1

In order to encode the adjacency matrix using binary string, I choose to transform the integer n into a binary number. This n represents the number of vertices in the graph. Then there are n^2 bits following. The value of bit m will be 1 if there is an edge from vertex (m/n) to vertex (m % n), and zero if there is no such an edge.
In order to encode the adjacency list, I will use a different encoding of integers, call it f(n). I will place a 0 immediately after every bit in the usual representation. In addition, the reason I use this encoding is because any sequence of integers encoded in this way cannot contain the string 11 and must contain at least one zero.

If I need to convert the adjacency list to adjacency matrix, I need to declare a 2d array. Then I use loop to iterate through each adjacency list and I mark each pair of connected vertices in the array to be 1. For example, in the above picture, vertex 1 connect to vertex 2, so I mark array [1,2] to be 1, but mark array [2, 1] to be 0. The time complexity of doing this transformation is polynomial time.

In the adjacency matrix, each row represents the connectivity of a vertex with all the other vertices. If I need to convert the adjacency matrix to adjacency list, I need to iterate through every row in the matrix. If the value of an element is 1, then connect the one of the corresponding pair of vertices to one another. The time complexity of this algorithm is polynomial time. In conclusion, these two representations are polynomially related.

2.

In order to show that vertex cover problem is a NP Complete problem, the first thing to prove is that this problem is a NP problem. I need to show that this vertex cover problem can be solved in polynomial time. The solution is

```
Solution (G, k):
  Let count to be 0
  For each vertex v in G.V:
      Remove all edges adjacent to v in G.E
      Count += 1
      If count == k:
          Return true
      Else: return False
```

This solution has polynomial time complexity. This solution shows that vertex cover problem is a NP problem. The second thing I need to prove is this problem

is a np-hard problem. Clique is a NP complete problem, so it is a np hard problem. I need to show that clique problem is reducible to vertex cover problem. I need to check for the existence of a clique of size k in G. consider the graph G' which consists of all the edges that are not in G. the problem of finding whether a clique of size k exists in G is the same as the problem of finding whether there is a vertex cover of size V – k in G'. assume there is a clique of size k in G. Let the set of vertices in the clique be V'. in G', let us pick any edge (u, v). Then at least one of u or v must be in the set V – V'. all edges covered in G' are covered by vertices in the set V – V'. Now assume that there is a vertex cover V'' of size V – k in G'. This means that all edges in G' are connected to some vertex in V''. As a result, if we pick any edge from G', both of vertices cannot be outside the set V''. This means all edges whose vertices are outside of V'' are the edges that constitute a clique of size k. As a result, I can say there is a clique of size k in G if and only if there is a vertex cover of size V – K in G', and hence, any instance of the clique can be reduced to an instance of the vertex cover problem. since clique problem is a NP hard problem, then vertex cover problem is a np complete problem.

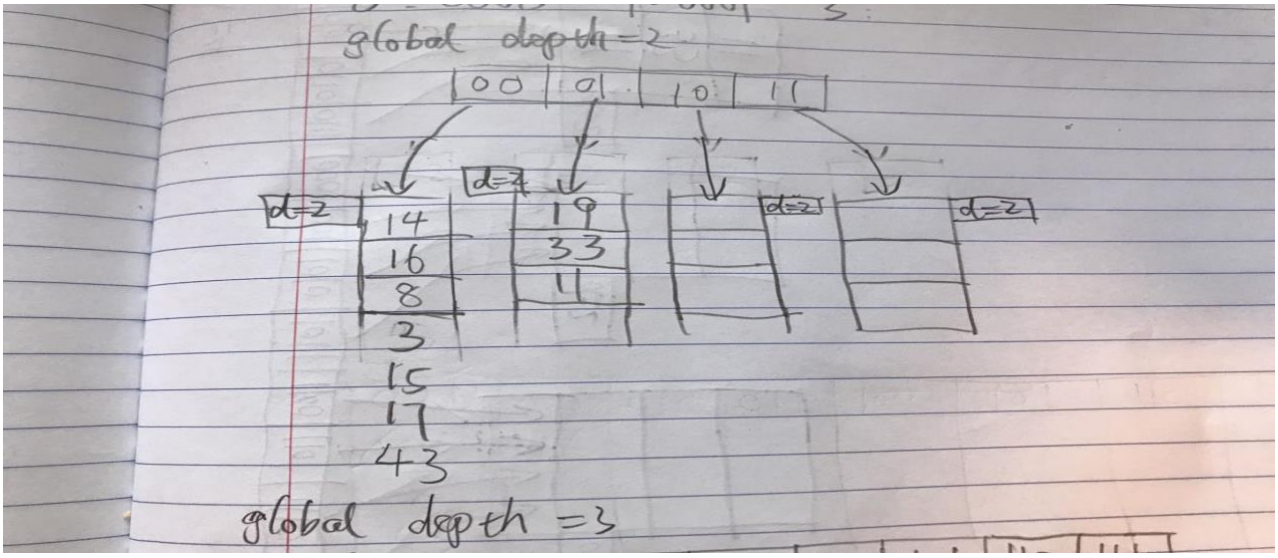
3.

