**Q4 - Explanation of strategy used to implement concurrency**

To implement concurrency, I had to first know what parts of the solution calculation could be concurrent. After investigation, it was deemed possible to make the calculation of the **closest parent for every child node** (root node not included) as a concurrent approach.

This could be implemented with a channel and multiple go routine calls that write to that channel, then listening to the channel and storing its values in an array. This go routine for calculating the closest parent for a child node must return two key calculated variables:

1. Index of the closest parent pool.
2. The distance to that closest parent pool.

So a struct type was created (ParentChildLink) and the channel was created of this type. This allows every storage call to the channel to contain more than one calculation from the go routine instance.

The problem with this was that there is no guarantee that the results will be stored to the channel in the order the go routines were called. In other words, no synchronisation is present.

To solve this problem rather quickly, a third value, the index of the child in the sorted pools array was added to the ParentChildLink struct, which allows us to know for which child the calculated values are for, so the order in which they come in is no longer relevant.

Now by reading from the channel n – 1 times where n is the size of the sorted pools array, we can create another array for the children’s closest parents and use the following key line (in pseudocode) to fill it up:

ClosestParentsArray[parentChildLink.ChildIndex] = parentChildLink

The array will be filled up without being concerned of the order it is being filled up.

Note: Almost every area of the code is commented if you require more information.