```
// numChild = number of children
// numClothes = number of clothes
// childSizes = child sizes array of length numChild
// clothesSizes = cloth sizes array of length numClothes,
// clothSeasons = cloth seasons array of length numClothes,
// rByN = the fair share of each child
// CandidateSolution = The base data structure for a candidate solution that contains the D-value for the
                       distribution and the corresponding price matrix
// INITIALIZE-PRICE-MATRIX (numChild,numClothes) = Initializes the 2D array that maps children to all eligible
clothes.
 -   P[i,j] - If a cloth[j] is eligible to be mapped to a child[i], then the cloth price [j] is mapped in the
     cell P[i,j] otherwise price value of 0 is mapped.
 -   Note that one cloth can be eligible to be mapped to more than one child.
 // INITIALIZE-2D-ARRAY-WITH-ZEROS (rowSize, colSize) = Initializes and returns a 2D array with values set to 0
// SET-SOLUTION-OF-OUT-OF-BOUNDS () = set an out of bound solution for the ineligible candidate solutions
// COMPUTE-DISTRIBUTION-SUM () = Computes the overall distribution absolute value for a candidate solution
```

```
MAIN ()
let assignment[1…numChild, 1….numClothes] be an 2D matrix used to finally assign the clothes to children
P[1…numChild, 1….numClothes] = INITIALIZE-PRICE-MATRIX(numChild, numClothes)
CandidateSolution optimalGlobalMinimumSolution =
        MINIMUM-FAIR-PRICE-DIFFERENCE (P, numChild, numClothes, rByN, clothSeasons)
for i = 0 to numChild
    for j = 0 to numClothes
        // assign a value either 0 or 1 based on final solutions's price matrix
        assignment[i][j] = min (1, optimalGlobalMinimumSolution.priceMatrix[i][j]))
```

```
MINIMUM-FAIR-PRICE-DIFFERENCE(P, numChild, numClothes, rByN, clothSeasons)
CandidateSolution globalMinimumSolution
globalMinimumSolution.Dvalue = ∞
for childIx = 0 to numChild
    baseMatrixPath[1…numChild, 1…numClothes] =
                    INITIALIZE-2D-ARRAY-WITH- ZEROS (numChild, numClothes)
    solutionFromRecursion = GET-MAIN-FAIR-PRICE-RECURSION-FN-HELPER (P, numChild, numClothes,
                childIx, clothJx, baseMatrixPath, rByN, globalMinimumSolution, clothSeasons)
        if solutionFromRecursion.Dvalue < globalMinimumSolution.Dvalue
            globalMinimumSolution.priceMatrix = solutionFromRecursion.priceMatrix
            globalMinimumSolution.Dvalue = solutionFromRecursion.Dvalue
// return only after exploring all candidate solutions in for loop
return globalMinimumSolution    // global optimal minimum distribution
```

```
GET-FAIR-PRICE-RECURSION-FN-HELPER(P, numChild, numClothes,
                              currIx, currJx, currMatrixPath, rByN, globalMinimumSolution, clothSeasons)
if (currIx < 0) or (currIx > (numChild) - 1) or (currJx < 0)
    return SET-SOLUTION-OF-OUT-OF-BOUNDS()
else if (currJx <= numClothes - 1)
    if (currPriceMatrix[currIx][currJx] == 0)
        return SET-SOLUTION-OF-OUT-OF-BOUNDS()
    for i = 0 to numChild
        newMatrixPath = INITIALIZE-2D-ARRAY-WITH-ZEROS (numChild, numClothes)
        // carryforward current matrix path to next recursion step
        copiedNewMatrixPath = copyMatrixToMatrix(currMatrixPath,newMatrixPath,numChild,numClothes)
        minTopBottomSolution =
            MINIMUM (GET-MAIN-FAIR-PRICE-RECURSION-FN-HELPER (currPriceMatrix,
                        numChild,numClothes, currIx + idx,currJx + 1, copiedNewMatrixPath,
                        rByNRatio, currGlobalOptimum, clothSeasons),
            GET-MAIN-FAIR-PRICE-RECURSION-FN-HELPER (currPriceMatrix, numChild,numClothes,
                        currIx - idx - 1, currJx + 1,  copiedNewMatrixPath, rByNRatio,
                        currGlobalOptimum, clothSeasons))
    if minTopBottomSolution.Dvalue >=0 and minTopBottomSolution.Dvalue < globalMinimumSolution.Dvalue
        globalMinimumSolution.Dvalue = minTopBottomSolution.Dvalue
        globalMinimumSolution.priceMatrix = minTopBottomSolution.priceMatrix
    else
    // full path explored
  return BUILD-SOLUTION-FROM-ELIGIBLE-PATH(currMatrixPath, numChild,
                                        numClothes, rByN, clothSeasons)
```

```
BUILD-SOLUTION-FROM-ELIGIBLE-PATH(matrixSolutionPath, numChild, numClothes, rByN, clothSeasons)
for i = 0 to numChild
    atleastWinter = false
    atleastSummer = false
    for j = 0 to numClothes
        if matrixSolutionPath[i][j] > 0
            if clothSeasons[j] = 'w'
                atleastWinter = True
            if clothSeasons[j] == 's'
                atleastSummer = True
    if atleastWinter = True and atleastSummer = False
        // atleast one winter cloth or summer cloth constraint failed
        return SET-SOLUTION-OF-OUT-OF-BOUNDS()
    CandidateSolution eligibleSolution
    eligibleSolution.Dvalue =
            COMPUTE-DISTRIBUTION-SUM(matrixSolutionPath,numChild, numClothes, rByN)
    eligibleSolution.priceMatrix = matrixSolutionPath
  return eligibleSolution
```