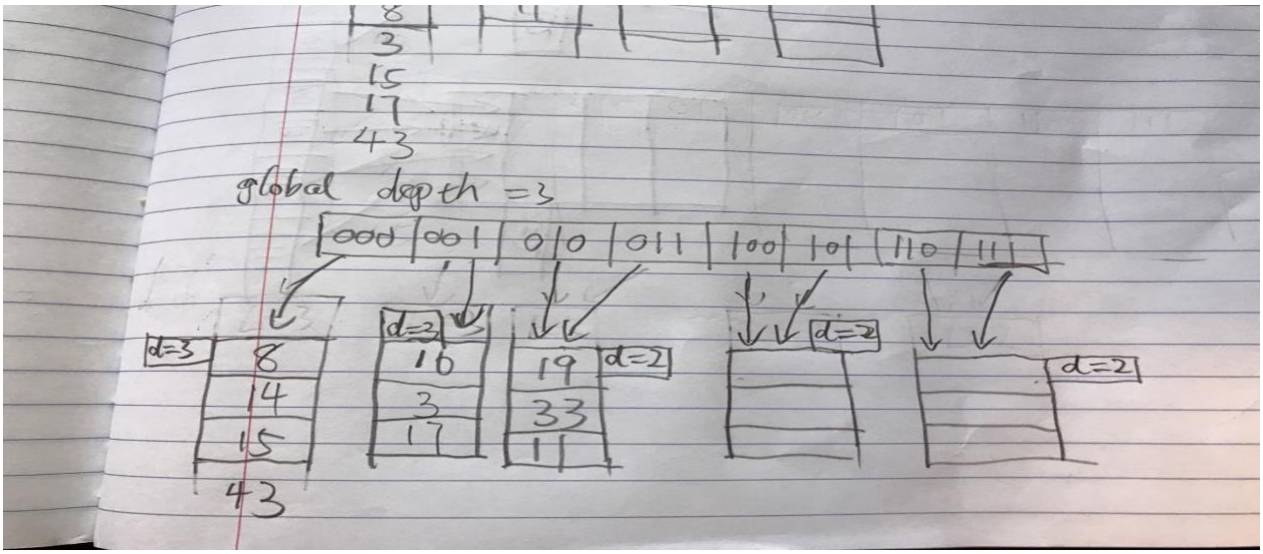
```
areRotation (T, V):
  length of T is called l1
  length of V is called l2

  if l1 != l2:  return False
  tmp = T + T                  # tmp is the concatenation of two Ts
  Next to figure out the occurrence of V in this String tmp by using the KMP
algorithm
  if the occurrence is more than 0, then return True
  otherwise, return False
```

