# CS 374 HW 5 Problem 2

Aldo Sanjoto, quddus2, Hieu Huynh

TOTAL POINTS

## 80 / 100

QUESTION 1

### 1 Wireless routers. 80 / 100

- **0 pts** Correct
- **75 pts** IDK
- **20 pts** Suboptimal Solution in O(n^3k)
- **100 pts** Exponential Time Algorithm
- **100 pts** Incorrect Algorithm
- **10 pts** Incorrect base case
- **50 pts** Incorrect recursive case
- **10 pts** Incorrect/missing runtime
- **10 pts** No specification of how to call function to get final answer
- **20 pts** No English description of function being calculated
- **50 pts** Suboptimal Polynomial Time Algorithm
- **20** **Point adjustment**

  Runtime is suboptimal (see solution for O(n^2 k)).

(Q2)

- Define: $my\_cost(s, Y) = \sum_{i=s}^{n} cost(l_i, Y)$ : cost of all customers with $l \geq l_s$ $(1 \leq s \leq n)$

  $\Rightarrow my\_cost(s, \emptyset) = \infty$

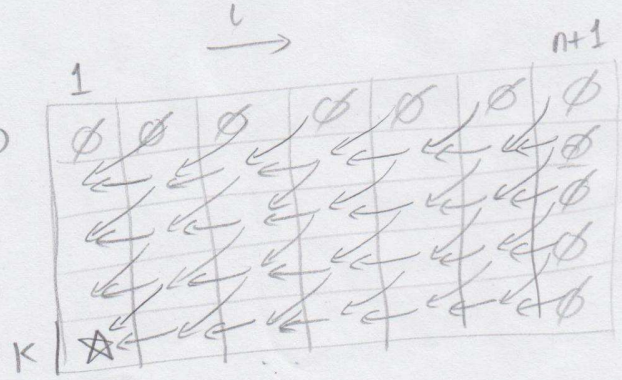
- Let $Opt(i, j)$ denote the optimal solution for placing $j$ routers in $(n-i+1)$ locations $l_i, \ldots, l_n$. We have the recurrence:

$$Opt(i,j) = \begin{cases} \emptyset & \text{if } i > n \text{ or } k \leq 0 \\ Y_1, & \text{if } my\_cost(i, Y_1) < my\_cost(i, Y_2) \\ Y_2, & \text{if } my\_cost(i, Y_1) \geq my\_cost(i, Y_2) \end{cases}$$

$Y_1$ and $Y_2$ are defined as following:

- $Y_1 = Opt(i+1, j)$
- $Y_2 = Opt(i+1, j-1) \cup \{l_i\}$

- We need to compute $Opt(1, k)$.
- We can memoize the function $Opt(i, j)$ into an array $A[1 \ldots n+1, 0 \ldots k]$. Each entry $A[i, j]$ is optimal solution for placing $j$ routers in $(n-i+1)$ locations $l_i, \ldots, l_n$.
- We have entry $A[i, j]$ depends on entry $A[i+1, j]$ and $A[i+1, j-1]$, we fill the array from top to bottom in a column and move to the column on the left when finish



```
Optimal_Y (L[l1...ln], K) {
    A[1..n+1, 0...k].
    For j ← 0 to K:                          // Base case
        A[n+1, j] = ∅
    For i ← 1 to n+1:                        // Base case
        A[i, 0] = ∅

    For i ← n to 1 {
        For j ← 1 to K {
            cost1 = my-cost(i, A[i+1, j]);
            temp = A[i+1; j-1];
            temp.add(li)
            cost2 = my-cost(i, temp)
            if (cost1 < cost2)
                A[i, j] = A[i+1, j];
            else
                A[i, j] = temp;
        }
    }
    return A[1, K];
}
```

The running time of this block equals 2 times the running time of function my-cost()

• Implement function my_cost (s, Y).

```
my_cost (s, Y) {
    If (Y == ∅) {
        return ∞;
    }
    total ← 0.
    For i ← s to n {          // n is the total of locations
        total = total + (li − nn(li, Y))^4.   // this take O(k) time
    }
    return total;
}
```

O(n·k) | O(k)

This function takes O(n·k) because each the block inside for loop take O(k). And we loop through it n times.

Analysis of the algorithm:

To fill each entry we need O(n·k) time because we call function my_cost() two times.

We have O(n·k) entries in the table

⟹ Total running time = $O(n^2k^2)$

## 1 Wireless routers. 80 / 100

- **0 pts** Correct
- **75 pts** IDK
- **20 pts** Suboptimal Solution in O(n^3k)
- **100 pts** Exponential Time Algorithm
- **100 pts** Incorrect Algorithm
- **10 pts** Incorrect base case
- **50 pts** Incorrect recursive case
- **10 pts** Incorrect/missing runtime
- **10 pts** No specification of how to call function to get final answer
- **20 pts** No English description of function being calculated
- **50 pts** Suboptimal Polynomial Time Algorithm

**- 20 Point adjustment**

Runtime is suboptimal (see solution for O(n^2 k)).