1 Question 1

$n = 25$ and $\forall (a, b) \in V^2, \mathbb{P}((a, b) \in E) = p = 0.2$.
If we don't consider self-loops, we have $degree(a) = \sum_{b \in V - \{a\}} \mathbb{1}_{(a,b) \in E}$.
Therefore, $\forall a \in V$

$$\begin{aligned}\mathbb{E}[degree(a)] &= \sum_{b \in V - \{a\}} \mathbb{E}[\mathbb{1}_{(a,b) \in E}] \\ &= \sum_{b \in V - \{a\}} \mathbb{P}((a, b) \in E) \\ &= (|V| - 1) \times p = 24 \times 0.2 = \boxed{4.8}\end{aligned}$$

Similarly, if $p = 0.4, \forall a \in V, \mathbb{E}[degree(a)] = (|V| - 1) \times p = 24 \times 0.4 = \boxed{9.6}$

2 Question 2

The readout function is responsible for combining the information from individual node or edge embeddings to create a graph-level representation. In our problem we expect the same graph representation coming from 2 differently ordered sets of the same node embeddings. It means that the readout function needs to be **permutation invariant**. Trainable linear layers do not naturally handle this permutation invariance. In contrast, operations like sum or mean are permutation-invariant, as they treat all nodes equally. This is what comes to mind when looking at figure 1. There is no special order between G_1, G_2 and G_3 so fully connected layer would not be adequate. Moreover, if not sufficiently trained, a fully connected layer would confuse the nodes, not knowing to which graph they belong. The advantage of sum readout is the possibility to directly code from which graph a node comes.
Apart from permutation invariance we can point out other drawbacks of fully connected layers. Fully connected layers have a quadratic time complexity with respect to the number of nodes in the graph. On the other hand, sum or mean operations have a linear time complexity, making them more scalable. Also, fully connected layers having a large number of parameters, they may overfit, especially when the amount of training data is limited. Sum or mean operations tends to generalize better to unseen graphs.

3 Question 3

The output produced in Task 8 is

```
neighbor_aggr = mean and readout = mean
representation:
 [[ 0.07760121  0.11595523 -0.4233936   0.42726734]
 [ 0.07760121  0.11595523 -0.4233936   0.42726734]
 [ 0.07760121  0.11595523 -0.4233936   0.42726734]
 [ 0.07760122  0.1159552  -0.4233936   0.42726737]
 [ 0.07760122  0.1159552  -0.4233936   0.42726737]
 [ 0.07760122  0.11595523 -0.4233936   0.42726737]
 [ 0.07760122  0.11595523 -0.4233936   0.42726737]
 [ 0.07760121  0.11595523 -0.4233936   0.42726737]
 [ 0.0776012   0.11595523 -0.4233936   0.4272674 ]
 [ 0.0776012   0.11595526 -0.4233936   0.4272674 ]]

neighbor_aggr = mean and readout = sum
representation:
 [[ -4.2675605   1.522087   -3.5237827  -8.235901 ]
 [ -4.7016435   1.6595953  -3.8642595  -9.06266  ]
 [ -5.135726    1.7971035  -4.2047343  -9.889421 ]
 [ -5.5698094   1.9346119  -4.5452113 -10.716181 ]
```

```
 [ -6.003892    2.072121   -4.8856874 -11.542943 ]
 [ -6.437974    2.2096288  -5.2261634 -12.369703 ]
 [ -6.872057    2.3471372  -5.56664   -13.196464 ]
 [ -7.3061404   2.4846454  -5.9071164 -14.023224 ]
 [ -7.7402225   2.6221538  -6.2475915 -14.849983 ]
 [ -8.174305    2.7596626  -6.5880685 -15.676744 ]]

neighbor_aggr = sum and readout = mean
representation:
 [[-0.4473583  -0.93896496  0.02548662  0.9355992 ]
 [-0.4473583  -0.93896496  0.02548659  0.93559927]
 [-0.44735825 -0.93896496  0.02548663  0.9355992 ]
 [-0.44735822 -0.93896496  0.02548663  0.9355992 ]
 [-0.44735822 -0.93896496  0.02548663  0.9355992 ]
 [-0.44735813 -0.93896496  0.02548664  0.93559927]
 [-0.44735813 -0.93896496  0.02548664  0.93559927]
 [-0.44735813 -0.93896496  0.02548664  0.9355992 ]
 [-0.44735813 -0.93896496  0.02548666  0.9355992 ]
 [-0.4473582  -0.93896496  0.02548666  0.9355992 ]]

neighbor_aggr = sum and readout = sum
representation:
 [[-3.7061605   8.700741    0.3713502   1.6469654 ]
 [-4.0706587   9.574213    0.39214268  1.8214387 ]
 [-4.435158   10.447684    0.41293326  1.9959124 ]
 [-4.799655   11.321155    0.43372527  2.1703856 ]
 [-5.164154   12.194627    0.45451632  2.344859  ]
 [-5.528651   13.068099    0.47530738  2.5193317 ]
 [-5.8931518  13.941569    0.49609938  2.6938057 ]
 [-6.2576485  14.81504     0.51688945  2.8682785 ]
 [-6.622148   15.688514    0.53767955  3.042752  ]
 [-6.9866457  16.561983    0.55847156  3.2172241 ]]
```

We observe that:

- neighbor_aggr = mean and readout = mean give the same graph description for every size.
- neighbor_aggr = mean and readout = sum give different graph descriptions depending on the size.
- neighbor_aggr = sum and readout = mean give the same graph description for every size.
- neighbor_aggr = sum and readout = sum give different graph descriptions depending on the size.

We deduce that mean readout describes the different size graphs the same way. GNN better captures the graph size with sum readout. The sum changes even if the mean does not, therefore the sum allows to count the number of nodes and use it in the graph representation.
We cannot deduce anything about neighbor aggregation.

4 Question 4

If we take $G_1 = C_3 \cup C_5$ and $G_2 = C_8$, where C_k is the cycle graph of size k, we obtain:

```
G1 representation: [ -0.2893014 -12.322963    5.6595573   7.44433  ]
G2 representation: [ -0.2893014 -12.322963    5.6595573   7.44433  ]
```

and G_1 and G_2 are not isomorphic.

5 Task 11

We observe

```
G1 representation: [ 3.1621916 -8.330429   1.5102267  1.1063328]
G2 representation: [ 3.1621916 -8.330429   1.5102267  1.1063328]
```

G_1 and G_2 have the same representation which means that the GNN is not able to make the difference between the two, eventhough there are not isomorphic.