, goodness for M is 91812, goodness for W is 8012936.
when n is 9500, goodness for M is 111279, goodness for W is 7712503.
when n is 10000, goodness for M is 111487, goodness for W is 8965547.

As the size of the problem increases, the goodness of M increase faster than W.
This is clearly a better result for M, since they have a overall better rank
match in preference list.
*/
```