Matthew Soicher
z5160737

Question 2

Our aim is to find a path so that we minimise the number of 'lower to higher elevation' moves along the map. We do not care about the actual difference itself; just minimising the amount of times this occurs. We are restricted to making moves only to the immediate square below us or to the right

Note:

- map refers to the given $2D$ map
- $cost[(i,j),(k,l)]$ refers to the cost of moving from (i,j) to (k,l) in map. If a move involves an elevation increase, the cost is 1, otherwise 0.

We note that the cost of visiting any square (c,r) is given by the cost of visiting the square one position above $(c,r+1)$ or one square to the left $(c-1,r)$ plus the cost of that move itself. See the table below for clarification.

$(1,R)$	$(2,R)$	…	$(C-1,R)$	(C,R)
$(1,R-1)$				
…				
$(1,2)$				
$(1,1)$	$(2,1)$	…	$(C-1,1)$	$(C,1)$

Our solution is to fill an array $costTable$ of size $R \times C$, whereby the amount (total cost to that position) in position (i,j) is given by the minimum amount of either the position above $(i,j+1)$ or to the left $(i-1,j)$, plus the cost of that move.

If the cost of the two options are the same, we will always choose the position which has the highest elevation. This is to decrease the chance of an increase in elevation by that move (to reduce total cost).

For example, if we are at position (i,j) which has an elevation of 15, and the minimum path is from the position above, which has a total cost of 4 and elevation of 12, the position (i,j) will contain 5 (the minimum cost of the two options, plus 1 due to an elevation increase).

The base case is our starting position $(1,R)$ which will contain the value 0.

The Algorithm:
Given (i,j) in $costTable$, let

- min = $minimum[(i,j+1),(i-1,j)]$ and
- coordinates of min be (m,m). And so,

$$costTable(i,j) = min + cost[(m,m),(i,j)].$$

We continuously fill the table until we reach position $(C,1)$, at which point the amount in $(C,1)$ will be our answer; the least 'lower to elevation' moves possible.

Matthew Soicher
z5160737

Since we are required to find the path itself, for every recursive step, whereby we find that minimum position, we store the coordinates in an array. The actual path is that array in reverse.

The time complexity of our algorithm is $\boldsymbol{O}(\boldsymbol{R} \times \boldsymbol{C})$