FINAL REPORT

- ARTIFICAL INTELLIGENCE-

MEMBERS

NAME	ID
TRAN QUOC BAO	521H0494
BUI HAI DUONG	521H0220
BUI ANH PHU	521H0508
NGUYEN HOANG PHUC	521H0511
HOANG DINH QUY VU	521H0517

TASK 1 – CONSTRAINT SATISFACTION

❑ NQueenSolver:

- **__init__():** initializes the size of the chessboard and sets all positions to -1.
- **__is_safe():** checks if a queen can be placed at position(x, y) on the board without being attacked by any other queens.
- **__backtracking():** recursively tries out all possible positions for each queen and checks if it is safe to place it there.
- **solve():** finds out the N-Queens problem using __backtracking() and print solution if it exists.

❑ EightQueenSolver:

- Inherits from class NQueenSolver.

TASK 1 – CONSTRAINT SATISFACTION

❑ Pseudo code:

- **Function** __is_safe(x, y) **returns** a Boolean value

  if AtTheSameRow/Colum/Diagonal(queen[x][y], another queen) → **return** False

  else → **return** True

- **Function** __backtracking(x = 0) **recurses**

  /* taking an optional x representing the current row being considered */

  if x equals size of board → place successfully all queens.

  else: loop do each column of the current row:

  - **if** safe → place the queen in that position and __backtracking(the next row).
  - **if** not safe → backtrack by resetting the current position to -1 and trying the next column.

TASK 1 – CONSTRAINT SATISFACTION

❑ Result:

```
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
```

EightQueenSolver()

```
Q . . . . . . . . . . .
. . Q . . . . . . . . .
. . . . Q . . . . . . .
. . . . . . . Q . . . .
. . . . . . . . . Q . .
. . . . . . . . . . . Q
. . . . . Q . . . . . .
. . . . . . . . . . Q .
. Q . . . . . . . . . .
. . . . . . Q . . . . .
. . . . . . . . Q . . .
. . . Q . . . . . . . .
```

NQueenSolver(12)

TASK 2 – ADVERSARIAL SEARCH

- **Node:**
  - **__init__():** initializes identifier, value.
  - **__str__():** represents the object.

- **MinimaxDecision:**
  - **__init__():** initializes root, terminalStates, successors.
  - **read():** reads a file and constructs a tree.
  - **print():** prints out the tree using backtracking.
  - **__backtracking():** a helper function for print() that recursively backtracks through the tree.
  - **run():** runs __minimax() on the tree.
  - **__minimax():** a helper function for run() that implements the minimax algorithm.
  - **getDepth():** gets the depth of the tree.

TASK 2 – ADVERSARIAL SEARCH

❑ Pseudo code:

- **Function** __backtracking(node) **recurses**

```
print(node)
if node.identifier is successor.key:
    loop do each successor of node → recursively call __backtracking(the current successor)
```

- **Function** __minimax(node, depth, flag) **returns** a value of node

```
if node is a terminal node or depth == 0 → return node.value
if flag is True:
        MAX ← -∞
        loop do each successor of node:
            MAX ← max(MAX, __minimax(successor, depth-1, False))
        return MAX
else:
        MIN ← +∞
        loop do each successor of node:
            MIN ← min(MIN, __minimax(successor, depth-1, True))
        return MIN
```

TASK 2 – ADVERSARIAL SEARCH

❑ **Result:**

```
(n00, None)
(n10, None)
(n20, None)
(n30, None)
(n41, 4)
(n42, 3)
(n43, 5)
(n31, None)
(n44, 2)
(n45, 1)
(n21, None)
(n32, None)
(n46, 4)
(n47, 2)
(n48, 3)
(n22, None)
(n33, None)
(n49, 5)
(n410, 4)
(n34, None)
(n411, 7)
(n35, None)
(n412, 3)
(n413, 2)
(n11, None)
...
(n311, None)
(n425, 5)
(n426, 3)
(n427, 1)
```

Print each node and value using backtracking algorithm

TASK 3 – LOGICAL AGENTS

❑ NQueenSolver:

- **__init__():** initializes the NQueenSolver class with the size of chessboard.
- **add_row_constraints():** ensures that each row has exactly one true value.
- **add_col_constraints():** ensures that each column has exactly one true value.
- **add_diagonal_constraints():** ensures that no two queens are on the same diagonal.
- **add_at_most_one():** ensures that at most one queen is placed on a given set of squares.
- **solve():** solves the N-Queens problems using a SAT solver–Glucose3.

