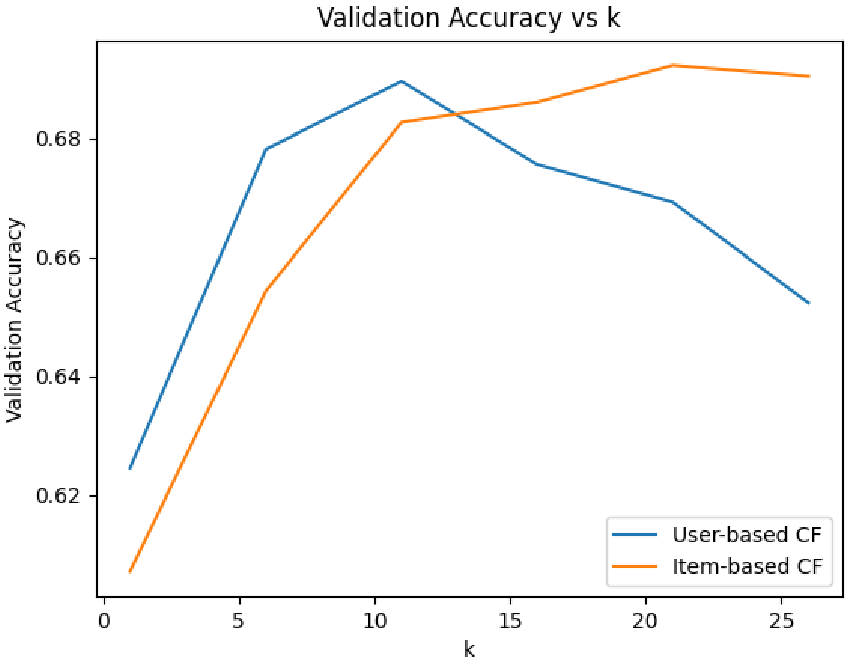
Final Report

Part A

Question 1

(a) (b) (c) The accuracy on the validation data with $k \in \{1, 6, 11, 16, 21, 26\}$ on user-based and item-based collaborative filtering is as follows:



Test Accuracy on user-based CF with k* = 11: 0.6841659610499576

Test Accuracy on item-based CF with k* = 21: 0.6816257408975445

(d) The test on user-based CF is slightly better than item-based CF.

Additionally, the test accuracy on user-based CF cost less time than item-based CF.

Therefore, user-based CF is better than item-based CF in this case.

(e)
- The KNN algorithm is computational expensive for large datasets.
- The Curse of Dimensionality: In high dimensions, “most” points are approximately the same distance and the nearest neighbors are not very useful.

Question 2

(a) Given the probability that the question j is correctly answered by student i is:

$$p_{ij} = \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}$$

The log-likelihood for all students is derived as follows:

$$\begin{aligned}
\log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) &= \sum_{i,j} (c_{ij} \log p_{ij} + (1 - c_{ij}) \log(1 - p_{ij})) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} \left(c_{ij} \log \left(\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right) + (1 - c_{ij}) \log \left(1 - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right) \right) \\
&= \sum_{i=1}^{n} \sum_{j=1}^{m} (c_{ij}(\theta_i - \beta_j) - \log(1 + \exp(\theta_i - \beta_j))),
\end{aligned}$$

where c_{ij} is the binary response of student i to question j.

The log-likelihood with respect to θ_i is:

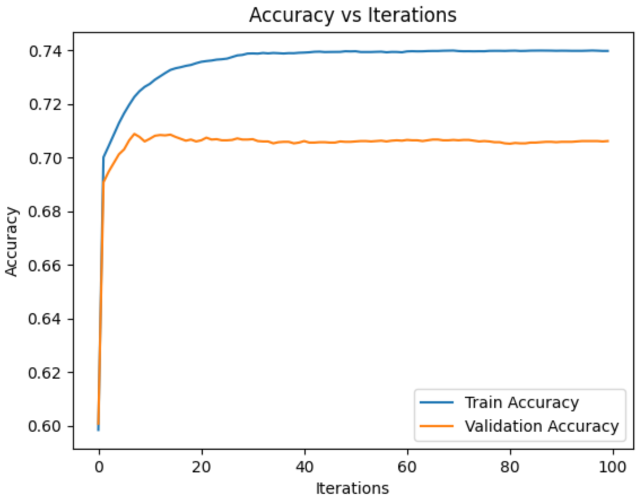
$$\begin{aligned}
\frac{\partial \log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})}{\partial \theta_i} &= \sum_{j=1}^{m} \left(c_{ij} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right) \\
&= \sum_{j=1}^{m} (c_{ij} - p_{ij}).
\end{aligned}$$

