HOMEWORK-3

Jagan Mohan Reddy Bijjam (JXM210003)
Sai Mallikarjuna Rao Bollam (SXB200136)

1.) • Given a_i, a_j, a_k, they form a convex set if $\frac{a_i + a_k}{2} > a_j$, where $i < j < k$.

• If A[1], A[2], A[3] form a convex set and we want to include A[4], then A[2], A[3], A[4] should form a convex set too, else we can skip to A[5] and test A[2], A[3], A[5]

• ∴ recurrence $LCS(i,j,k) = Max(1 + LCS(j,k,k+1), LCS(i,j,k+1))$.

• ~~If the above recurrence is true only if (i,j,k)~~

• $1 + LCS(j,k,k+1)$ is possible only if (i,j,k) form a convex set.

• Since the recurrence is only on k, we loop through all initializations of i & j.

• Since the above recurrence gives numbers of such convex sets, the true LCS would be 2 more than the number of convex sets.

Runtime :-

.) As we have three for loops running from 1 to n and the code inside the innermost for loop takes unit time, the run time of the algorithm is $O(n^3)$.

Pseudo Code:-

```
Procedure (A[1, ---, n]) {
    output ← 0
    if (n < 3) return 0;
    dp[n, n, n+1]  (Initialize to zeroes)
    for (i = n-2; i ≥ 1; i--){
        for (j = n-1; j ≥ i; j--) {
            for (k = n; k ≥ j; k--){
                t1 = 0;
                if (A[i] + A[k] > 2·A[j])
                    t1 = 1 + dp[j, k, k+1];
                t2 = dp[i, j, k+1];
                dp[i, j, k] = max(t1, t2)
                output = max(output, dp[i, j, k])
            }
        }
    }
    if (output == 0) return 0;
    else return output+2; // Total numbers in the longest convex subsequence.
}
```

→ Since LCS(i,j,k) depends on (j,k,k+1) & (i,j,k+1) we fill the table from highest indices of (i,j,k) and progress towards 1.

2.) → To obtain maximum profit by cutting a marble of size n×m, we have 3 options :-

1. Not to cut it at all → in which case the price will be P[n][m].
2. To cut it horizontally → in which case the price would be the maximum of the prices obtained at each horizontal cut.
3. To cut it vertically → in which case the price would be the maximum of the prices obtained at each vertical cut.

→ Hence the recurrence relation is

$$dp[n][m] = \max\left\{ P(n,m),\ \max_{1 \le i \le \lfloor n/2 \rfloor}\{dp[i,m] + dp(n-i,m)\},\ \max_{1 \le j \le \lfloor m/2 \rfloor}\{dp[n,j] + dp[n,m-j]\} \right\}$$

Runtime :-

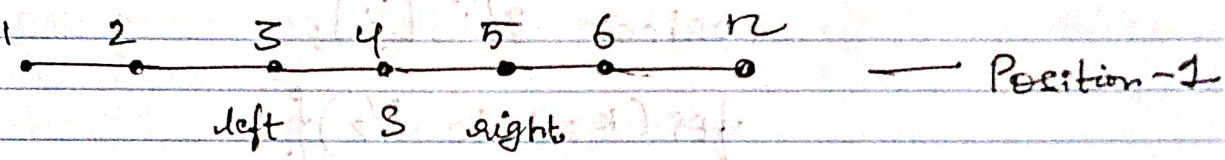
- As there are 2 outer for loops looping from 1 to n, and 1 to m respectively, Also inside the innermost for loop, there are 2 for loops - 1 looping from 1 to n/2 and the other looping from 1 to m/2, the run time would be $O\left(mn\left(\frac{m+n}{2}\right)\right) \simeq O(mn(m+n))$. If we assume n & m to be equal then the runtime would be $O(n^3)$.