❑ EightQueenSolver:

- Inherits from class NQueenSolver.

TASK 3 – LOGICAL AGENTS

- **Function** add_row_constraints(clauses, vars, n):
  loop do i,j in range(n):
  add each vars[i][j] to clauses
  loop do j1 in range(n) and j2 in range(j1+1,n)
  add each [-vars[i][j1], -vars[i][j2]] to clauses

- **Function** add_col_constraints(clauses, vars, n):
  loop do i,j in range(n):
  add each vars[i][j] to clauses
  loop do i1 in range(n) and i2 in range(i1+1,n)
  add each [-vars[i1][j], -vars[i2][j]] to clauses

- **Function** add_diagonal_constraints(clauses, vars, n):
  loop do k in range(1-n, n)
  diag1 ← [vars[i][i-k] **loop do** i in range(n) if 0 <= i-k < n]
  diag2 ← [vars[i][n-1-(i+k)] **loop do** i in range(n) if 0 <= n-1-(i+k) < n]
  add at most one diag1 to clauses **if** diag1.len > 1
  add at most one diag2 to clauses **if** diag2.len > 1

- **Function** add_at_most_one(clauses, lits):
  loop do i in range(lits.len) and j in range(i+1, lits.len)
  add each [-lits[i], -lits[j]] to clauses

TASK 3 – LOGICAL AGENTS

❑ Result:

```
There are 92 solutions to the 8 x 8 problem
Random solution:
. . . . . Q . .
. . Q . . . . .
. . . . Q . . .
. . . . . . . Q
Q . . . . . . .
. . . Q . . . .
. Q . . . . . .
. . . . . . Q .
```

EightQueenSolver()

```
There are 14200 solutions to the 12 x 12 problem
Random solution:
. . . . . . . . . . . Q
. . . . . . Q . . . . .
. . Q . . . . . . . . .
. . . . . . . Q . . . .
. Q . . . . . . . . . .
. . . . Q . . . . . . .
. . . . . . . . Q . . .
. . . . . . . . . . Q .
Q . . . . . . . . . . .
. . . Q . . . . . . . .
. . . . . Q . . . . . .
. . . . . . . . . Q . .
```

NQueenSolver(12)

TASK 4 – MACHINE LEARNING

Load data from mnist.npz

```
# Load MNIST dataset
(train_X, train_y), (test_X, test_y) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

Convert matrix 3D to 2D

```
train_X = train_X.reshape(train_X.shape[0], -1)
test_X = test_X.reshape(test_X.shape[0], -1)
```

```
train_X.shape, test_X.shape

```

```
((60000, 784), (10000, 784))
```

```
test_X.shape , test_y.shape
```

```
((10000, 784), (10000,))
```

```
train_X[:5]
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

TASK 4 – MACHINE LEARNING

❑ Decision Tree model

DecisionTreeClassifier

```
dtree_Class = DecisionTreeClassifier(random_state=42)
dtree_Class.fit(train_X, train_y)

train_preds = dtree_Class.predict(train_X)
test_preds = dtree_Class.predict(test_X)

# Compute accuracy on training set
train_acc_treeClass = accuracy_score(train_y, train_preds)
print("Training accuracy:", train_acc_treeClass)

# Compute accuracy on test set
test_acc_treeClass = accuracy_score(test_y, test_preds)
print("Test accuracy:", test_acc_treeClass)
```

```
Training accuracy: 1.0
Test accuracy: 0.8755
```

DecisionTreeRegressor

```
tree_Reg = tree.DecisionTreeRegressor()
tree_Reg.fit(train_X, train_y)

train_preds = tree_Reg.predict(train_X)
test_preds = tree_Reg.predict(test_X)

# Compute accuracy on training set
train_acc_treeReg = accuracy_score(train_y, train_preds)
print("Training accuracy:", train_acc_treeReg)

# Compute accuracy on test set
test_acc_treeReg = accuracy_score(test_y, test_preds)
print("Test accuracy:", train_acc_treeReg)
```

```
Training accuracy: 1.0
Test accuracy: 1.0
```

Fit data samples and compute the accuracies in the training and test sets

TASK 4 – MACHINE LEARNING

❑ Decision Tree model

```
# Save model to file
with open("decision_tree.txt", "w") as f:
    f.write(export_text(dtree_Class, feature_names=["pixel"+ str(i) for i in range(784)]))
```

