**2. Holiday Special - Putting Shifts Together**

##### (a) Describe the optimal substructure of this problem.

The optimal substructure of this problem is to find the most chef with the most back to back steps and choose that chef. After choosing that chef we eliminate the steps he does, you then find the next chef with the most back to back steps that are not done by the previous chefs, adding each chef to the optimal solution.

#### (b) Describe the greedy algorithm in plain words that could find an optimal way to schedule the volunteers for one recipe.

The greedy algorithm that could find an optimal way to schedule the volunteers for one recipe would be to find the chef with the most back to back steps and assign him those steps for that recipe, this eliminating those steps from the open steps. Then find the next chef with the most back to back steps and assign him those steps for the recipe, eliminating those steps from the open steps. Repeat this process until all steps are done.

#### (c) Code your greedy algorithm in the file "HolidaySpecial.java" under the "makeShifts" method where it says "Your code here". Read through the documentation for that method. Note that I've already set up everything necessary for the provided test cases. Do not touch the other methods except possibly adding another test case to the main method. Run the code without your implementation first and you should see this:

Done.

#### (d) What is the runtime complexity of your greedy algorithm? Again, you don't need to factor in the setup of the signup table, just your scheduling algorithm.

M = #of cooks

N= # of steps

The runtime complexity of our greedy algorithm is O(M\*N) + O(2N). M\*N because we have a nested for look that looks at each cook and their steps. Plus 2n because our while would run N times if each step theoretically did 1 step. The last for look would run N if each chef only did 1 step.

#### (e) In your write-up file, based on your answer to part b, give a full proof that your greedy algorithm returns an optimal solution.

Suppose our algorithm from answer B produces this Solution, and the optimal solution is:

Algo: S1+S2+S3+.…+SN

Opt: W1+W2+W3+….WQ

Let i be the first index where Si and Wi not be equal to each other. By design of our algorithm, S1 should have the longest number of back to back steps. Which means W1<S1. Which means we can replace W1 with S1 to use less chefs. We can use the same argument for Wi+1… WQ.

By the end our modified Opt = Algo. We reach a contradiction that Opt was necessarily more optimal.