1.
We can deduce a recurrence relation where P(i) is the max profit we can make at every i:

$$P(i) = max\{p\_i + P(previous(i)), P(i - 1)\}$$

So if i = 0 to n then our max profit is P(n) where n=total # of locations:

**Pseudo-code:**

```
for i in range (2, n):
 if (m_i− m_i−1 > k):
  previous(i) = i−1
 else:
  previous(i) = previous(i−1)
for i in range (2, n):
 P(i) = max(p_i + P(previous(i)), P(i – 1)
print P(n)
```

where the complexity is O(n) (we loop twice for n-1 iterations) trivially due to the single run through.

2.

```
def subseq(seq):
        lens = [] #to store our longest subseqs
        for i in range(1,n):
         lens[i,i] = 1 #each character is its own palindrome (size 1)
        for i in range(1,n):
         for j in range (1, n−i):
           k = j+i
           lens[j,k] = cost(lens,seq,j,k)
           lens[k,j] = cost(lens,seq,k,j)
        return lens[1,n] #n is the last index in seq
def cost(lens, seq, x, y):
        if x = y:
          return lens[x, y]
        else if seq[x] = seq[y]:
          if x+1 >= y−1:
            return 2
          else:
            return 2 + lens[x+1, y−1]
        else:
          return maximum(lens[x+1,j], lens[x, y−1])
```

Complexity: Since we loop n-1 times in every n loops (in subseq), then our complexity is: O(n*(n-1))=O(n^2-n)=O(n^2)