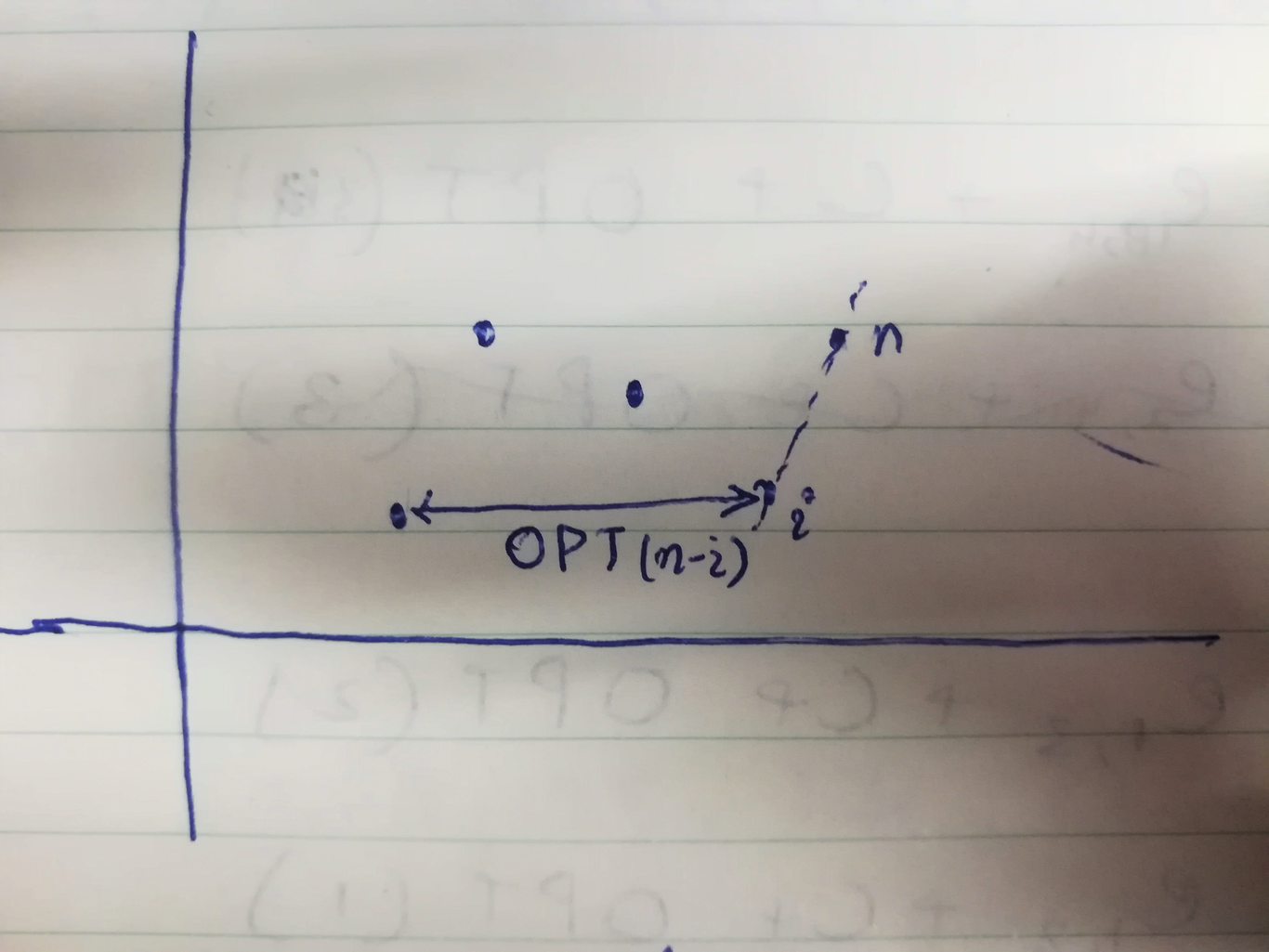
Title

Rahul Motan

# Assignment - 3

1 a) To prove the problem can be solved using Dynamic Programming, We need to first prove the problem has a optimal substructure. Let’s assume we are given points such that

There is no intuitive way to suggest an optimal solution but to check for all given points & find the best set of fitting lines. To do this we can assume we have an optimal solution , and in this lies the  point . Now, there must be a line segment beginning at some point ‘’ that connects the  point and is the best fit among all possible  lines drawn from

