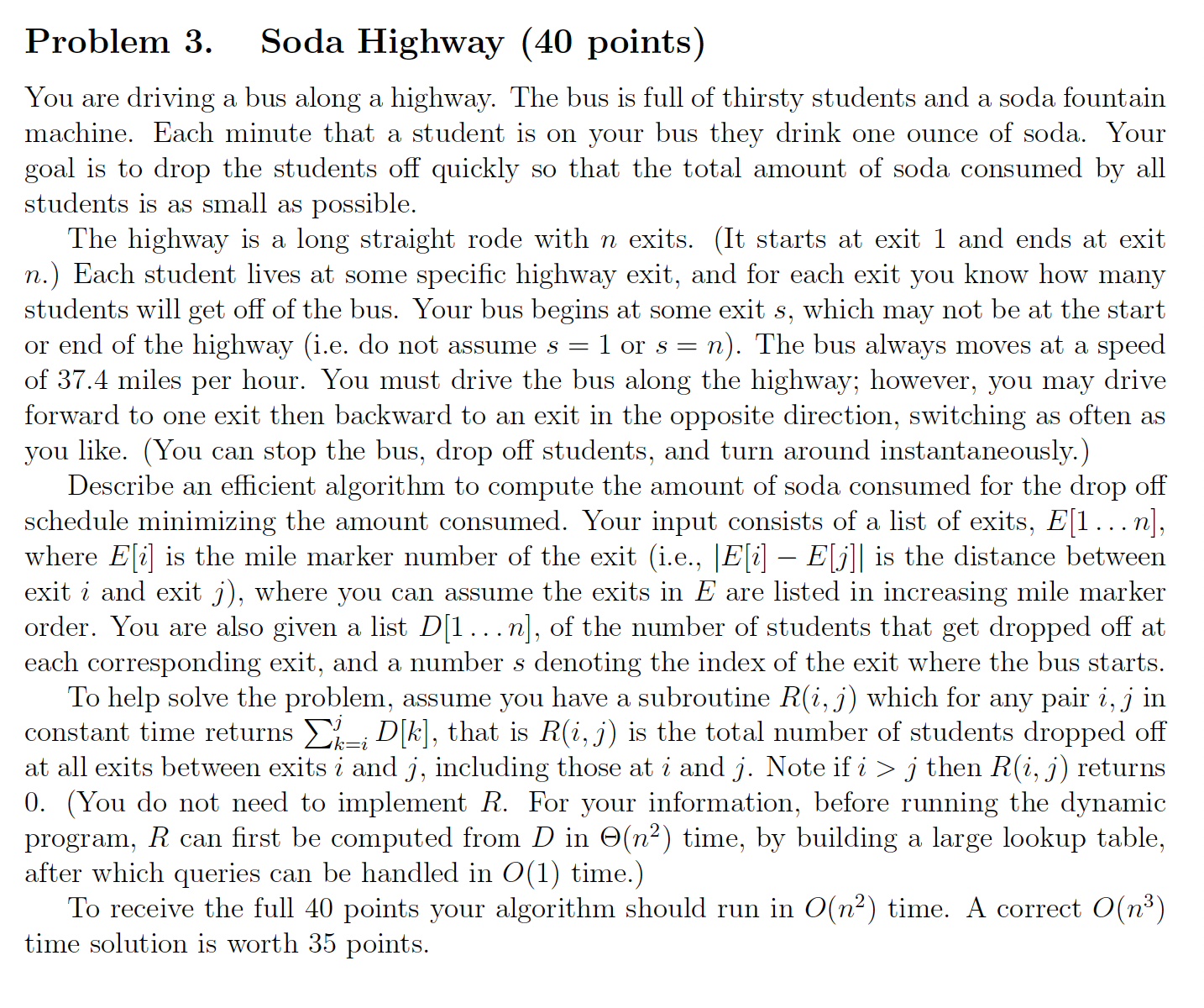
**Homework 3**

**Indukuri Vijay Anand Varma (VXI210000)**

**Karthik Raghunath Ananda Kumar (KXA200005)**



**Intuition:**

Intuitively, we can say that number of ounces can be minimized by minimizing the total distance travelled by all the students while on bus. Minimizing (Students on bus \* Distance travelled by the bus).

