```
::::::::::::::
srcFiles
::::::::::::::
::::::::::::::
Driver.java
::::::::::::::
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 * Brian Chesko
 * DAA Programming Assignment 3
 * Fully complete and tested, 2019/03/13
 * Modified 2019/03/31
 */
public class Driver {

    public static void main(String[] args) throws IOException {
        System.out.println("Enter matrix size");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in
));
        int n = Integer.parseInt(reader.readLine().trim());
        System.out.println(n);
        short[][] matrix = new short[n][n];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                matrix[i][j] = (short) Integer.parseInt(reader.readLine().trim());
                System.out.printf("%d\t", matrix[i][j]);
            }
            System.out.println();
        }

        // Note: here, I had to resolve the weirdest bug. I was getting
        // different results after merging the solvers into one folder
        // despite not changing any meaningful code.
        // As it turned out, the matrices passed were being modified
        // across solvers which messed up 2/3 of the solvers final
        // results since they shared the same reference.
        Solver sol0 = new SolverP0(matrix);
        Solver sol1 = new SolverP1(deepMatrixCopy(matrix));
        Solver sol2 = new SolverP2(deepMatrixCopy(matrix));

        int[] solution = sol0.solve();
        sol1.solve();
        sol2.solve();

        System.out.println("== Number of partial assignments explored ==");
        System.out.printf("\tP0: %d \tP1: %d \tP2: %d\n",
            sol0.getPartialExploredSize(),
            sol1.getPartialExploredSize(),
            sol2.getPartialExploredSize());
        System.out.println("==== Number of full assignments explored ===");
        System.out.printf("\tP0: %d \tP1: %d \tP2: %d\n",
            sol0.getFullyExploredSize(),
            sol1.getFullyExploredSize(),
            sol2.getFullyExploredSize());
        System.out.println("=== Total number of assignments explored ===");
        System.out.printf("\tP0: %d \tP1: %d \tP2: %d\n",
            sol0.getTotalExploredSize(),
            sol1.getTotalExploredSize(),
            sol2.getTotalExploredSize());
        System.out.println("Best job assignment is:");
        for (int i = 0; i < n; i++) {
            System.out.printf("Person %d assigned job %d\n", i, solution[i]);
        }
        System.out.printf("Best job assignment cost: %d\n",
            sol0.getSolutionProductivity());
    }

    private static short[][] deepMatrixCopy(short[][] matrix) {
        short[][] copy = new short[matrix.length][matrix[0].length];
        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = matrix[i][j];
            }
        }
        return copy;
    }

}
::::::::::::::
Solver.java
::::::::::::::
public abstract class Solver {
    public abstract int[] solve();
    public abstract long getPartialExploredSize();
    public abstract long getFullyExploredSize();
    public abstract long getTotalExploredSize();
    public abstract int getSolutionProductivity();
}
::::::::::::::
SolverP0.java
::::::::::::::
import java.util.*;
import java.util.function.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.stream.Collectors;

/**
 * Brian Chesko
 * DAA Programming Assignment 2
 * Fully complete and tested, 2019/02/27
 * Modified 2019/03/20
 */
public class SolverP0 extends Solver {
    private short[][] matrix;
    private int size;
    private int highestProductivity;
    private long solutionsExplored;
    private long partialExplored;
    private int[] bestArrangement;

    public SolverP0(short[][] jobEmployeeMatrix) {
        this.matrix = jobEmployeeMatrix;
        this.size = matrix.length;
        this.highestProductivity = 0;
        this.solutionsExplored = 0;
        this.partialExplored = 0;
        this.bestArrangement = null;
    }
```

```java
    /**
     * Reinitialize the solver to work with the specified matrix.
     * @param jobEmployeeMatrix The new matrix to solve.
     */
    public void setJobEmployeeMatrix(short[][] jobEmployeeMatrix) {
        this.matrix = jobEmployeeMatrix;
        this.size = matrix.length;
        this.highestProductivity = 0;
        this.solutionsExplored = 0;
        this.partialExplored = 0;
        this.bestArrangement = null;
    }

    /**
     * Finds the most productive set of jobs assignments given the job
     * employee matrix.
     * @return an array containing the most productive arrangement, such that
     * array[i] is the best job for employee i.