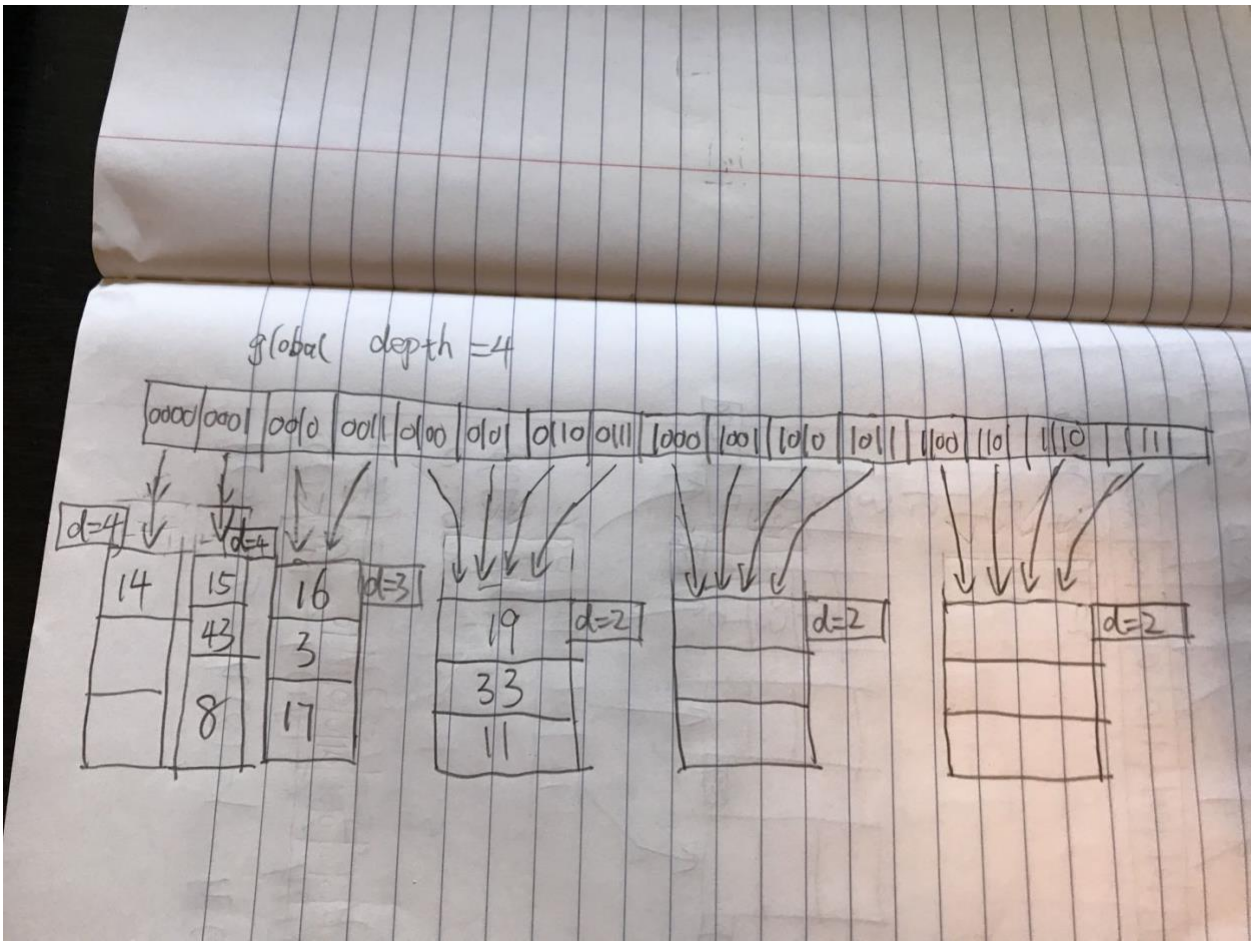
time complexity is O (n/m) n means the length of T, m means the length of V
space complexity is O (1)

4.
H (3) = 3, h(8) = 1, h(11) = 4, h(14) = 0, h(15) = 1, h(16) = 2, h(17) = 3, h(19) = 5, h(20) = 6, h(33) = 5, h(43) = 1, h(48) = 6

Decimal to binary:
0: 0000. 2: 0010. 1: 0001. 3: 0011. 4: 0100. 5: 0101. 6: 0110






global depth =4
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
d=4
d=4
14
15
43
8
16
3
17
d=3
19
33
11
d=2
d=2
d=2