Dennis Kuzminer
CSCI-UA 310-001 PS5

3.

a. *Opt*(i) :

```
if i = 0 then
    result ← 0
else
    result ← ∞
    for k in [0 .. i) do
        penaltyForLastDay ← (200 – (a_i – a_k))^2
        penaltyForPreviousDays ← Opt(k)
        result ← min(result, penaltyForLastDay + penaltyForPreviousDays)
return result
```

b. *Opt*(i) :

```
if T[i] = ⊥ then
    if i = 0 then
        T[i] ← 0
    else
        T[i] ← ∞
        for k in [0 .. i) do
            penaltyForLastDay ← (200 – (a_i – a_k))^2
            penaltyForPreviousDays ← Opt(k)
            T[i] ← min(result, penaltyForLastDay +
                penaltyForPreviousDays)
return T[i]
```

This algorithm will execute operations up to n for each n, making run time without memoization n(n+1)/2 or just O(n^2).

c. *Opt*(n) :

```
for i in [0 .. n)
    T[i] ← 0        // default value
for i in [0 .. n)
    result ← ∞
    for k in [0..i) do
        penaltyForLastDay ← (200 – (a_i – a_k))^2
        penaltyForPreviousDays ← T[i]
        if penaltyForLastDay + penaltyForPreviousDays < result
            result = penaltyForLastDay + penaltyForPreviousDays
    T[i] = result
return T[i]
```

For the same reason as in (b), we can see that this is also O(n^2)

d. *Sol*(i):

```
if i = 0 then
    return emptyList()
```

```
else
    for k in [0 .. i) do
        if T[i - k] + (200 – (a_i – a_k))^2 = T[i] then
            return concat(Sol(i - k), k)
```

This algorithm can be implemented in linear time (O(n)), if the concat operation is also linear. This can be achieved with a linked list data structure storing out answer.