“If we have a sequence of numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,13, 14, 15 and we want this sequence to be in an order such that every number and the number next to it when added together form another number that is a perfect square. Perfect squares are numbers such as 4, 9, 16 etc which can be formed by squaring and integer.

An example of a correct solution to the above problem is:

8, 1, 15, 10, 6, 3, 13, 12, 4, 5, 11, 14, 2, 7, 9

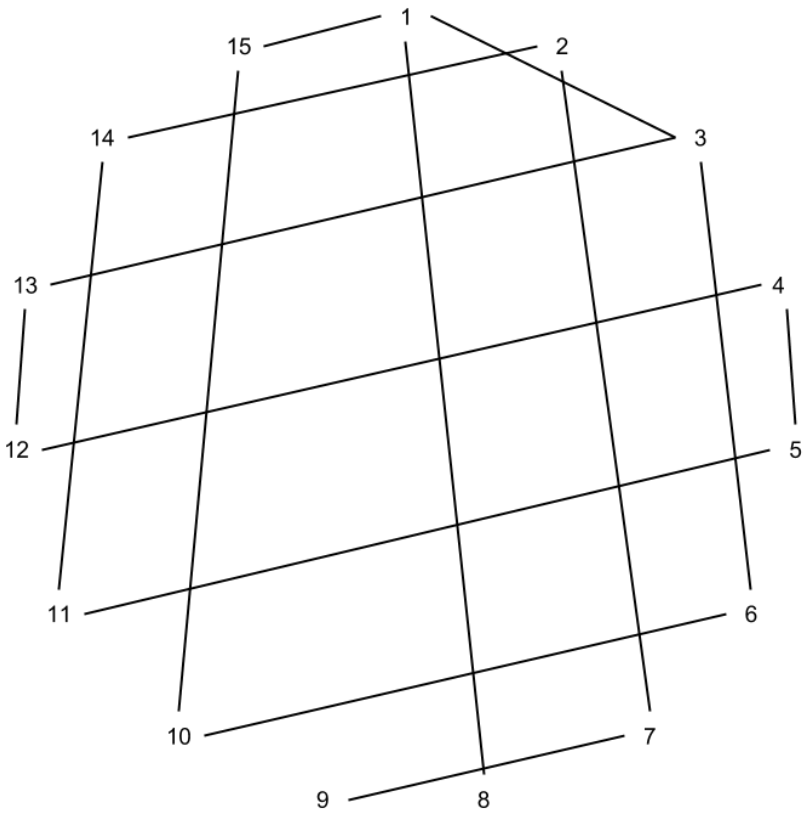
Write a program in pseudo-code or code which can find solutions to this problem.”

My first instinct during the interview was to write down all the patterns and facts. Thing such as what each number that is a perfect square becomes when I subtract the current number from it. This laid some groundwork for my next step which was to continue this step generally and make a table of it.

	4	9	16	25
1	3	8	15	-
2	-	7	14	-
3	1	6	13	-
4	-	5	12	-
5	-	4	11	-
6	-	3	10	-
7	-	2	9	-
8	-	1	-	-
9	-	-	7	-
10	-	-	6	15
11	-	-	5	14
12	-	-	4	13
13	-	-	3	12
14	-	-	2	11
15	-	-	1	10

Where the column headings are the perfect squares up the the largest that is less than or equal to the largest possible sum (n + (n + 1)), the row headings are the numbers 1 to n, and the cells are the result of subtracting the column heading from the row heading or a dash ‘-’ if the result is less than or equal to ‘1’.

With this information I decided to do what my Calculus class has long taught me to do, ‘draw the picture’. I remember from my algorithms class it’s best to draw all the numbers out in a clock-like pattern circle as vertices, and connect all these ‘nodes’ with lines to represent an ‘edge’. This resulted in the following diagram:



The question from observation now becomes a ‘longest path’ problem. This is a special case known as a ‘Hamiltonian Path’, which is a path that visits every node but is not necessarily a cycle, and does not visit the same node twice. This is an NP-hard problem and thus takes exponentially more time the larger ‘n’ is to find a solution if one exists.

Two observations thus far that optimize the search for a solution should be noticed:

- If any node has ‘0’ edges to it, the solution is the empty set. This occurs only for very small ‘n’.

- The count of nodes that have only one edge to them can be kept, and if the count increases to more than two, the solution is the empty set. A list can only have two endpoints.

There are two ways to find a solution:

With observation that there is at least one, and no more than two vertices, I know that I can start with one of these and form a chain from there. I test exhaust all options with this first node to either find a solution, or return the empty set. This is less complex than finding the Hamiltonian path.

Brute force is required if this optimization is not available and each vertex must be checked on at a time and exhaust all it's options at each choice point during traversal. This can take much longer, especially for larger numbers.