     */
    public int[] solve() {
        // Already solved for this matrix, return previous solution.
        if (bestArrangement != null)
            return bestArrangement;

        //int positionsPerThread = 2;
        int numThreads = 4;//(size + positionsPerThread - 1) / positionsPerThread;
        int positionsPerThread = (size + numThreads - 1) / numThreads;
        ExecutorService threadPool = Executors.newFixedThreadPool(numThreads);
        // Create one thread for every set of 'positionsPerThread' starting positi
ons
        for (int threadNo = 0; threadNo < numThreads; threadNo++) {
            int start = threadNo * positionsPerThread;
            SolvingPartition partition = new SolvingPartition(
                    start,
                    threadNo == numThreads - 1 ? size : start + positionsPerThread
,
                    this
            );
            // Add the thread to the thread pool
            threadPool.execute(partition);
        }

        // Now all threads have been added to the thread pool,
        // try to close the process and wait until complete
        try {
            threadPool.shutdown();
            while (!threadPool.isTerminated()) {
                threadPool.awaitTermination(60, TimeUnit.SECONDS);
            }
        } catch (InterruptedException e) {
            // Should do something more useful with this but it's fine
            e.printStackTrace();
        }

        return bestArrangement;
    }

    public void setHighestProductivity(int prod) {
        this.highestProductivity = prod;
    }
```

```java
    public void setBestArrangement(int[] arr) {
        this.bestArrangement = arr.clone();
    }

    private class SolvingPartition implements Runnable {
        private int firstEmpStartJob;
        private int firstEmpEndJob;
        private Consumer<SolverP0> endCallback;
        private SolverP0 wrapper;
        private int[] bestArrangement;
        private long solutions;
        private long partialSolutions;
        private int highestProductivity;

        /**
         * @param firstEmpStartJob the index of the first employee tree to check
         * @param firstEmpEndJob non inclusive index of last employee tree
         */
        SolvingPartition(int firstEmpStartJob, int firstEmpEndJob,
                SolverP0 wrapper) {
            this.firstEmpStartJob = firstEmpStartJob;
            this.firstEmpEndJob = firstEmpEndJob;
            this.endCallback = (x) -> {
                if (x.getSolutionProductivity() < this.highestProductivity) {
                    x.setHighestProductivity(highestProductivity);
                    x.setBestArrangement(bestArrangement);
                }
                x.incrementFullExplored(solutions);
                x.incrementPartialExplored(partialSolutions);
            };
            this.solutions = 0;
            this.partialSolutions = 0;
            this.bestArrangement = new int[size];
            this.wrapper = wrapper;
            this.highestProductivity = 0;
        }

        @Override
        public void run() {
            int[] arrangement = new int[size];
            int[] partialProductivities = new int[size];
            boolean[] columnUsed = new boolean[size];
            short[][] matrix = deepMatrixCopy();
            // Ensure prefill arrangement to all unused
            for (int i = 0; i < size; i++) {
                arrangement[i] = -1;
            }

            int emp = 0;
            int job = firstEmpStartJob;

            while (emp < size && emp >= 0) {
                int lastJobToCheck = emp == 0 ? firstEmpEndJob : size;
                int prevJob = arrangement[emp];
                if (prevJob != -1) {
                    // Don't check the same index twice for the same employee,
                    // move to the next job.
                    job = prevJob + 1;
                    // Reset variables tracking the previous setup.
                    columnUsed[prevJob] = false;
                    arrangement[emp] = -1;
                } else if (emp > 0) {
```

```java
                    // Haven't seen this employee before (for this subtree)
                    // so start from beginning of job search.
                    job = 0;
                }
                boolean foundCol = false;
                while (job < lastJobToCheck && !foundCol) {
                    partialSolutions++;
                    if (!columnUsed[job]) {
                        foundCol = true;
                        columnUsed[job] = true;
                        arrangement[emp] = job;
                        if (emp == 0) {
                            partialProductivities[emp] = matrix[emp][job]; //matri
x.getProductivity(emp, job);
                        } else {
                            partialProductivities[emp] = partialProductivities[emp
 - 1] +  matrix[emp][job];//matrix.getProductivity(emp, job);
                        }
                    } else {
                        job++;
                    }
                }

                // We ALWAYS backtrack after the emp == size - 1 iteration.