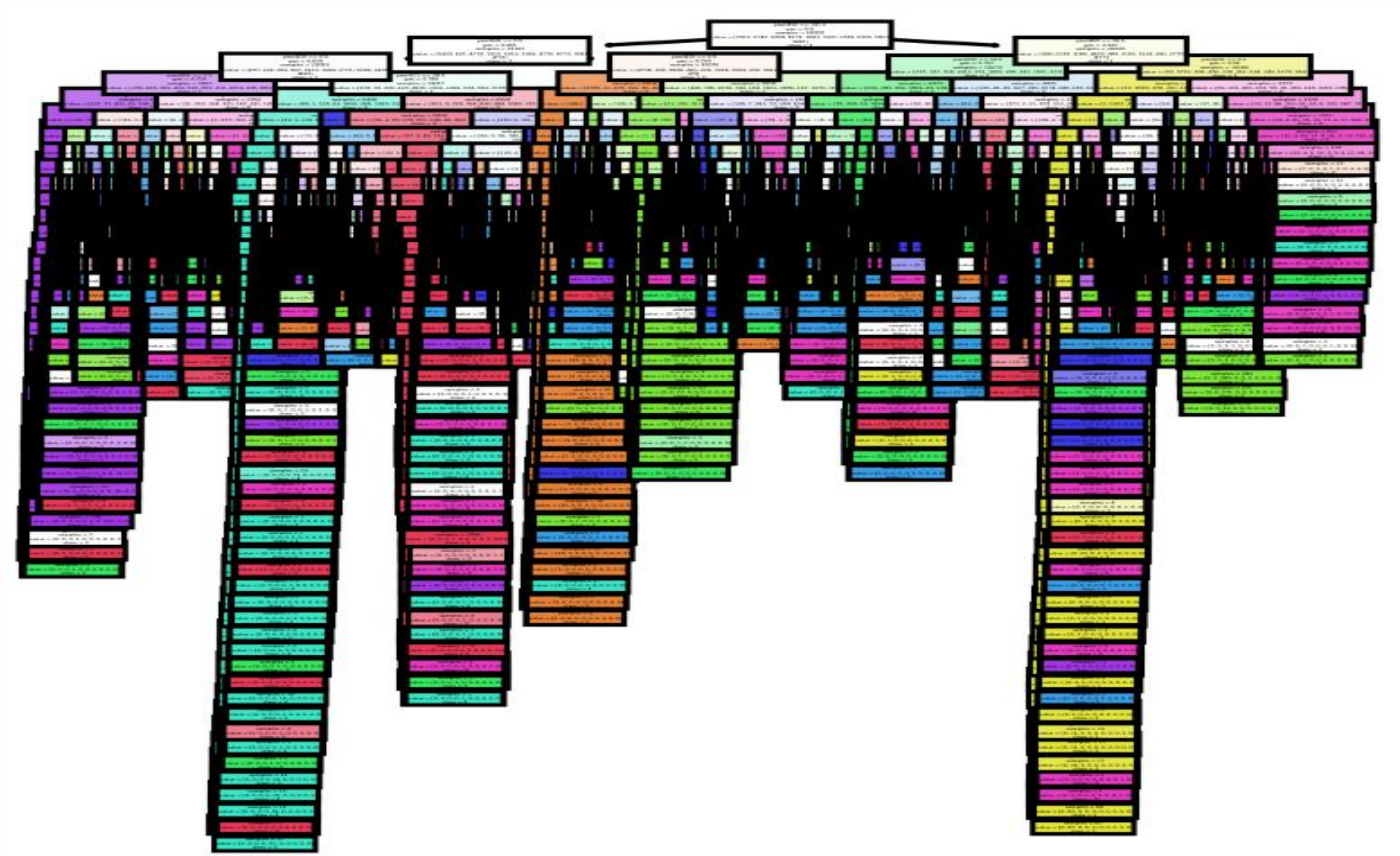
```
## Classifier
with open("decision_tree.txt", "r") as f:
    tree_data = f.read()
    dtree_loaded = DecisionTreeClassifier(random_state=42)
    dtree_loaded = dtree_loaded.fit(train_X.reshape((train_X.shape[0], -1)), train_y)
    dtree_loaded_str = export_text(dtree_loaded, feature_names=["pixel{}".format(i) for i in range(784)])
    dtree_loaded_str = dtree_loaded_str.replace('\n', '')

# Create new decision tree from loaded model
dtree_new = DecisionTreeClassifier(random_state=42)
dtree_new.fit(train_X.reshape((train_X.shape[0], -1)), train_y)

# Make predictions on new data
new_data = np.array([test_X[0], test_X[1], test_X[2], test_X[3], test_X[4]])
new_preds = dtree_new.predict(new_data.reshape((new_data.shape[0], -1)))
print("Predictions for new data:", new_preds)
```

```
Predictions for new data: [7 2 1 0 4]
```