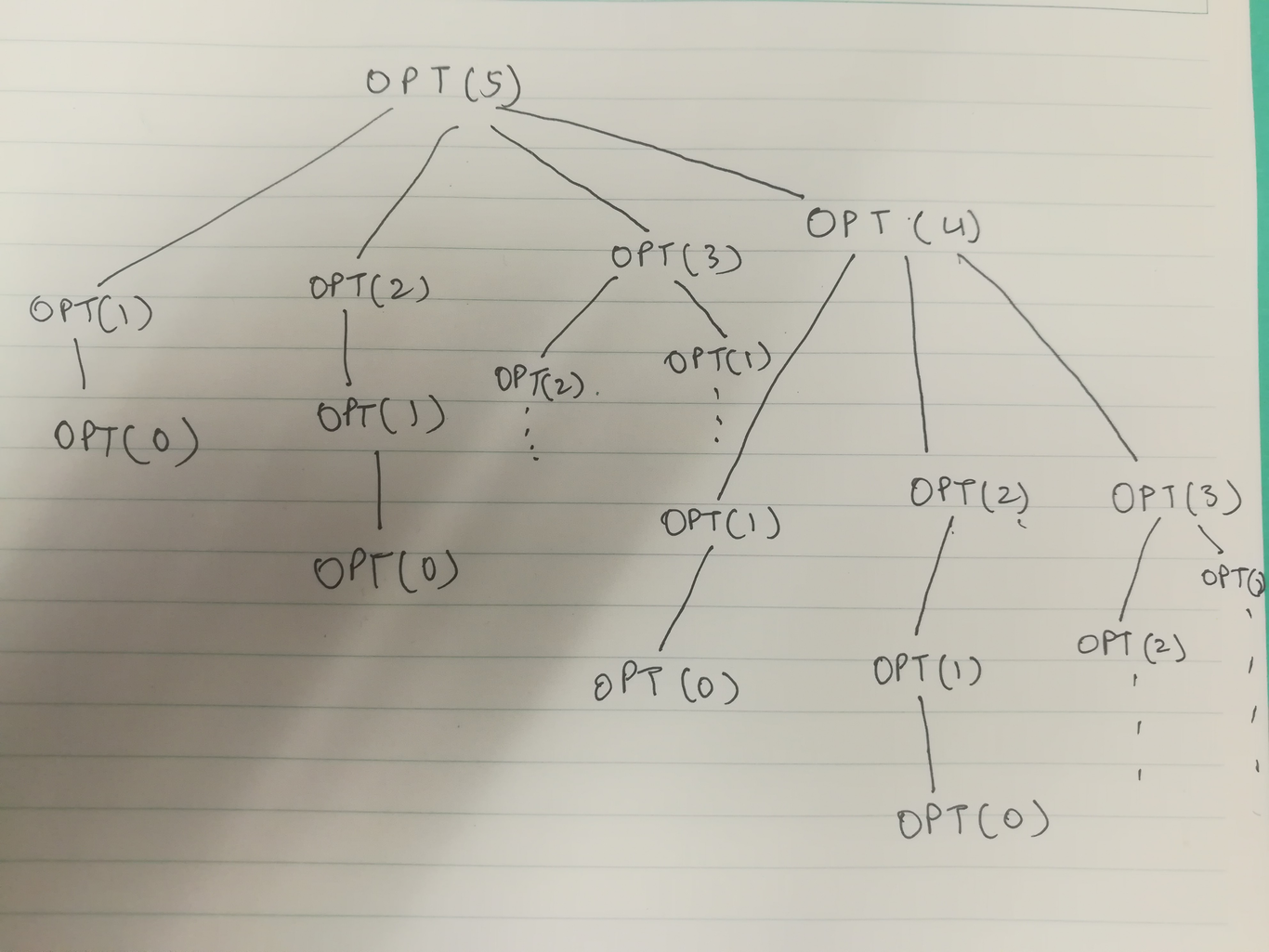


For n = 5

Thus, further evaluation of the above sub-structure gives the following recurrence for ,

here we can say,  as for no point there will be no penalty.

b) So for the above given recursion we can draw a recursion tree to visualize if our optimal substructure stands correct.



Recursion tree for n = 5

As we can see this type of recursion depicts optimal substructure as our sub-problems lead to the solution of the problem if we go bottom-up.

Since we have to find the best fitting line, we’ll have to minimize the penalty for all the points.

So the recurrence for the remaining  can be given as:-

c) Error computation involves computing error for  pairs and using the formula that is given it will take    time for calculating all the co-efficients for every pair so this takes the time complexity to

Next we iterate over all the pairs to calculate the penalty for each pair of points starting from 1 to n. So that takes time in the order of . So the total time complexity remains to be   .

**Space Complexity**: For memoization we’ll need an array of size  and to save all the error (for all i,j pairs) we’ll need  space

d**)      BEST-FIT(){**

{

                                 {

                                  }

                        }

                       Create memo table M of size n

                                 //  OPT(0) = 0

                       {

                             {

                             }

                      }

                    traverse backwards through A beginning from A[n] till 0 as the value at A[n] will be the  where the last optimal                     partition was created save the values in array say parts[]

                    return parts

        }

2) a) In this problem we are given a sequence of ‘n’ weights for each participant i.e total ‘m’ participants. Let’s suppose we have an           optimal solution for this problem, say .

          Now, consider the  weight of a participant, that weight can either belong to the longest weight loss sequence or not, if it does           we just add this to already found subsequence & we get an even longer subsequence otherwise we just look at the remaining n-1            weights and find a subsequence from it.  Given problem is quite similar to the problem of finding longest common subsequence           in two strings.

    b) Given a sequence of numbers, say  we can find the longest decreasing subsequence which will be a             subset of . It need not consist of consecutive members from the original array. Let that sequence be              where  and . We need to find all the possible subsets of the form given above and then check which is the longest of them all. To compute that we compare every  and increase the length of the subset if the conditions match. Individually every member of the array ’A’is in itself a decreasing subsequence.

    c)             LongestLoss(int[] ar){

                            // initialize memo table with value 1 as all of the elements are decreasing subsequence in itself

                                M[0…m][0…n] = 1

                                SSeq[0…m][0….n]

                                 {

                                     {

                                             {

                                                {

                                                    }

                                                }

                                            }

                                     }

                        max, (i,j)Now find the maximum in M and save the value and index [i][j] of the location

                        weeks[]from this SSeq[i][j] traverse backwards as all the week numbers are stored as linked to the index

                        return max, weeks