                // We also ALWAYS backtrack after job == size - 1, but ONLY after
visiting children trees (if necessary).
                if (emp == size - 1) {
                    int partialProductivity = partialProductivities[emp];
                    solutions++;
                    if (partialProductivity > highestProductivity) {
                        highestProductivity = partialProductivity;
                        bestArrangement = arrangement.clone();
                    }
                    // Reset tracking variables for this position
                    columnUsed[arrangement[emp]] = false;
                    arrangement[emp] = -1;
                    emp--;
                } else if (!foundCol) {
                    // Not the last employee, but still need to backtrack.
                    emp--;
                } else {
                    emp++;
                }
            }

            this.endCallback.accept(wrapper);
        }
    }

    /**
     * Converts the list into an int[] array for returning to the main program
     * @param arrayList
     * @return
     */
    private int[] convertToIntArray(List<Integer> arrayList) {
        int[] array = new int[arrayList.size()];
        for (int i = 0; i < array.length; i++) {
            array[i] = arrayList.get(i);
        }
        return array;
    }
```

```java
    /**
     * @return the best job assignment for the current matrix.
     */
    public int[] getBestArrangement() {
        return bestArrangement;
    }

    /**
     * @return the total number of solutions explored while finding the best assig
nment
     */
    public long getExploredSize() {
        return solutionsExplored;
    }

    /**
     * @return the number of partial  solutions explored while finding the
     * best assignment
     */
    public long getPartialExploredSize() {
        return partialExplored;
    }

    /**
     * @return the number of full successful  solutions explored while
     * finding the best assignment
     */
    public long getFullyExploredSize() {
        return solutionsExplored;
    }

    /**
     * @return the total number of solutions explored while finding the best
     * assignment
     */
    public long getTotalExploredSize() {
        return partialExplored + solutionsExplored;
    }

    public void incrementFullExplored(long sol) {
        this.solutionsExplored += sol;
    }

    public void incrementPartialExplored(long partials) {
        this.partialExplored += partials;
    }

    /**
     * @return the overall productivity of the best assignment
     */
    public int getSolutionProductivity() {
        return highestProductivity;
    }

    private short[][] deepMatrixCopy() {
        short[][] copy = new short[matrix.length][matrix[0].length];
        for (int i = 0; i < copy.length; i++) {
            for (int j = 0; j < copy[0].length; j++) {
                copy[i][j] = matrix[i][j];
            }
        }
        return copy;
```

```
    }

}
::::::::::::::
SolverP1.java
::::::::::::::
import java.util.Arrays;

/**
 * Brian Chesko
 * DAA Programming Assignment 3 Pt 1
 * Fully complete and tested, 2019/03/13
 */
public class SolverP1 extends Solver {
    private short[][] matrix;
    private int size;
    private int highestProductivity;
    private long solutionsExplored;
    private long partialExplored;
    private int[] bestArrangement;

    public SolverP1(short[][] jobEmployeeMatrix) {
        this.matrix = jobEmployeeMatrix;
        this.size = matrix.length;
        this.highestProductivity = 0;
        this.solutionsExplored = 0;
        this.partialExplored = 0;
        this.bestArrangement = null;
    }

    /**
     * Reinitialize the solver to work with the specified matrix.
     * @param jobEmployeeMatrix The new matrix to solve.
     */
    public void setJobEmployeeMatrix(short[][] jobEmployeeMatrix) {
        this.matrix = jobEmployeeMatrix;
        this.size = matrix.length;
        this.highestProductivity = 0;
        this.solutionsExplored = 0;
        this.partialExplored = 0;
        this.bestArrangement = null;
    }

    /**
     * Finds the most productive set of jobs assignments given the job
     * employee matrix.
     * @return an array containing the most productive arrangement, such that
     * array[i] is the best job for employee i.