Save, load the model from a file .txt and run inference

TASK 4 – MACHINE LEARNING

- Decision Tree model



Draw the tree structure using tree.plot_tree()

TASK 4 – MACHINE LEARNING

❑ Naïve Bayes classifier

GaussianNB

```
navi_GB = GaussianNB()
navi_GB.fit(train_X, train_y)

train_preds_NB = navi_GB.predict(train_X)
test_preds_NB = navi_GB.predict(test_X)

# Compute accuracy on training set
train_acc_Gau = accuracy_score(train_y, train_preds_NB)
print("Training accuracy:", train_acc_Gau)

# Compute accuracy on test set
test_acc_Gau = accuracy_score(test_y, test_preds_NB)
print("Test accuracy:", test_acc_Gau)
```

```
Training accuracy: 0.5649
Test accuracy: 0.5558
```

BernoulliNB

```
clf = BernoulliNB(force_alpha=True)
clf.fit(train_X, train_y)

train_preds_NB = clf.predict(train_X)
test_preds_NB = clf.predict(test_X)

# Compute accuracy on training set
train_acc_Ber = accuracy_score(train_y, train_preds_NB)
print("Training accuracy:", train_acc_Ber)

# Compute accuracy on test set
test_acc_Ber = accuracy_score(test_y, test_preds_NB)
print("Test accuracy:", test_acc_Ber)
```

```
Training accuracy: 0.83125
Test accuracy: 0.8413
```

Fit data samples and compute the accuracies in the training and test sets

TASK 4 – MACHINE LEARNING

❑ Naïve Bayes classifier

```python
## Just use GaussianNB to save and load file
# Save the pre-trained model to file
with open("naive_bayes_model.pkl", "wb") as f:
    pickle.dump(navi_GB, f)

# Load the model from a file
with open("naive_bayes_model.pkl", "rb") as f:
    nb_loaded = pickle.load(f)


# Run inference (prediction) for at least 5 input samples
samples = test_X[:5]
predictions = nb_loaded.predict(samples)
print("Predictions:", predictions)
```

```
Predictions: [9 2 1 0 9]
```

Save, load the model from a file .pkl and run inference

TASK 4 – MACHINE LEARNING

❑ K-NN model

```
classifier = KNeighborsClassifier(n_neighbors= 5)
classifier.fit(train_X, train_y)

```

```
▸ KNeighborsClassifier
```

```
train_preds_KNN = classifier.predict(train_X)
test_preds_KNN = classifier.predict(test_X)
```

```
# Compute accuracy on training set
train_acc_KNN = accuracy_score(train_y, train_preds_KNN)
print("Training accuracy:", train_acc_KNN)

# Compute accuracy on test set
test_acc_KNN = accuracy_score(test_y, test_preds_KNN)
print("Test accuracy:", test_acc_KNN)
```

```
Training accuracy: 0.9819166666666667
Test accuracy: 0.9688
```

Fit data samples and compute the accuracies in the training and test sets

TASK 4 – MACHINE LEARNING

❑ **K-NN model**

```
with open("knn_model.pkl", "wb") as f:
    pickle.dump(classifier, f)
```

```
with open("knn_model.pkl", "rb") as f:
    knn_loaded = pickle.load(f)
```