The log-likelihood with respect to β_j is:

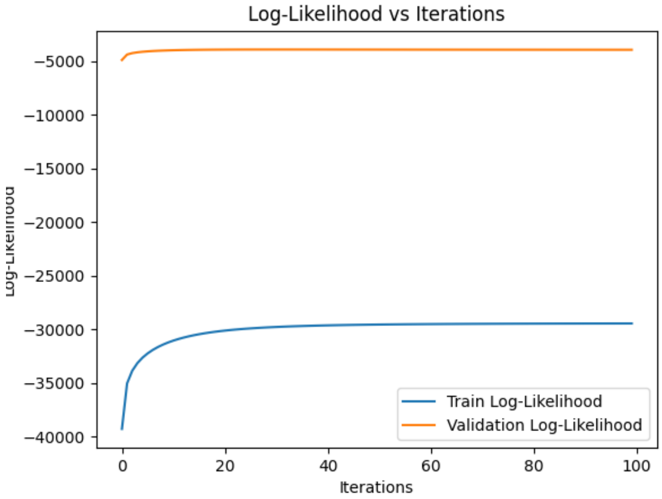
$$\begin{aligned}
\frac{\partial \log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})}{\partial \beta_j} &= \sum_{i=1}^{n} \left(c_{ij} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right) \\
&= \sum_{i=1}^{n} (c_{ij} - p_{ij}).
\end{aligned}$$

(b) The hyperparameters I selected are: learning rate $= 0.01$ and iterations $= 100$.

The training and validation accuracies vs iterations are in the graph below:



The log-likelihoods vs iterations are in the graph below:

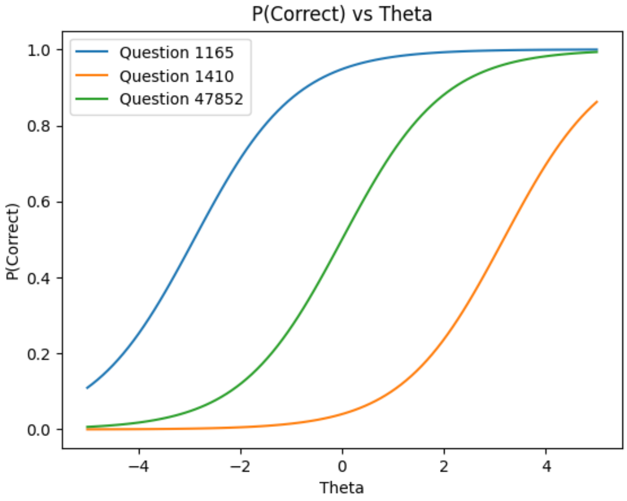


(c) The Final Validation Accuracy: 0.7063223257126728

The Final Test Accuracy: 0.707310189105278

(d) I select the lowest difficulty question j_1 (Question 1165), the highest difficulty question j_2 (Question 1410) and the average difficulty question j_3 (Question 47852).

The probability of the correct response is in the graph below:



The shape of the curves are like the sigmoid function as expected.

Fix a question j. As θ_i increases, the probability of the correct response p_{ij} increases. This means if a student has a higher ability, the probability of the correct response increases.

Fix a student i. As β_j increases, the probability of the correct response p_{ij} decreases. This means if a question has a higher difficulty, the probability of the correct response decreases.

Question 3

We choose Option 2

(a)
- ALS breaks down large matrices into lower-dimensional matrices, while neural networks model non-linear relationships through layers.
- ALS is less flexible than neural networks because it is designed for matrix factorization, whereas neural networks can model non-linear relationships.
- ALS is more computationally efficient than neural networks for sparse datasets because neural networks require significant computational resources.