     */
    public int[] solve() {
        // Already solved for this matrix, return previous solution.
        if (bestArrangement != null)
            return bestArrangement;

        bestArrangement = new int[size];
        int[] arrangement = new int[size];
        int[] partialProductivities = new int[size];
        int[] maxSubtreeProductivities = new int[size];
        boolean[] columnUsed = new boolean[size];
        // Ensure prefill arrangement to all unused
        for (int i = 0; i < size; i++) {
            arrangement[i] = -1;
            // Do preprocessing to determine largest productivity of each
            // size subtree.  This lets us prune subtree searches that cannot
            // possibly beat a current max partial assignment.
            int largestVal = Integer.MIN_VALUE;
            for (int val : matrix[i]) {
                if (val > largestVal) {
                    largestVal = val;
                }
            }
            for (int j = 0; j <= i; j++) {
                maxSubtreeProductivities[j] += largestVal;
            }

        }

        int emp = 0;
        int job;

        while (emp < size && emp >= 0) {
            int prevJob = arrangement[emp];
            if (prevJob != -1) {
                // Don't check the same index twice for the same employee,
                // move to the next job.
                job = prevJob + 1;
                // Reset variables tracking the previous setup.
                columnUsed[prevJob] = false;
                arrangement[emp] = -1;
            } else {
                // Haven't seen this employee before (for this subtree)
                // so start from beginning of job search.
                job = 0;
            }
            boolean foundCol = false;

            // If the partial solution + the largest combination after this is les
s
            // than the highest seen, we can't possibly beat it. Skip that subtree
.
            int prodSoFar = emp == 0 ? 0 : partialProductivities[emp - 1];
            if (prodSoFar + maxSubtreeProductivities[emp] > highestProductivity) {
                while (job < size && !foundCol) {
                    if (!columnUsed[job]) {
                        foundCol = true;
                        columnUsed[job] = true;
                        arrangement[emp] = job;
                        partialProductivities[emp] = prodSoFar + matrix[emp][job];
                    } else {
                        job++;
                    }
                }
            } else {
                this.partialExplored++;
            }

            // We ALWAYS backtrack after the emp == size - 1 iteration.
            // We also ALWAYS backtrack after job == size - 1, but ONLY
            // after visiting children trees (if necessary).
            if (emp == size - 1) {
                this.solutionsExplored++;
                int partialProductivity = partialProductivities[emp];
                if (foundCol) {
                    // Save new best solution, if needed
```

```java
                    if (partialProductivity > highestProductivity) {
                        highestProductivity = partialProductivity;
                        for (int i = 0; i < size; i++) {
                            bestArrangement[i] = arrangement[i];
                        }
                    }
                    // Reset tracking variables for this setup so we can search mo
re
                    columnUsed[arrangement[emp]] = false;
                    arrangement[emp] = -1;
                }

                // Reset tracking variables for this position
                emp--;
            } else if (!foundCol) {
                // Not the last employee, but still need to backtrack.
                emp--;
            } else {
                emp++;
            }
        }

        return bestArrangement;
    }

    /**
     * @return the best job assignment for the current matrix.
     */
    public int[] getBestArrangement() {
        return bestArrangement;
    }

    /**
     * @return the number of partial  solutions explored while finding the
     * best assignment
     */
    public long getPartialExploredSize() {
        return partialExplored;
    }

    /**
     * @return the number of full successful  solutions explored while
     * finding the best assignment
     */
    public long getFullyExploredSize() {
        return solutionsExplored;
    }

    /**
     * @return the total number of solutions explored while finding the best
     * assignment
     */
    public long getTotalExploredSize() {
        return partialExplored + solutionsExplored;
    }

    /**
     * @return the overall productivity of the best assignment
     */
    public int getSolutionProductivity() {
        return highestProductivity;
    }
}
::::::::::::::
SolverP2.java
::::::::::::::
import java.util.Arrays;

/**
 * Brian Chesko
 * DAA Programming Assignment 3 Pt 2
 * Fully complete and tested, 2019/03/20
 */
public class SolverP2 extends Solver {
    private short[][] matrix;
    private short[][] swapMatrix;
    private int size;
    private int highestProductivity;
    private long solutionsExplored;
    private long partialExplored;
    private int[] bestArrangement;

    public SolverP2(short[][] jobEmployeeMatrix) {
        this.matrix = jobEmployeeMatrix;
        this.size = matrix.length;
        this.highestProductivity = 0;
        this.solutionsExplored = 0;
        this.partialExplored = 0;
        this.bestArrangement = null;
        this.createAndSwapMatrix();
    }

    /**
     * Reinitialize the solver to work with the specified matrix.
     * @param jobEmployeeMatrix The new matrix to solve.
