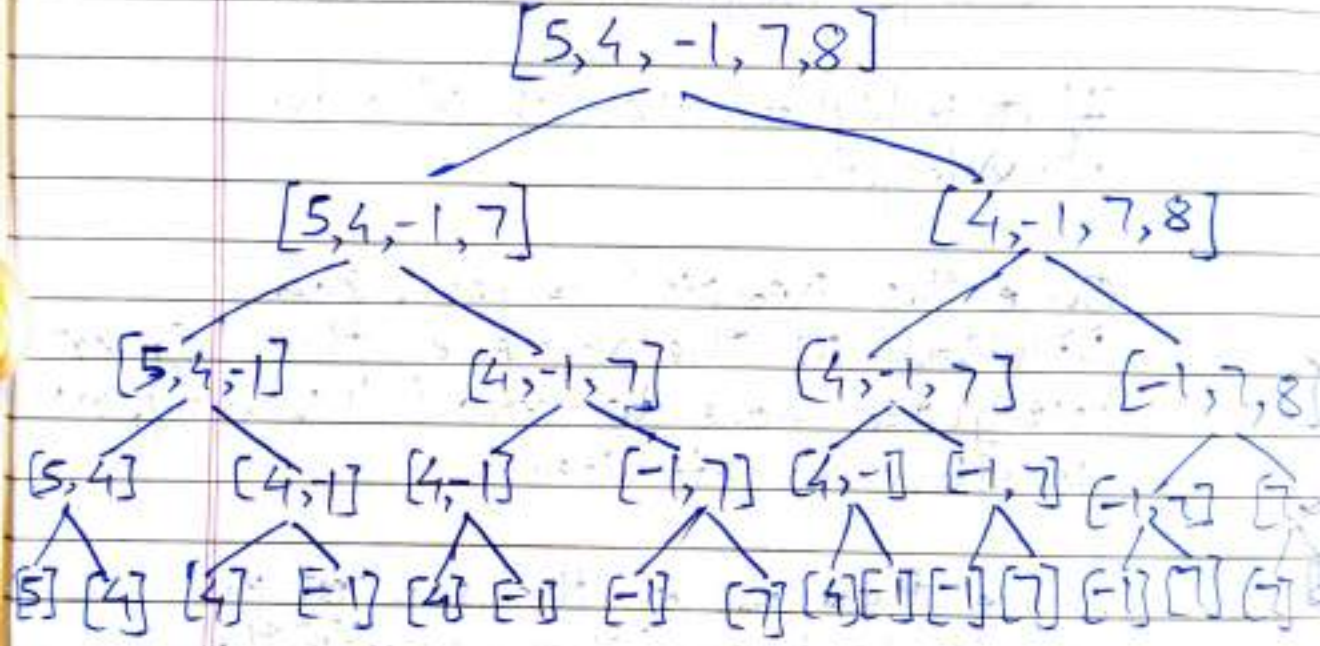
CSCI 570 Anash Mehta

Homework #7

Q1

Let's take the example of a=[5,4,-1,7,8]. Here, since we want to calculate the largest contiguous value, we will eliminate either the first or last element and then recursively solve the function.



Our base case would be an array that contains only one value, and would return the sum of that to the parent. The parent will compare its own sum with the one received from the children. The final answer will be found at the top node when the comparison is made.

To optimize our algorithm, we will store the key value as the array that we are processing and the value as the sum of that array. At the parent, we compare both values with its sum and check which is the largest and return it.

Algorithm:

```
computeLargestSum(arr, memo) {
  if arr.length = 1:
      return arr[0].
  for i=0 to arr.length
      sum += i
  if arr in memo:
      return memo[arr]
  for i=0 to arr.length
      sum += i
  end for

  temp1 = computeLargestSum(arr[0:n-1], memo)
  temp2 = computeLargestSum(arr[1:n], memo)

  memo[arr] = greatest of temp1, temp2
              and sum.
  return memo[arr]
end function
```

The algorithm will take $O(n^2)$ time.
- n for calculating sum n times.

Q2

Here, we maintain two pointers p1 and p2 and store the highest profit in a seperate 'maxProfit' variable which is initialized to 0.

We will use the prices in an array and iterate over those prices. Pointer p1 begins with the first value and pointer p2 with the second value.

~~If we observe that the profit is falling below 0, we shift the p1 and p2 pointers to right (increment).~~

If we observe that price at p1 is less than p2, we check for the profit obtained and store the higher value in maxProfit.

If it is not, we assign p2's value to p1.

In either case, we increment p2. n is the length of prices array.

Algorithm:

```
HighestProfit(prices):
    p1 = 0, p2 = 1, maxProfit = 0.
    while p2 < n:
        if (prices[p1] < prices[p2])
            profit = prices[p2] - prices[p1]
            maxProfit = max(profit,
                            maxProfit)
        else
            p1 = p2
        p2 += 1
```

end if
end while
return maxProfit

The algorithm runs through the array once, hence the time complexity is O(n).

Q3 :-

For finding the answer, we need to find the highest happiness score in the increasing subsequence. To find the increasing subsequence, we need to see if element i is greater than the previous element in the increasing subsequence.

Let OPT(i) = max value of happiness score in an increasing subsequence from [1...i]

For every element i, we check for every element j from 1 to i if we find a higher happiness score.

$$OPT(i) = \max\{OPT(j), x\,a[j] * j\}, OPT(i))$$

PAGE No.
DATE / /

Algorithm:

```
FindMaxHappinessScore(arr)
  Initialize OPT(0) = arr[0]
  For i=n to 0:
      For j=1 to i:
          If arr[j] < arr[i];
              OPT(i) = Max(OPT(i), OPT(j)
                                  +(arr[j]-j
          End If
      End For
  End For
End Function
```

The algorithm runs for $O(n^2)$ time complexity.

Q4. Here, we assume a given binary matrix A[m][n] which contains all 0's and 1's. We construct another matrix, S[m][n] of the same size and initialize the first row and column values from A matrix.

Our recurrence formula would be:

$$S[i][j] = Min(S[i-1][j], S[i-1][j-1], S[i][j-1]) + 1. \quad \hookrightarrow (1)$$

This is if A[i][j] is 1.
If A[i][j] is 0, we just copy the values.

Once we calculate all the values for matrix S, we find out the maximum entry and return it as the longest length of a subsquare.

Algorithm:

```
FindHighestLengthOFSubsquare(A, m, n)
    Initialize a matrix S
    Copy the first row & column from A
        to initialize.
    For i=1 to m:
        For j=1 to n:
            Use Rec. Formula (1) if A[i][j]=1
            Else S[i][j] = 0
        End For
    End For
```

Find Highest value in the matrix
Return max value.
End Function.

The Algorithm runs in a time complexity of O(m x n)

i.e. no. of rows x no. of columns.