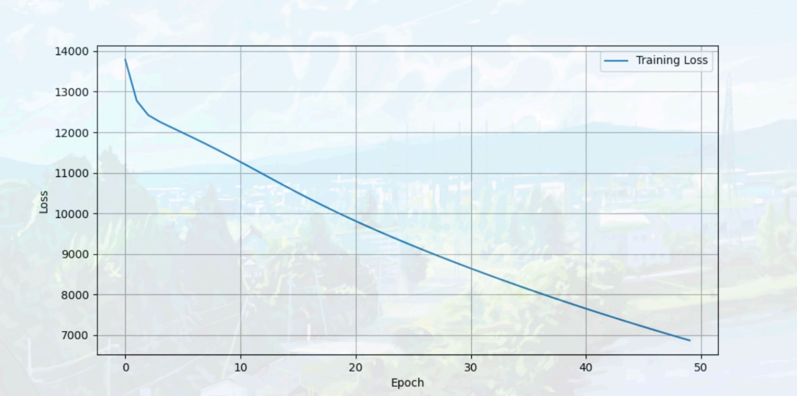
(b) The implementation is in `neural_network.py`.

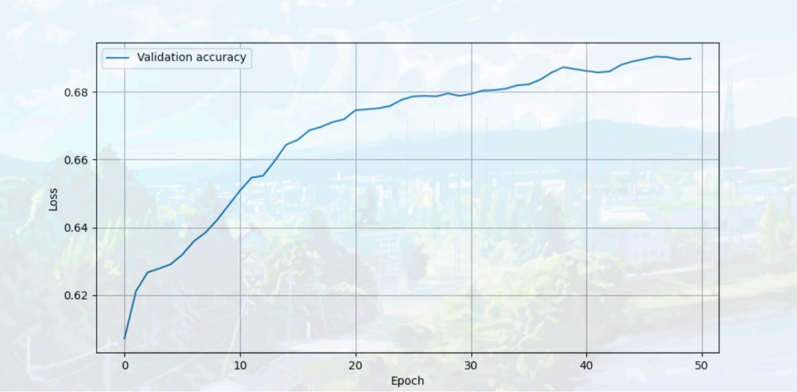
(c) The optimization hyperparameters we chose are:

`k = 50, lr = 0.01, num_epoch = 50`

The Validation Accuracy we obtained is 0.68981.

(d) The plot with `k = 50, lr = 0.01, num_epoch = 50` is shown below:





The Final Test Accuracy is 0.68558.

(e) The best regularization penalty we found is $\lambda = 0.01$. With this value of λ, we obtained:

Final Validation Accuracy: 0.67824

Final Test Accuracy: 0.68078

The model performed about the same with the regularization penalty. This may be because our model is already well-regularized and does not overfit, or only has negligible overfitting issues.

Question 4

The Final Validation Accuracy is: 0.66286

The Final Test Accuracy is: 0.66949

Ensemble process:

We use three neural network models to implement bagging ensemble. We first randomly sample with replacement from the training dataset. Then we train three different neural networks independently for each training sample. These three neural networks are independent and can run individually. After all models are trained, we use them to make predictions separately, and we take the average of each of their predictions as our final prediction.

Better or Not:

No, the bagging model is about the same performance as the single neural network model, so it doesn't improve the performance.

Reason:

Ensembling the same model trained on different data subsets lacks model diversity, which does not always improve the model performance.

Additionally, the small training subset could be another problem. When the training set is small, there could be an issue that the training subset is even smaller, so that each model is not well trained, which results in poor performance.

