Table of Contents

Question 1.

The subproblems we will be looking at are the best laundry services to get for loads up to week $i-1$ or $i-3$ where the best services are decided by minimizing cost using $\min(\texttt{price\_table}[i-1])$.
For a set of loads $l_0, \cdots, l_i$, we can find can either choose L_1 or L_2 for l_i. If we:

- Choose L_1 then we add $r \times l_i$ and the optimal cost for $l_0, \cdots, l_{i-1}$
- Choose L_2 then we add $3 \times w$ and the optimal cost for $l_0, \cdots, l_{i-3}$

The recurrence relation for the optimal cost for l_i is

$$\text{Opt}(i) = \begin{cases} 0 & \text{if } i = 0 \\ l_i \times r + \text{Opt}(i-1) & \text{if } i = 1 \text{ or } 2 \\ \min \begin{pmatrix} l_i \times r + \text{Opt}(i-1), \\ 3w + \text{Opt}(i-3) \end{pmatrix} & \text{if } i \geq 3 \end{cases}$$

To find the optimal schedule, we can trace backwards. If we chose L_1, go back one load. If we chose L_2, go back 3 loads.

Algorithm
Choose_Laundromats (`loads`: [array of load size per week], `r`: L_1 `rate`, `w`: L_2 `rate`)

```
1: price_table = [0*n]
2: schedule = []
3:
4: for i = 1 to n do
5:     if i == 1 or i == 2 then
6:         price_table[i] = min(l_i r + price_table[i-1], 3w)
7:     else
8:         price_table[i] = min(l_i r + price_table[i-1], 3w + price_table[i-3])
9:
10: bt_index = n
11: while bt_index ≥ 1 do
12:     if price_table[bt_index-1] + l_{bt_index-1} r = price_table[bt_index] then
13:         schedule.prepend(L_1)
14:         bt_index −= 1
15:     else
16:         schedule.prepend(L_2)
17:         bt_index −= 3
18:
19: return schedule
```

Proof of Correctness (by Induction):

Base Case:

If $i = 0$, return 0, since we can't choose any laundromats

If $i = 1$, return rl_1 since we can't choose L_2

If $i = 2$, return $rl_1 + rl_2$, since we can't choose L_2

Induction Hypothesis:

Assume Opt(j) returns the optimal load costs for $0 \leq j < i$

Induction Step:

Consider l_i:

If l_i should use L_1, then the new cost will be $rl_i + \text{Opt}(i-1)$

If l_i should use L_2, then the new cost will be $3w + \text{Opt}(i-3)$

Since we pick the minimum of those two values, and they are the only two options, the recurrence relation uses the minimum cost of the values by the induction hypothesis.

For the time complexity we have $\mathcal{O}(n)$. This is because we have two for loops at lines 4 and 11 which loop from 1 to n and from n to 1. In each of these loops we only do $\mathcal{O}(1)$ operations such as min, addition, multiplication, and array look up and array assignment, so in total we use $n\mathcal{O}(1) \in \mathcal{O}(n)$ time.

For space complexity we have $\mathcal{O}(n)$. This is because we have the input array which is $\mathcal{O}(n)$ and we initialize an array with n zeros, and we create an empty array for our return result, which can at most have n elements, as each load cannot use more than one laundry service. All operations are done on these arrays or using integer variables, so in total we use $3n \in \mathcal{O}(n)$ space.

Question 2.

The subproblem we will be looking at is the longest increasing trend that ends at index i.
We can simply take the max value of longest increasing trend that ends at index i and return it.

To compute the solution using the subproblems, we have to is to look at all previous indexes. We want the current price $S[i]$ to be bigger than the longest increasing trend's price $S[j]$ for some j before i. This means that we can add $S[i]$ to the end of the longest increasing trend, so we have the length of that $+1$

Suppose the sequence S starts at index 1. The recurrence relation is

$$\text{Opt}(i) = \begin{cases} 0 & \text{if } i = 0 \\ \max\left(\max\limits_{\substack{1 \le j < i \\ S[i] > S[j]}} \big(\text{Opt}[j]\big) + 1,\ 1 \right) & \text{otherwise} \end{cases}$$

Algorithm
Find_Longest_Increasing_Trend (`prices`: [array of price quotes])

```
1: lit_at_index = [1 * prices.length]
2:
3: for i in prices do
4:     for j in prices[:i] do
5:         if prices[j] < prices[i] and lit_at_index[i] ≤ lit_at_index[j] then
6:             lit_at_index[i] = lit_at_index[j] + 1
7:
8: return max(lit_at_index)
```

The time complexity of `Find_Longest_Increasing_Trend()` is $\mathcal{O}(n^2)$, and the space complexity is $\mathcal{O}(n)$.
We have a for loop that iterates over `prices` which is of size n, and the inner for loop runs from 1 to n. Every other line is $\mathcal{O}(1)$, so we have a final time complexity of $\mathcal{O}(n^2)$.

For space complexity, we just keep a memoization array of size n, so the final space complexity is $\mathcal{O}(n)$.

Proof of Correctness (by Induction):

Base Case:

If $i = 0$, return 0, since there are no numbers for the increasing subsequence.

Induction Hypothesis:

Assume Opt(j) returns the optimal solution for $0 \leq j < i$

Induction Step:

Consider Opt(i)

We go over all the elements before i and select the largest possible sequence with an element smaller than use $+1$. Since all the largest possible sequence for each element is optimal by our induction hypothesis, retrieving the largest one $+1$ will give us the maximal strictly increasing sequence.