Question 1.1

```
We can assume sam has two options, walk the shortest route of k steps, or walk to longest route of l-k steps,
giving us a time complexity of O(l-k). If we expand this to our worst case of walking the longest route when
k = 1, we get a worst case complexity of O(l-1) or more simply O(l)
```

Question 1.2

```
In this scenario our code works as follows

Walk k steps in either direction:
    if gate found:
        stop
    else:
        walk 2k steps back in the other direction

Our best case scenario is always k steps, and our worst case is always 3k steps with the average being 2k steps.

so the time complexity is O(3k) which is equivalent to O(k)
```

Question 1.3

```
PartCPsuedocode(startpoint, endpoint)
    n = 1
    totalsteps = 0

    currentDirection = Left or Right
    oppositeDirection = Right or Left

    while endpoint not found
        move n steps in currentDirection from startpoint
        totalsteps = totalsteps + n

        if endpoint found
            return totalsteps

        move n steps in OppositeDirection
        totalsteps = totalsteps + n
        n = n + 1
        swap initalDirection and oppositeDirection

Due to the nature of this algorithm our worst case would be when k is exactly half of l, which gives the
time complexity being (sum of f(k) from 1 to k-1 )+ k, with f(k) = 2k.

this gives us a time complexity of O(k^2) for even values of l


For odd values of k we assume we walk the the shorter side from the middle on the last iteration which gives
us the (sum of f(k) from 1 to k+1) - 1 , with f(k) = 2k.

this gives us a time complexity of O(((k+1)/2)^2 - 1) for odd values of l


Part A has a time complexity of O(l), and due to the fact for the worst case for part C we assume k is equal
to l/2 we can directly compare O(l) and O((l/2)^2). Due to the quadratic nature of O((l/2)^2) at large l it
will be much larger the O(l) hence Part A would be a better strategy for this problem

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
```

Question 1.4

We want to initally set distance walked in a direction to 1, then choose a direction either left or right and move that distance in that direction, we then move twice that distance in the other direction so we end up discovering equal distance to either side of the origin. We then return to the start quadruple the distance and repeat.

If at any point during the algorithm we find the gate we stop.

So in each n iteration we walk 4*distance with the distance being equal to 2^n, hence the dsitance travelled each iteration being 4*2^n = 2^(n+1)

So for each iteration we

Iteration	Travelled this iteration	Distance covered in each direction	Total
1	4	1	4
2	8	2	12
3	16	4	28
4	32	8	60
5	64	16	124
6	128	32	252
7	256	64	508
8	512	128	1020
9	1024	256	2052

We can see a pattern forming where the total distance is equal to 2^(n+2) - 4

We can then assume that the worst case is when the last full iteration covers (k+1) / 2 distance
i.e. k = 3, 7, 15, 31
As we then have to complete another almost complete cycle - 1 to find the gate

Hence worst case time is equal to or less then 8(k + 1) - 4
as 8(k+1) gives us the next equivilent full total of 2^(n+2) if k is 1 less then the next distance covered in each direction

so worst case time complexity is O(8(k + 1) - 4) which is linear k and equivalent to O(k)

Question 2.2

My code for question 2.1 works by using Prims algorithm for finding the minimum cost spanning tree to calculate the cable installation cost and comparing it against the antenna installation cost, which is given by numHouses * antennaCost.

My primMST algorithm works by finding the lowest cost to each node and 'locking' the node when we know that no cheaper cost can be found and 'locking' an edge when we know it must be in the MST.

It starts but initalising an array of size numHouses and locking node 0.
From then we queue all non locked edges in a primary queue
Then we keep selecting the minimum cost edge from the primary queue
    If that edge has 1 connection to the current tree - i.e. the start OR end is in the lockedNodes array we then know it must be in the MST
    Then we lock the edge and lock both the nodes the edge is attatched to
    We then free the primary queue and start again iterating through for (numNodes - 1) times, to ensure each node is connected.

After this is done I call a treeCost function on the graph and sum together the cost of each locked edge giving us the total cost of spanning the MST

Question 3

```
Old chip (Euclid):
Minimum operations: 12
Average operations: 38.843400
Maximum operations: 93

New chip (Euclid)
Minimum operations: 12
Average operations: 38.843400
Maximum operations: 93

Old chip (Sieve)
Minimum operations: 6
Average operations: 818.965800
Maximum operations: 2764

New chip (Sieve)
Minimum operations: 6
Average operations: 244.945500
Maximum operations: 717
```