CIS 675 Homework 3

Design and Analysis of Algorithms

Saket Kiran Thombre

SU ID: 899913802

NetID: sthombre

Email: sthombre@syr.edu

Collaborators for Q1, Q2:

1. Saurav Shashank Narvekar

   NetID: sanarvek
2. Chandan Pothumarthi

   NetID: cpothuma

Problem 1 Finding Opposite Pairs (Point 5)

A stockbroker wants to trade n × 100 shares of the stock AMZN in blocks of 100. Temporary market rules prevent him/her from short selling; in other words, he/she must own a block of shares before selling them. At the end of the day, he/she wants to own zero shares of AMZN. How many ways can he/she do his/her trading? For example, when n = 2, he/she can do 'BUY, SELL, BUY, SELL' or 'BUY, BUY, SELL, SELL' and so there are 2 ways. Present your answer as a recurrence equation. (3 points) Explain your solution and why your solution is correct. (2 points)

Note: You have to provide a DP solution. Other type of algorithmic solution is not acceptable.

Solution: In this solution, I have tried to explain all the operations a stockbroker can make for different values of N. We divide the set into 2-sub problems i.e., Buy and Sell.

Sell operation must be always followed with a Buy operation. Number of Buy and Sell operations should never be different i.e., Buy = Sell.

To derive this formula for total combinations of Buy and Sell Operations, we assume that all possible Buys and Sell Operation cycles must end at even indexes in the set. So, if the Buy and Sell Operation set is even, then the operation must contain $\frac{n-2}{2}$ more probable sets we can compute.

For example,

BUY	SELL	BUY	BUY	SELL	SELL	BUY	SELL
0	1	2	3	4	5	6	7	8

So, the probable sets can occur between the indexes 2,4,6,8 and the remaining terms should be then calculated by keeping into account that first term is always going to be BUY and last term always SELL.

We will now calculate all possible sequences and find recursive relation:

For all probable sets, B = Buy Operation and S = Sell Operation.

For N = 0, BS(0) = 1,

For N = 1, BS(1) = 1,

Probable sets: BS

For N= 2, BS (2) = [BS(0) * BS(1)] + [BS(1) * BS(0)] = 2,

Probable sets: BSBS, BBSS

For N = 3, BS (3) = [BS(0) * BS(2)] + [BS(1) * BS(1)] + [BS(2) * BS(0)] = 5

Probable sets: BBBSSS, BBSBSS, BSBBSS, BSBSBS, BBSSBS

For N = 4, BS (4) = [BS(0) * BS(3)] + [BS(1) * BS(2)] + [BS(2) * BS(1)] + [BS(3) * BS(0)] = 14

Probable sets: BBBBSSSS, BBSBSBSS, BBBSBSSS, BBBSSSBS, BBSSBBSS, BBSBSBSS, BSBBSSBS, BBSSBSBS, BSBSBSBS, BSBBBSSS, BBSSBBSS, BSBBSBSS, BBSBBSSS, BSBBSSBS

Similarly for N = n, BS(n) = [BS(0) * BS($\frac{n-2}{2}$)] + [BS(1) * BS($\frac{n-4}{2}$)] + [BS(2) * BS($\frac{n-6}{2}$)] + … + [BS($\frac{n-2}{2}$) * BS(0)]

We also must keep in mind that half of the operations are going to be BUY and the other half SELL. Hence, we get the term BS ($\frac{n-2}{2}$).

Solving it further we get the equation,

BS(n) = $\sum_{i=0}^{n-1} \sum_{j=n-1}^{0} BS(i) * BS(j)$

Since, this equation holds for all values of n, therefore we can say the solution holds true.

Pseudo-code for the above equation is

```
for i
    function(n)
    j = n-1
    i = 0
    total = 0
        while j >= 0 & i < n
        total += function(i) * function (j)
        i++
        j--
```

Problem 2: A fair division of the class (Point 5)

You are given an array that holds the weights of n people in the class W = (w1, w2, . . . , wn). Your goal is to divide the n people into two teams such that the total weight of the two teams is equal or as close as possible to equal. Describe such an algorithm and give its running time. The total number of people on each team should differ by at most 1. Assume that M is the maximum weight of a person, i.e., ∀iwi ≤ M. The running time should be a polynomial function of n and M. The output should be the list of people on each team and the difference in weight. Explain your algorithm (2 points), explain why your algorithm is correct (2 points), and explain the runtime of your algorithm (1 point). Note: You have to provide a DP solution. Other type of algorithmic solution is not acceptable. To get full points, your solution's runtime needs to be O(n 3M) or less (yes there is a more effective solution).

Solution:

In this question, we try to divide the people in 2 groups such that their difference is as minimum as possible.

There are only two options left after splitting the people into two groups. The individual now being chosen is either a member of group 1 or group 2. We do this again for each person and look for the best answer.

```
All the algorithm lines are numbered.

Checking if the number of weights is not going out of bound
```

```
1.   if weightArray = weightNumber,
2.      return

Selecting all the given weights to verify that the number of remaining elements
does not fall below the number of elements needed to create the solution.
3.   if (weightNumber/2-selectedWeight) > (weightNumber - weightArray)
4.      return

5.   equalWeight(givenWeight, weightNumber, currentList, selectedWeight,
solution, difference, itemSum, total, selectedPosition+1)

Adding the element from the list to the solution
6.   selectedWeight = selectedWeight + 1

7.   total = total + givenWeight[weightArray]

Taking items ast current position.
8.   currentList[weightArray] = true

Forming a solution to balanve the weights and considering it te best solution
9.   if selectedWeight = weightNumber/2, then

Checking If we find a better solution to balance the weights
10.   if difference of (itemSum/2 and total) < diff, then

11.      difference = difference of (itemSum/2 and total)

12.      while i = 0 to weightNumber, do
13.            solution[i] = currentList[i]
14.         done
15.   else

Else we consider the weight from the current list is to be included in the
solution
16.   equalWeight(givenWeight, weightNumber, currentList, selectedWeight,
solution, difference, itemSum, total, selectedPosition+1)

Removing current list element before returning to the function call
17.   currentList[weightArray] = false
```

After initializing the current list as empty, we start adding people one by one. We keep on adding till the current list become n/2. Now we check if the current list we have is the best possible solution or not. If yes, we use it else we try finding a better solution and update it.

The time complexity for the given solution is O(n^2)