     */
    public void setJobEmployeeMatrix(short[][] jobEmployeeMatrix) {
        this.matrix = jobEmployeeMatrix;
        this.size = matrix.length;
        this.highestProductivity = 0;
        this.solutionsExplored = 0;
        this.partialExplored = 0;
        this.bestArrangement = null;
        this.createAndSwapMatrix();
    }

    /**
     * Debug method for printing matrices
     */
    public void printMatrices() {
        System.out.println("== Sorted job/employee matrix ==");
        for (short[] row : matrix) {
            for (short val : row) {
                System.out.print(val + "\t");
            }
            System.out.println();
        }
        System.out.println("== Swap matrix ==");
        for (short[] row : swapMatrix) {
            for (short val : row) {
                System.out.print(val + "\t");
            }
            System.out.println();
```

```java
        }
    }

    /**
     * Sorts each row of the matrix in descending productivity order
     * via insertion sort. Insertion sort is used because it’s faster
     * for small n (in fact the standard JDK Arrays.sort method uses
     * insertion sort for n<47) as well as being simple to implement.
     *
     * The swap matrix created stores the original column in the new
     * column, such that swapMatrix[i][j] is the original column that
     * matrix[i][j] corresponded to.
     */
    private void createAndSwapMatrix() {
        swapMatrix = new short[size][size];
        for (int i = 0; i < size; i++) {
            swapMatrix[i][0] = 0; // Initial value
            // Sort row i by insertion sort
            for (short j = 1; j < size; j++) {
                swapMatrix[i][j] = j; // Initial value
                // Store working value in a short just so we don’t
                // have to cast ever.
                short currValue = matrix[i][j];
                // Find ending position for current index j
                int endPos = 0;
                while (endPos < j && matrix[i][endPos] >= currValue) {
                    endPos++;
                }
                // Shift values at [endPos, j - 1] right by 1, working
                // right to left
                for (int k = j - 1; k >= endPos; k--) {
                    matrix[i][k + 1] = matrix[i][k];
                    swapMatrix[i][k + 1] = swapMatrix[i][k];
                }
                // Put working value at its destination
                matrix[i][endPos] = currValue;
                swapMatrix[i][endPos] = j;
            }
        }
    }

    /**
     * Finds the most productive set of jobs assignments given the job
     * employee matrix.
     * @return an array containing the most productive arrangement, such that
     * array[i] is the best job for employee i.
     */
    public int[] solve() {
        // Already solved for this matrix, return previous solution.
        if (bestArrangement != null)
            return bestArrangement;

        bestArrangement = new int[size];
        int[] arrangement = new int[size];
        int[] partialProductivities = new int[size];
        int[] maxSubtreeProductivities = new int[size];
        boolean[] columnUsed = new boolean[size];
        // Ensure prefill arrangement to all unused
        for (int i = 0; i < size; i++) {
            arrangement[i] = -1;
            // Do preprocessing to determine largest productivity of each
            // size subtree.  This lets us prune subtree searches that cannot
            // possibly beat a current max partial assignment.
            int largestVal = Integer.MIN_VALUE;
            for (int val : matrix[i]) {
                if (val > largestVal) {
                    largestVal = val;
                }
            }
            for (int j = 0; j <= i; j++) {
                maxSubtreeProductivities[j] += largestVal;
            }

        }

        int emp = 0;
        int job;

        while (emp < size && emp >= 0) {
            int prevJob = arrangement[emp];
            if (prevJob != -1) {
                // Don’t check the same index twice for the same employee,
                // move to the next job.
                job = prevJob + 1;
                // Reset variables tracking the previous setup.
                columnUsed[swapMatrix[emp][prevJob]] = false;
                arrangement[emp] = -1;
            } else {
                // Haven’t seen this employee before (for this subtree)
                // so start from beginning of job search.
                job = 0;
            }
            boolean foundCol = false;

            // If the partial solution + the largest combination after this is les
s
            // than the highest seen, we can’t possibly beat it. Skip that subtree
.
            int prodSoFar = emp == 0 ? 0 : partialProductivities[emp - 1];
            if (prodSoFar + maxSubtreeProductivities[emp] > highestProductivity) {
                while (job < size && !foundCol) {
                    if (!columnUsed[swapMatrix[emp][job]]) {
                        foundCol = true;
                        columnUsed[swapMatrix[emp][job]] = true;
                        arrangement[emp] = job;
                        partialProductivities[emp] = prodSoFar + matrix[emp][job];
                    } else {
                        job++;
                    }
                }
            } else {
                this.partialExplored++;
            }

            // We ALWAYS backtrack after the emp == size - 1 iteration.