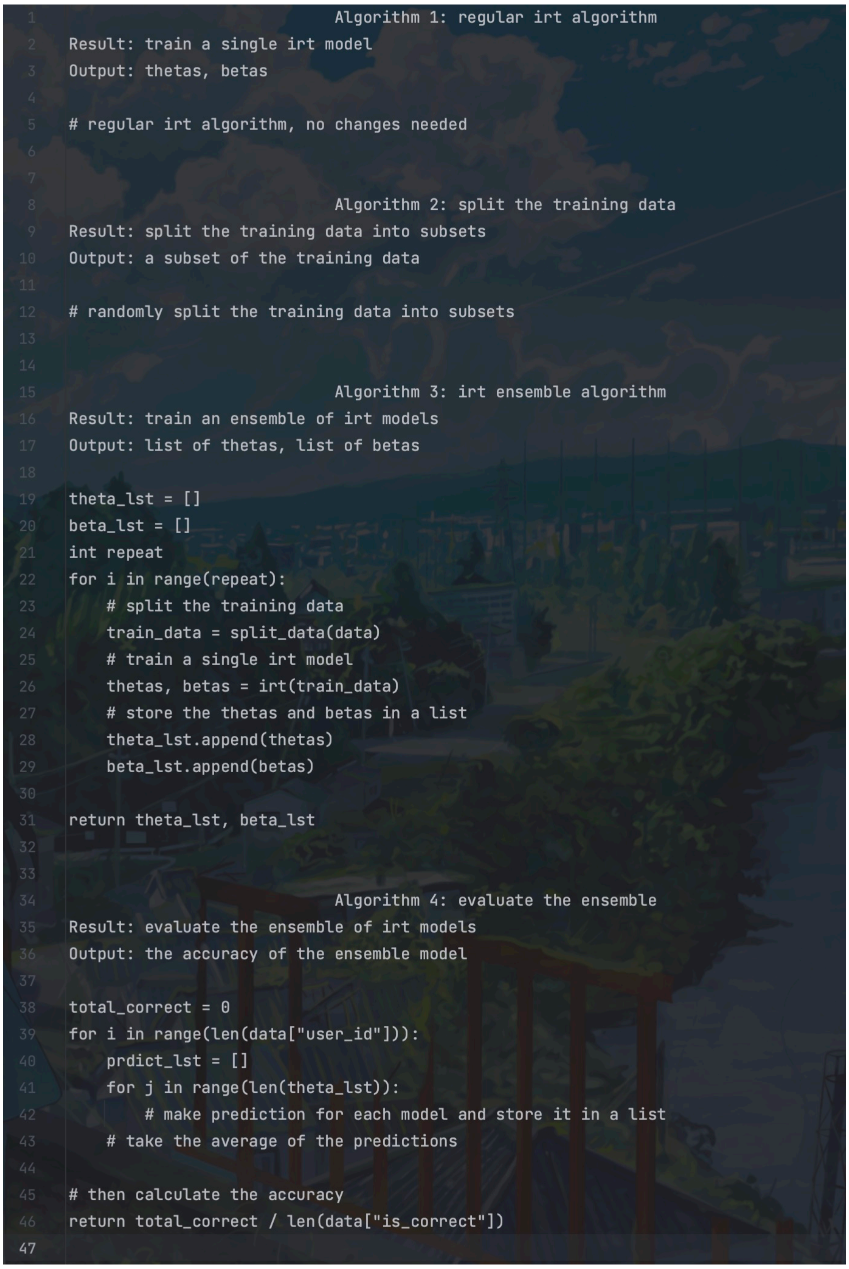
Part B

Formal Description

The performance of the IRT algorithm in part A is not satisfactory. We believe that the main reason is that a single decision tree is a high-variance model. We believe that the decision tree in part A overfits the training data, which makes the model's generalization ability poor.

Therefore, we decided to reduce the variance by averaging the predictions of multiple decision trees using a random forest. Each tree is trained on a different random subset of the training data. The results are aggregated at the end to make the model more robust by smoothing the data.