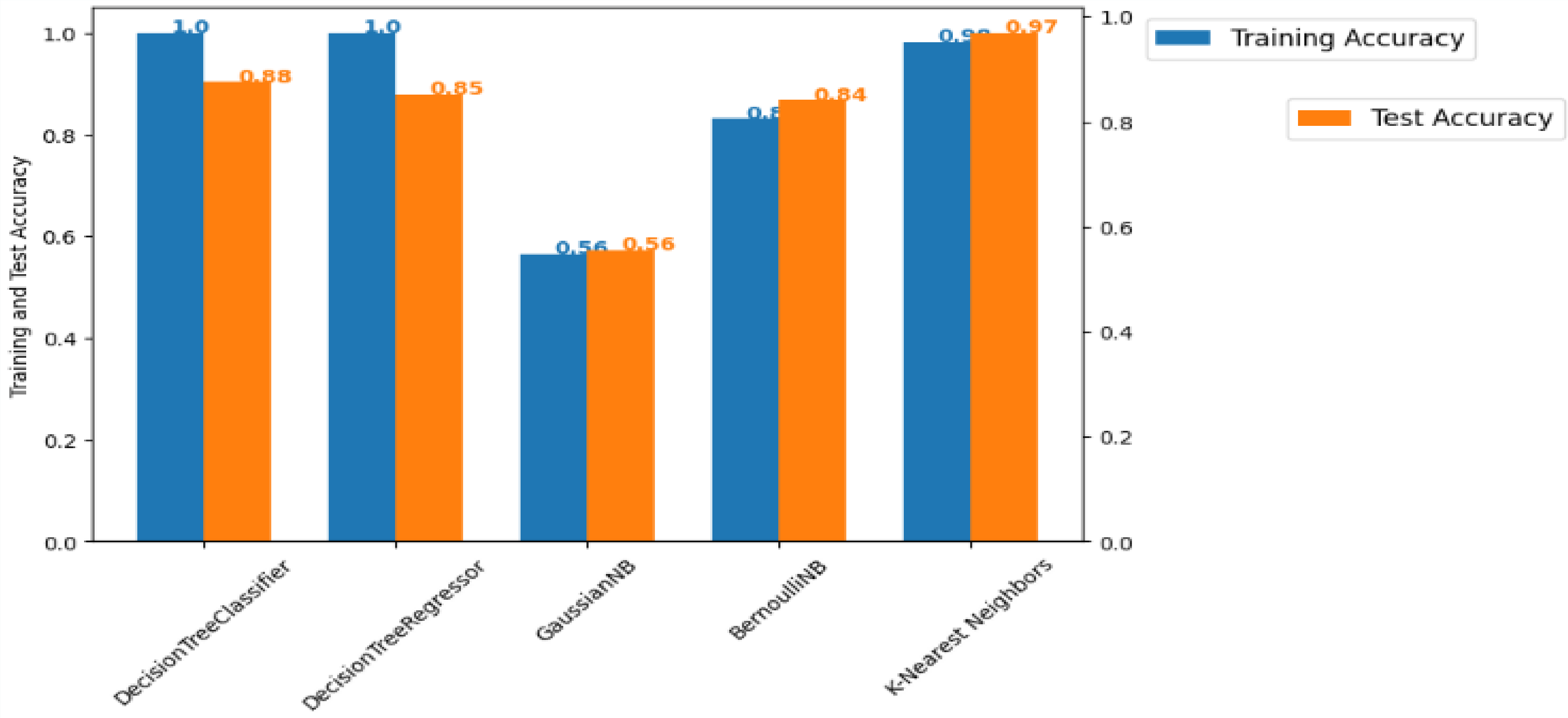
```
samples = test_X[5:10]
predictions = knn_loaded.predict(samples)
print("Predictions:", predictions)
```

```
Predictions: [1 4 9 5 9]
```

Save, load the model from a file .pkl and run inference

TASK 4 – MACHINE LEARNING

❑ **Demonstrate the accuracies in the training and test sets of the models**



ADVANTAGES:

- Basically understanding with python, hence ease code implementation.
- Some online documents are helpful to complete the requirements of the topic.
- The instructions in the lectures can be clearly understood.

DISADVANTAGES:

- The team can’t solve all the requirements because having multiple deadlines at the same time.
- There are some problems with understanding the algorithm.
- Rushing to make the deadline coincided with the long holiday, so the team can’t discuss directly.

ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

PROGRESS TABLE

TASK 1	REQUIREMENT 1	NGUYEN HOANG PHUC
	REQUIREMENT 2	
TASK 2	REQUIREMENT 1	TRAN QUOC BAO BUI HAI DUONG
	REQUIREMENT 2	
	REQUIREMENT 3	
TASK 3	REQUIREMENT 1	BUI ANH PHU
	REQUIREMENT 2	
TASK 4	REQUIREMENT 1	HOANG DINH QUY VU
	REQUIREMENT 2	
	REQUIREMENT 3	
	REQUIREMENT 4	
TASK 5	PRESENTATION	TRAN QUOC BAO BUI HAI DUONG
	SCRIPT	MEMBERS

Q&A

THANKS FOR WATCHING!