            // We also ALWAYS backtrack after job == size - 1, but ONLY
            // after visiting children trees (if necessary).
            if (emp == size - 1) {
                this.solutionsExplored++;
                int partialProductivity = partialProductivities[emp];
                if (foundCol) {
                    // Save new best solution, if needed
                    if (partialProductivity > highestProductivity) {
                        highestProductivity = partialProductivity;
```

```
                        for (int i = 0; i < size; i++) {
                            bestArrangement[i] = swapMatrix[i][arrangement[i]];
                        }
                    }
                    // Reset tracking variables for this setup so we can search mo
re
                    columnUsed[swapMatrix[emp][arrangement[emp]]] = false;
                    arrangement[emp] = -1;
                }

                // Reset tracking variables for this position
                emp--;
            } else if (!foundCol) {
                // Not the last employee, but still need to backtrack.
                emp--;
            } else {
                emp++;
            }
        }

        return bestArrangement;
    }

    /**
     * @return the best job assignment for the current matrix.
     */
    public int[] getBestArrangement() {
        return bestArrangement;
    }

    /**
     * @return the number of partial  solutions explored while finding the
     * best assignment
     */
    public long getPartialExploredSize() {
        return partialExplored;
    }

    /**
     * @return the number of full successful  solutions explored while
     * finding the best assignment
     */
    public long getFullyExploredSize() {
        return solutionsExplored;
    }

    /**
     * @return the total number of solutions explored while finding the best
     * assignment
     */
    public long getTotalExploredSize() {
        return partialExplored + solutionsExplored;
    }

    /**
     * @return the overall productivity of the best assignment
     */
    public int getSolutionProductivity() {
        return highestProductivity;
    }

}
```

```
*** ./rewrite/testing/: directory ***

::::::::::::::
outFiles
::::::::::::::
::::::::::::::
rewrite/testing/output1
::::::::::::::
Enter matrix size
7
35      10      15      38      16      22      25
2       36      22      7       19      2       8
10      21      8       26      21      12      39
26      32      6       15      29      32      26
35      7       10      30      17      17      21
34      0       38      28      36      21      28
7       15      36      9       36      4       35
== Number of partial assignments explored ==
        P0: 45500       P1: 64  P2: 6
==== Number of full assignments explored ===
        P0: 5040        P1: 31  P2: 1
=== Total number of assignments explored ===
        P0: 50540       P1: 95  P2: 7
Best job assignment is:
Person 0 assigned job 3
Person 1 assigned job 1
Person 2 assigned job 6
Person 3 assigned job 5
Person 4 assigned job 0
Person 5 assigned job 2
Person 6 assigned job 4
Best job assignment cost
        P0: 254         P1: 254         P2: 254
::::::::::::::
rewrite/testing/output2
::::::::::::::
Enter matrix size
3
9       7       9
7       5       0
6       5       0
== Number of partial assignments explored ==
        P0: 24  P1: 2   P2: 2
==== Number of full assignments explored ===
        P0: 6   P1: 6   P2: 4
=== Total number of assignments explored ===
        P0: 30  P1: 8   P2: 6
Best job assignment is:
Person 0 assigned job 2
Person 1 assigned job 0
Person 2 assigned job 1
Best job assignment cost
        P0: 21  P1: 21  P2: 21
::::::::::::::
rewrite/testing/output3
::::::::::::::
Enter matrix size
12
376     826     969     461     59      834     550     81      9       172     70
4       71
475     344     62      366     631     992     283     389     372     718     85
```

```
6       22
332     826     716     642     855     436     258     583     262     227     44
673
62      946     106     423     119     810     494     946     506     908     31
2       137
900     947     526     272     665     735     294     350     913     362     34
4       768
150     602     352     412     830     748     437     244     695     543     66
7       166
353     513     112     211     421     776     348     674     724     227     29
8       741
962     945     91      875     659     787     996     810     390     700     57
4       572