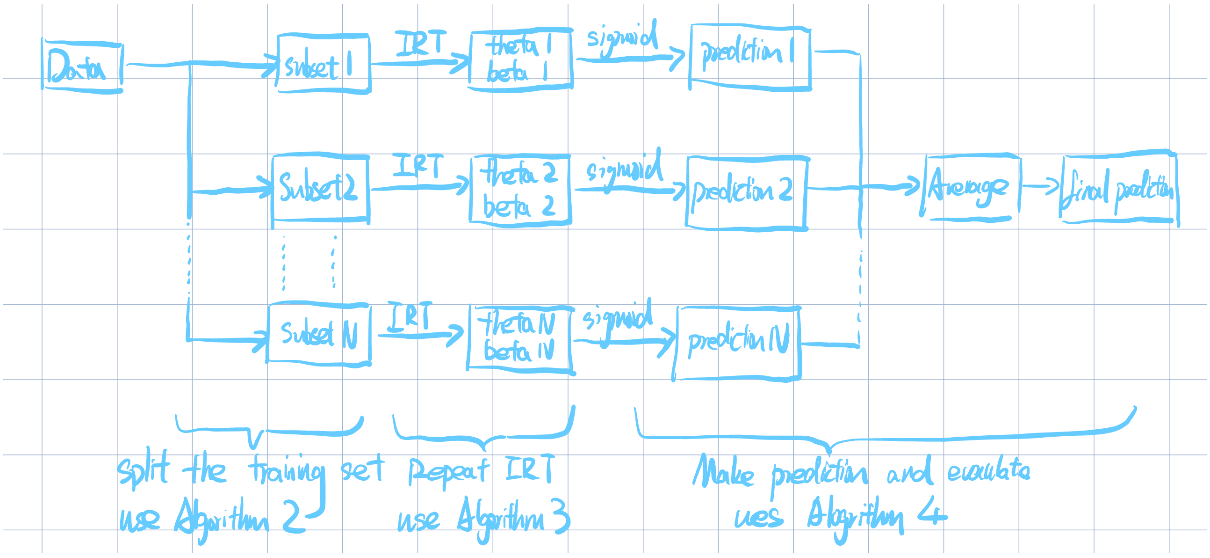
Algorithm Box



```
                            Algorithm 1: regular irt algorithm
Result: train a single irt model
Output: thetas, betas

# regular irt algorithm, no changes needed


                            Algorithm 2: split the training data
Result: split the training data into subsets
Output: a subset of the training data

# randomly split the training data into subsets


                            Algorithm 3: irt ensemble algorithm
Result: train an ensemble of irt models
Output: list of thetas, list of betas

theta_lst = []
beta_lst = []
int repeat
for i in range(repeat):
    # split the training data
    train_data = split_data(data)
    # train a single irt model
    thetas, betas = irt(train_data)
    # store the thetas and betas in a list
    theta_lst.append(thetas)
    beta_lst.append(betas)

return theta_lst, beta_lst


                            Algorithm 4: evaluate the ensemble
Result: evaluate the ensemble of irt models
Output: the accuracy of the ensemble model

total_correct = 0
for i in range(len(data["user_id"])):
    prdict_lst = []
    for j in range(len(theta_lst)):
        # make prediction for each model and store it in a list
    # take the average of the predictions

# then calculate the accuracy
return total_correct / len(data["is_correct"])

```

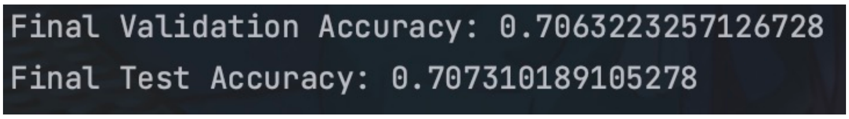
By training multiple decision trees on different data subsets in the form of random forests, the overfitting problem of the model can be reduced.

Idea Diagram



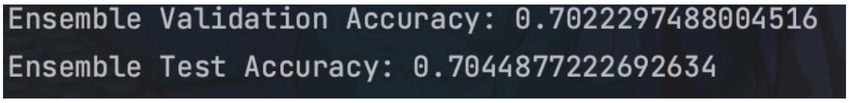
Comparison or Demonstration

For comparison, when using the single irt algorithm, we obtained the following statistics. We use this group of data as the baseline models:



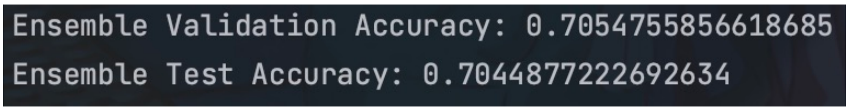
```
Final Validation Accuracy: 0.7063223257126728
Final Test Accuracy: 0.707310189105278
```

When using a random forest consisting of two decision trees, we get the following statistics:



