2 Name your Jupyter Notebook as:

`TASK2_<your name>_<centre number>_<index number>.ipynb`

This task is to compare the searching efficiency of Hash Table versus Binary Search on a sorted list.

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [ ]: #Task 2.1
        Program code

        Output:
```

Task 2.1
Write a function, `task2_1()` to:

- initialise a global 1-dimensional list
- generate 50 random integers between 1 and 1000 (inclusive) [Note: `import random` and use `random.randint()` to generate the random integers]
- store each integer in the list
- output the contents of the list. [2]

Test your program and show the output. [1]

Task 2.2
Implement a Hash Table with 10 buckets that uses chaining with Linked List for its collision resolution.

- The Hash Table, Linked List and Node are implemented using Object-Oriented Programming (OOP)
- Write program code for the 3 classes based on the specifications below
- Create the necessary Hash Table, Linked List and Node objects
- Insert all the values in the global list from Task2.1 into the Hash Table.
- Display the Hash Table. [17]

Sample Final Output for displaying hash table (your values will be different):

```
#Each bucket has its own LinkedList with 0 to many Nodes

index 0: [710, 660, 410, 670]
index 1: [241, 301, 61, 651]
index 2: [192, 372, 532, 22]
index 3: [363, 633, 253, 553]
index 4: [244, 414, 594, 964]
index 5: [465, 75, 295, 15, 795, 525, 725]
index 6: [296, 96, 136, 416, 336, 976, 356, 916, 206]
index 7: [587, 337, 507]
index 8: [28, 888, 288, 528, 378, 308, 348, 628, 208]
index 9: [249, 789]
```

Class: Node		
Identifier	**Data Type**	**Description**
`data`	`Integer`	• The Node's data
`next`	`Node Object`	• The next Node in the Linked List. • Default value is `None`.
`get_data()`	`Function`	• Return the value of the `data` attribute
`get_next()`	`Function`	• Return the next Node object
`set_data(value)`	`Procedure`	• Set the value of the `data` attribute with the given value
`set_next(nextNode)`	`Procedure`	• Set the value of the `next` attribute with the given Node object

Class: LinkedList		
Identifier	**Data Type**	**Description**
`head`	`Node Object`	• The first Node in the Linked List. • Default value is `None`.
`add_to_end(value)`	`Procedure`	• Create a new Node object with the given value • Add the new Node object to the end of the Linked List.
`search(target)`	`Function`	• Search for the target value in the Linked List. • Return `True` if found, otherwise return `False`.
`get_values()`	`Function`	• Return all the Nodes' data in the Linked List as a python list • Return `"Empty Linked List"` if Linked List is empty

Class: HashTable		
Identifier	**Data Type**	**Description**
`size`	`Integer`	• The size of the Hash Table • Set the size to **10**
`array`	`Python List of LinkedList Objects`	• Python List containing **10** Linked List objects
`hash(value)`	`Function`	• Map the given value to the array index using the formula `value%size` • Return the array index

`insert(value)`	`Procedure`	• Insert given value in the correct Linked List object in the `array` attribute
`search(target)`	`Function`	• Search for the target value in the correct Linked List object in the `array` attribute • Return `True` if found, otherwise return `False`.
`display()`	`Procedure`	• Display all the values in the Hash Table (follow the Sample Final Output shown above)

Task 2.3

Write the procedure `task2_3()` to search for every integer present in the global list from Task 2.1 in the Hash Table created in Task 2.2. [2]

Task 2.4

Another method to search for specific values in the global list is to first sort the list using merge sort followed by performing binary search.

- Write a separate function to perform merge sort on the given list and return the sorted list.
- Write a separate function to perform binary search on the given sorted list. The function is to return `True` if the target value is found, otherwise return `False`.
- Write the procedure `task2_4()` to execute merge sort on the global list from Task 2.1 followed by searching for every integer present in the global list using the binary search function. [12]

Task 2.5

The `timeit` library is built into Python and can be used to time simple function and procedure calls. Example code is shown in `Task2_timing.py`.

Using the `timeit` module, display the time taken to execute the procedures `task2_3()` and `task2_4()`. [1]

Save your Jupyter Notebook for Task 2.