448     12      816     143     555     483     661     261     996     773     47
2       769
902     173     443     978     752     742     719     714     687     163     94
1       698
950     937     508     692     989     83      264     438     95      80      93
3       2
915     595     615     911     720     88      681     622     613     476     60
0       365
== Number of partial assignments explored ==
        P0: 7242208140  P1: 40280       P2: 4153
==== Number of full assignments explored ===
        P0: 479001600   P1: 4376        P2: 356
=== Total number of assignments explored ===
        P0: 7721209740  P1: 44656       P2: 4509
Best job assignment is:
Person 0 assigned job 2
Person 1 assigned job 5
Person 2 assigned job 1
Person 3 assigned job 7
Person 4 assigned job 8
Person 5 assigned job 4
Person 6 assigned job 11
Person 7 assigned job 6
Person 8 assigned job 9
Person 9 assigned job 3
Person 10 assigned job 10
Person 11 assigned job 0
Best job assignment cost
        P0: 10812       P1: 10812       P2: 10812
::::::::::::::
rewrite/testing/output4
::::::::::::::
Enter matrix size
10
41      678     642     857     189     533     928     492     480     94
28      83      631     788     489     520     709     697     150     319
617     66      815     643     435     702     104     29      800     556
517     841     235     160     50      424     693     978     268     173
73      296     608     704     436     97      576     146     794     726
817     763     144     633     406     932     687     862     961     487
419     478     680     654     638     731     430     683     61      699
856     134     995     816     190     432     913     766     930     59
845     747     822     989     732     581     921     772     443     234
259     862     713     940     868     703     671     299     387     732
== Number of partial assignments explored ==
        P0: 46023410    P1: 8064        P2: 1985
==== Number of full assignments explored ===
        P0: 3628800     P1: 1398        P2: 184
=== Total number of assignments explored ===
```

```
        P0: 49652210    P1: 9462        P2: 2169
Best job assignment is:
Person 0 assigned job 6
Person 1 assigned job 3
Person 2 assigned job 8
Person 3 assigned job 7
Person 4 assigned job 9
Person 5 assigned job 5
Person 6 assigned job 4
Person 7 assigned job 2
Person 8 assigned job 0
Person 9 assigned job 1
Best job assignment cost
        P0: 8492        P1: 8492        P2: 8492
::::::::::::::
rewrite/testing/output5
::::::::::::::
Enter matrix size
11
225     3057    9142    433     5757    7741    1672    9780    4533    1512    71
33
9896    1381    6868    8280    251     1772    7739    9789    3723    5267    44
90
293     428     6332    5247    4461    799     904     2309    8877    1129    17
18
8020    7914    7475    2113    5939    3607    2998    7451    741     2895    88
32
3961    7527    9083    5734    5266    5224    5809    6885    9715    2454    36
66
2399    7701    8127    3198    4957    6788    2076    2439    8506    6448    67
05
5982    4913    2644    5941    7911    95      6682    806     5279    6996    83
34
4363    2730    9952    5939    8539    3190    2006    993     6856    4406    50
46
4983    7604    6356    8124    6032    8795    6630    8832    5500    8964    37
45
4497    4906    1657    4592    7940    8815    9872    4936    3501    587     76
66
9806    6526    2557    2996    8533    9902    9852    9291    1301    1187    68
95
== Number of partial assignments explored ==
        P0: 554887432   P1: 12185       P2: 1455
==== Number of full assignments explored ===
        P0: 39916800    P1: 528         P2: 15
=== Total number of assignments explored ===
        P0: 594804232   P1: 12713       P2: 1470
Best job assignment is:
Person 0 assigned job 7
Person 1 assigned job 0
Person 2 assigned job 3
Person 3 assigned job 10
Person 4 assigned job 8
Person 5 assigned job 1
Person 6 assigned job 4
Person 7 assigned job 2
Person 8 assigned job 9
Person 9 assigned job 6
Person 10 assigned job 5
Best job assignment cost
        P0: 97772       P1: 97772       P2: 97772
::::::::::::::
```

```
rewrite/testing/output6
::::::::::::::
Enter matrix size
13
667     909     127     345     483     390     981     230     76      392     94
3       206     804
842     131     111     258     866     851     966     322     71      849     46
7       221     573
949     226     451     389     228     470     299     708     816     134     98
797     365
526     541     660     732     698     503     863     809     113     729     12