Pseudo Code :-

```
Procedure (P[1,---,n][1,---,m]){
  dp[1,---n][1,---m] (Initialized to zeros)
  dp[i][j] = P[i][i];
  for (i=1 to n){
    for (j=1 to m){
      price = P[i][j];
      for (k=1 to i/2){
        price = max(price, dp[k][j]+dp[i-k][j])
      }
      for (k=1 to j/2){
        price=max(price, dp[i][k]+dp[i][j-k])
      }
      dp[i][j] = price;
    }
  }
  return dp[n][m];
}
```

3.) → In order to calculate minimum number of ounces of soda consumed by all students, we need to calculate, minimum distance travelled by all students while on the bus.

no: of ounces = $\frac{\text{distance}}{\text{speed}} \times$ rate of consumption

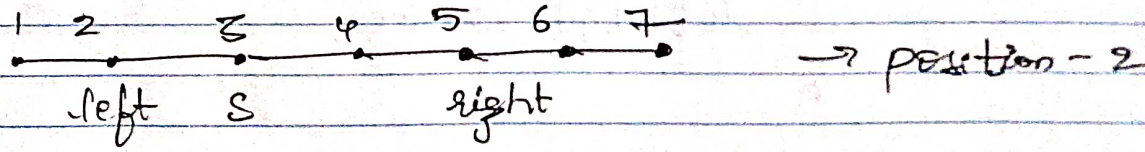


→ Let s be the index of the exit where the bus will start.

→ Let the bus go towards left by one exit index, the cumulative distance travelled by the students in the bus would be

travelcost = $[R(1, left) + R(right, n)] * (E(s) - E(left))$

→ The new position would be :-



→ dist(position 1) = travel cost + dist(position 2)

⇒ If the bus goes left

min_dist(left, right, s) = travel_cost + min_dist(left-1, right, left).

→ Since the bus can also go right, the final recurrence relation would be

***) min_dist(left, right, s) =

$$\min\{([P(1,left)+P(right,n)]*|E(left)-E(s)|+ min\_dist(left-1, right, left)),$$
$$([P(1,left)+P(right,n)]*|E(right)-E(s)|+ min\_dist(left, right+1, right))\}$$

→ If bus it is at s=0 or s=n, we can just iteratively calculate the distance.

→ One optimization we can add is by observing that bus is always at one mile marker ahead of left or one mile marker behind of right.

→ So, instead of passing position of bus, we just pass a boolean indicating either of the above discussed scenario.

→ TRUE indicates one mile marker ahead of left and FALSE indicates one mile marker behind of right.

∴ The pseudo code with the updated recurrence would be:-

Pseudo code :-

```
Procedure (E[1,---,n], R[1,--n][1,---n/2], start){
    left = start - 1;
    right = start + 1;
    indicator = 0; // 0 → start = left+1,
    memo[n][n][2] // Initialized to null
    dist = min-dist (left, right, indicator){
        If (memo[left][right][indicator] != null){
            return memo[left][right][indicator];
        }
        If (indicator == 0){
            s = left+1;
        }
        else{
            s = right -1;
        }
        If (left == 0 && right == n) return 0;
        If (left == 0){
            return R(right, n)*|E(s)-E(right)|+
                                min-dist(left, right+1, 1);
        }
```

```
if(right == n+1){
    return R(1,left) * |E(s) - E(left)| +
                min_dist(left-1, right, 0)
}

return min([ A ], [ B ]); where

A = [(R(1,left) + R(right,n)) * |E(s) - E(left)| +
                min_dist(left-1, right, 0))]

B = [(R(1,left) + R(right,n)) * |E(s) - E(right)| +
                min_dist(left, right+1, 1))]

} // end of 'min_dist'

min_ounces_of_soda = dist/speed , x rate of consumption

return min_ounces_of_soda;

}
```

Runtime :-

→ Since we are using memoization and each recursion takes O(1) time before passing it to other recursions, time complexity would be the dimension complexity of the memo table which is $n \times n \times 2 \cong O(n^2)$