Assignment 1 Written Problems

Problem 1

Part A

The worst case for this algorithm would involve Sam being 1 step away from the gap and picking the direction which takes him around the whole fence which takes l-1 steps to traverse.

l - 1 = l – lower order terms

= O(l) worst-case time complexity

as running time grows linearly with the number of steps, l, required to traverse the whole fence in the worst case.

Part B

An algorithm that could be used here is one in which Sam goes k steps in the left direction first, and if he does not reach the gap then Sam goes back to the origin and goes k steps in the right direction (or 2k steps to the right from where he was). In the worst case, the gap is on Sam's right and so Sam travels 3k steps in total. However, regarding time complexity, we are interested in the growth rate of the algorithm and so despite the worst case requiring 3k steps, we can ignore the constant factor 3 and conclude that this algorithm has worst case time complexity O(k) as the algorithm running time grows linearly with the number of steps, k, to the gap.

Part C

```
function FINDGAP()
    S ← 1
    while gap is not found do
        walk S steps to the right
        if gap is not found then
            walk S steps back to the origin
            S ← S + 1
            walk S steps to the left
            if gap is not found then
                walk S steps back to the origin
                S ← S + 1
                continue
        return gap found        ▷ (this return statement will be reached if the gap is found via going left or right)
```

The worst case of this algorithm involves the case in which the gap is on the opposite side of the fence from the origin and therefore the case in which S reaches l / 2. The basic operation is walking 1 step and this is performed 2S times (S steps in one direction and S steps back to the origin) per traversal with S incrementing by 1 until a l / 2 step traversal is taken (in which the journey back to the origin is not performed as the gap is found). Therefore, the time cost of this algorithm (or the total number of steps) in the worst case can be represented as:

$$\frac{\boldsymbol{l}}{\boldsymbol{2}} + \sum_{\boldsymbol{i=1}}^{\frac{\boldsymbol{l}}{\boldsymbol{2}}-\boldsymbol{1}} \boldsymbol{2i}$$

$$= \frac{\boldsymbol{l}}{\boldsymbol{2}} + \boldsymbol{2} \sum_{\boldsymbol{i=1}}^{\frac{\boldsymbol{l}}{\boldsymbol{2}}-\boldsymbol{1}} \boldsymbol{i}$$

$$= \frac{\boldsymbol{l}}{\boldsymbol{2}} + \boldsymbol{2} \frac{\left(\frac{\boldsymbol{l}}{\boldsymbol{2}} * \left(\frac{\boldsymbol{l}}{\boldsymbol{2}} - \boldsymbol{1}\right)\right)}{\boldsymbol{2}}$$

$$= \frac{\boldsymbol{l}}{\boldsymbol{2}} + \left(\frac{\boldsymbol{l}}{\boldsymbol{2}}\right)^{\boldsymbol{2}} - \frac{\boldsymbol{l}}{\boldsymbol{2}}$$

$$= \frac{\boldsymbol{l^2}}{\boldsymbol{4}} = \frac{\boldsymbol{1}}{\boldsymbol{4}} \boldsymbol{l^2}$$

Therefore, the worst-case time complexity is $O(l^2)$ as the running time grows quadratically with the number of steps required to traverse the whole fence.

Considering that l >> k, this strategy would likely be better than Strategy A as, despite its worse worst-case runtime, it will perform better on average as we expect k steps to be close to the origin in relation to the whole fence, l steps. Strategy A will require approximately l steps to reach the gap half the time (i.e. the wrong direction is chosen) when k is near the origin whereas Strategy C will reliably reach the gap in a reasonable amount of time regardless of which direction the gap is in. Furthermore, it is worth noting that a worst case of k = l/2 may not even be possible depending on the exact bounds of k and so the worst-case runtime of this strategy may not be as high as what has been calculated above.

Part D

Consider a strategy where we initialise n to 0 and move 2^n steps to the left, turn around if no gap is found and go back to the origin, move 2^n steps to the right, turn around if no gap is found and go back to the origin. We increment n by 1 and repeat this strategy until the gap is reached (i.e. k steps have been traversed in the correct direction).

Let t(k) = the number of steps required to reach the gap for any gap that is k steps from the origin.

The case in which the gap is on the right will have a worse runtime than the case in which the gap is on the left for the same k so we will calculate the runtime for the cases in which the gap is on the right from the origin.

For k = 1, t(1) = 3

(2^0 left + 2^0 to the origin + 2^0 right)

For k = 2, t(2) = 10

(t(1) + 2^0 to the origin + 2^1 left + 2^1 to the origin + 2^1 right)

For k = 4, t(4) = 24

(t(2) + 2^1 to the origin + 2^2 left + 2^2 to the origin + 2^2 right)

etc.

for k > 1: $t(k) = t(k/2) + 2^{\log2(k) - 1} + 3*2^{\log2(k)}$

$$t(k) = \begin{cases} 3, & k = 1 \\ t\left(\frac{k}{2}\right) + \frac{7k}{2}, & k > 1 \end{cases}$$

So,

$$t(k) = t\left(\frac{k}{2}\right) + \frac{7k}{2}$$

$$= \left(t\left(\frac{k}{2^2}\right) + \frac{7k}{2^2}\right) + \frac{7k}{2}$$

$$= \left(\left(t\left(\frac{k}{2^3}\right) + \frac{7k}{2^3}\right) + \frac{7k}{2^2}\right) + \frac{7k}{2}$$

$$\vdots$$

$$= t\left(\frac{k}{2^{log2(k)}}\right) + 7\left(2^{log2(k)} - 1\right)$$

$$= t(1) + 7(k-1) = 3 + 7k - 7$$

$$t(k) = 7k - 4$$

Therefore, t(k) = 7k – 4 in the worst case

So we have t(k) ϵ O(k).

Problem 2

Part B

The program works because it employs most of Prim's Algorithm (excluding the previous path tracking) to grow a spanning tree in a greedy way by selecting the minimum cost edge each time we are adding a new edge and vertex to the spanning tree. We track the vertices that have already been visited and the ones that have not been visited via a priority queue. As all edges must be in a spanning tree, selecting the locally optimal solutions will give us a globally optimal solution in the form of a minimum spanning tree. This minimum spanning tree represents the cheapest possible cable connection setup with all the possible connections considered which then allows comparison between a cable connection cost and a radio connection cost (which is simply calculated by multiplying the fixed cost of radio installation per house by the number of houses). To work out the cable connection cost we simply must sum up all the weights in the minimum spanning tree which is done as vertices are taken out of the priority queue in the program (as this means the cheapest path to this vertex has already been found). It is worth noting that this algorithm can work here because we do not have any 'negative cost' edges in this problem.

Problem 3

Summary

Old chip (Euclid):

Minimum operations: 22

Average operations: 48.843400

Maximum operations: 103

New chip (Euclid)

Minimum operations: 22

Average operations: 48.843400

Maximum operations: 103

Old chip (Sieve)

Minimum operations: 6

Average operations: 818.690000

Maximum operations: 2710

New chip (Sieve)

Minimum operations: 6

Average operations: 244.876600

Maximum operations: 717