431     51
83      280     870     305     205     819     883     8       208     463     47
9       859     171
295     994     621     444     711     147     337     723     879     35      57
8       94      845
691     823     209     122     226     293     754     96      950     960     91
5       833     968
476     296     799     335     820     446     681     441     242     392     94
1       580     116
820     615     694     267     812     386     90      22      860     317     31
5       615     765
617     575     681     450     895     509     98      47      844     270     49
3       526     712
736     918     5       316     386     825     931     81      444     96      81
9       535     118
679     204     785     294     969     402     221     2       204     117     51
1       654     164
708     925     9       234     989     97      504     994     413     243     17
1       697     324
== Number of partial assignments explored ==
        P0: 101734972118        P1: 242820      P2: 72593
==== Number of full assignments explored ===
        P0: 6227020800  P1: 9854        P2: 3339
=== Total number of assignments explored ===
        P0: 107961992918        P1: 252674      P2: 75932
Best job assignment is:
Person 0 assigned job 6
Person 1 assigned job 9
Person 2 assigned job 0
Person 3 assigned job 3
Person 4 assigned job 11
Person 5 assigned job 1
Person 6 assigned job 12
Person 7 assigned job 10
Person 8 assigned job 8
Person 9 assigned job 4
Person 10 assigned job 5
Person 11 assigned job 2
Person 12 assigned job 7
Best job assignment cost
        P0: 11632       P1: 11632       P2: 11632
::::::::::::::
rewrite/testing/output7
::::::::::::::
Enter matrix size
7
35      10      15      38      16      22      25
2       36      22      7       19      2       8
10      21      8       26      27      12      19
26      12      6       15      29      2       16
15      7       10      30      17      17      21
34      0       38      28      36      21      28
7       15      10      9       6       4       25
== Number of partial assignments explored ==
        P0: 45500       P1: 94  P2: 80
==== Number of full assignments explored ===
        P0: 5040        P1: 28  P2: 7
=== Total number of assignments explored ===
        P0: 50540       P1: 122         P2: 87
Best job assignment is:
Person 0 assigned job 3
Person 1 assigned job 1
Person 2 assigned job 4
Person 3 assigned job 0
Person 4 assigned job 5
Person 5 assigned job 2
Person 6 assigned job 6
Best job assignment cost
        P0: 207         P1: 207         P2: 207
::::::::::::::
rewrite/testing/output7_1
::::::::::::::
Enter matrix size
7
35      10      15      38      16      22      25
2       36      22      7       19      2       8
34      0       38      28      36      21      28
15      7       10      30      17      17      21
10      21      8       26      27      12      19
26      12      6       15      29      2       16
7       15      10      9       6       4       25
== Number of partial assignments explored ==
        P0: 45500       P1: 108         P2: 104
==== Number of full assignments explored ===
        P0: 5040        P1: 30  P2: 34
=== Total number of assignments explored ===
        P0: 50540       P1: 138         P2: 138
Best job assignment is:
Person 0 assigned job 0
Person 1 assigned job 1
Person 2 assigned job 2
Person 3 assigned job 5
Person 4 assigned job 3
Person 5 assigned job 4
Person 6 assigned job 6
Best job assignment cost
        P0: 207         P1: 207         P2: 207
::::::::::::::
rewrite/testing/output7_2
::::::::::::::
Enter matrix size
7
7       15      10      9       6       4       25
26      12      6       15      29      2       16
10      21      8       26      27      12      19
15      7       10      30      17      17      21
34      0       38      28      36      21      28
2       36      22      7       19      2       8
35      10      15      38      16      22      25
== Number of partial assignments explored ==
        P0: 45500       P1: 929         P2: 96
==== Number of full assignments explored ===
        P0: 5040        P1: 515         P2: 19
```

```
=== Total number of assignments explored ===
        P0: 50540       P1: 1444        P2: 115
Best job assignment is:
Person 0 assigned job 6
Person 1 assigned job 0
Person 2 assigned job 4
Person 3 assigned job 5
Person 4 assigned job 2
Person 5 assigned job 1
Person 6 assigned job 3
Best job assignment cost
        P0: 207         P1: 207         P2: 207
```