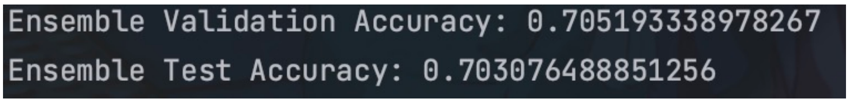
```
Ensemble Validation Accuracy: 0.7022297488004516
Ensemble Test Accuracy: 0.7044877222692634
```

When using a random forest consisting of three decision trees, we get the following statistics:



```
Ensemble Validation Accuracy: 0.7054755856618685
Ensemble Test Accuracy: 0.7044877222692634
```

When using a random forest consisting of four decision trees, we get the following statistics:



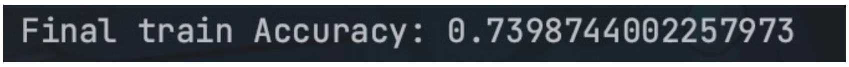
```
Ensemble Validation Accuracy: 0.705193338978267
Ensemble Test Accuracy: 0.703076488851256
```

After comparison, our model does not significantly improve the accuracy.

Experiment to Test Our Hypothesis

We use the accuracy of the training dataset and the accuracy of the validation dataset to determine whether the model has signs of overfitting. If the training accuracy is significantly higher than the validation accuracy, it means that the model is too sensitive to the training dataset and has signs of overfitting. On the contrary, if the training accuracy and validation accuracy are close, it means that the model does not have overfitting.

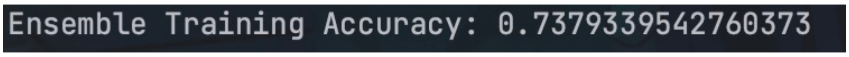
The test accuracy of the original model is as follows:



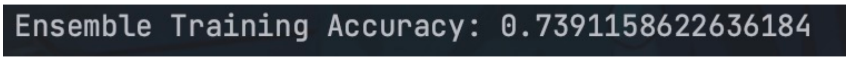
```
Final train Accuracy: 0.7398744002257973
```

We can see that the training accuracy of the original model is significantly higher than the validation accuracy, indicating that the original model may be overfitting.

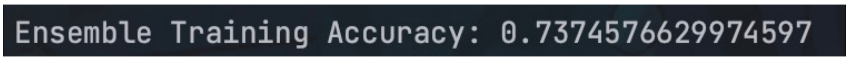
The random forests consisting of 2, 3, and 4 decision trees have the following statistics:



```
Ensemble Training Accuracy: 0.7379339542760373
```



```
Ensemble Training Accuracy: 0.7391158622636184
```



```
Ensemble Training Accuracy: 0.7374576629974597
```

Unfortunately, Random Forest only slightly reduces the gap between training accuracy and validation accuracy, and the overfitting problem still exists.

Limitations

As mentioned above, implementing Random Forest algorithm did not improve our original model significantly. This could be due to the following reasons:

- The given dataset is not large enough or diverse enough. Random Forest is known to perform well on large-scale datasets with many features. In our improved model, we split the dataset into many subsets and trained a Random Forest model on each subset. However, the dataset may not be large enough to benefit from this approach.

  Hence, changing the dataset may improve the performance of the Random Forest algorithm.

- In our Random Forest algorithm, we assumes all predictions from the base models are equally important and we average all predictions of each decision tree to get the final prediction. This may not be the case in practice. Some models may perform better on certain types of data, and simply averaging their predictions may not be the best approach.

  We can use a weighted average of the predictions to give more importance to the better performing models, but this requires tuning the weights, which can be time-consuming.

- When ensembling multiple models, we need to ensure generalization. It may be because the base models are too similar and the overfitting problem still exists.

  We may need to use regularization to limit the complexity of the base models.

- The Random Forest algorithm is computationally expensive. It requires a lot of time to tune the hyperparameters and train the model. We may not have been able to find the best hyperparameters in the time we had.

- IRT model itself may be limited in predicting the student's performance. If we integrate more complex models in the ensemble, we may improve the performance of the model.

Contributions

Part A

- Quesion 1 and 2: Mingzhe Zhang
- Question 3 and 4: Zhiyuan Meng

Part B

- Question 1, 2 and 3: Zhiyuan Meng
- Question 4: Mingzhe Zhang