Similar to LIS, we can consider two options for all the exits. One is going left from that exit and another is going right. Initially, we take given starting point as our current exit, exit before starting point as left pointer and exit after starting point as our right pointer. If we go left, right pointer will remain unchanged and both left and current pointers will reduce by one exit. Similarly, going right will not change left pointer but both right and current pointer will increase by one exit.

Cost incurred after taking a left step, cost[left] = (E(current) – E(left)) \* (R (1, left) + R (right, N))

(E(current) – E(left)) gives the distance in miles between current exit and left exit.

(R (1, left) + R (right, N)) gives the total number of students travelling from current exit to left exit

Cost after taking a right step, cost[right] = (E(right) – E(current)) \* (R (1, left) + R (right, N))

(E(right) – E(current)) gives the distance in miles between current exit and right exit.

(R (1, left) + R (right, N)) gives total number of students travelling from current exit to right exit

Recursively left step incurs a total cost of: cost[left] + Solution (left-1, left, right)

Recursively right step incurs a total cost of: cost[right] + Solution (left, right, right+1)

Total cost at an exit will be Minimum of (cost[left] + Solution (left-1, left, right)) and (cost[right] + Solution (left, right, right+1))

Base case will be when left exit becomes zero or right exit becomes N+1. Also, when both left becomes zero and right becomes N+1, we return zero.

**Pseudo Code:**

def Solution (E [], D [], R [] [], left, current, right):

if (left = 0 && right = N+1) return 0;

if (left = 0) {

return (Solution (E [], D [], R [] [], left, current, right + 1)) + (R (right, N) \* E[right] – E[current]);

}

if (right = N+1) {

return (Solution (E [], D [], R [] [], left-1, current, right)) + (R (1, left) \* E[current] – E[left]);

}

costLeft = (E(current) – E(left)) \* (R (1, left) + R (right, N))

costRight = (E(right) – E(current)) \* (R (1, left) + R (right, N))

updatedLeftCost = costLeft + Solution (E [], D [], R [] [], left-1, left, right)

updatedRightCost = costRight + Solution (E [], D [], R [] [], left, right, right+1)

return min (updatedLeftCost, updatedRightCost)

Above pseudo code runs in O(2n) time because of the recursion. Now we try to induce memorization into above recursive solution to get better time complexity.

**Pseudo Code with Memoization:**

In this we try to use the previous move of bus (LEFT or RIGHT). Using this move we get the current exit of bus. Let’s call this variable as prev\_move and its values will be LEFT or RIGHT. Let **cache** be a 3D array where first dimension will be left exits, second will be right exits and third will give us information about our previous move. If our previous move is LEFT, then current exit will be left + 1. If our previous move is RIGHT, then current exit will be right – 1.

def Solution (E [], D [], R [] [], cache [] [] [], left, right, prev\_move):

if (cache [left] [right] [prev\_move] ! = null) {

return cache [left] [right] [prev\_move];

}

if (left = 0 && right = N+1) {

return 0;

}

if (prev\_move = “LEFT”) {

current = left + 1

} else if (prev\_move = “RIGHT”){

current = right - 1

}

if (left = 0) {

return (Solution (E [], D [], R [] [], left, right + 1, prev\_move)) + (R (right, N) \* E[right] – E[current]);

}

if (right = N+1) {

return (Solution (E [], D [], R [] [], left-1, right, prev\_move)) + (R (1, left) \* E[current] – E[left]);

}

costLeft = (E(current) – E(left)) \* (R (1, left) + R (right, N))

costRight = (E(right) – E(current)) \* (R (1, left) + R (right, N))

updatedLeftCost = costLeft + Solution (E [], D [], R [] [], left-1, right, “LEFT”)

updatedRightCost = costRight + Solution (E [], D [], R [] [], left, right+1, “RIGHT”)

cache [left] [right] [prev\_move] = minimum (updatedLeftCost, updatedRightCost)

return minimum (updatedLeftCost, updatedRightCost)

**Time Complexity:**

This pseudo code with memoization runs in O(n2) because we are creating a table size of n\*n\*2 = 2n2 and each entry will be created in constant time.