H2 Computing Practical Worksheet – T2W5

1a The text file DATA.TXT contains the results of 100 students. For each student, the student's index number and the scores for 10 subjects are listed. More specifically, each line of the text file contains the data for 1 student in the following format:
<index number>,<score 1>, …, <score 10>

Write a program that will read the data from the file and then output the following:

- The top 3 and bottom 3 students for each subject. Your output should be formatted as follows (note that the example values shown are not accurate).

```
Subject 1

Top 3 Students:
Student 11          95
Student 4           93
Student 78          90

Bottom 3 Students:
Student 73          25
Student 44          29
Student 6           34
```

  Note that subject 1 refers to the subject in the first score column.

- The top 3 and bottom 3 students based on overall score (i.e., average score). Use the same format as shown above

- Note that for both of the above functionality, you are to utilise the quicksort algorithm. There should only be 1 implementation of this algorithm which is called to sort for each subject and to sort the average.

1b You are also to compute the grades for all scores using the following table:

Score	**Grade**
Above 70	A
60-69	B
55-59	C
50-54	D
45-49	E
40-44	S
Below 40	U

- Once computed, save all the grades (maintaining the original order as defined in DATA.TXT) into a new text file called GRADES.TXT

- Compute the grade frequency for each subject and then for each subject, output the following.

```
Subject 1

A      10 Students
B      15 Students
C      20 Students
D      25 Students
E      15 Students
S      10 Students
U      5 Students
```

- Finally, output the subject with the highest percentage of As and Bs.

```
Subject 7 has the most As and Bs.
```

2 Implement the following Hybrid data structure using object-oriented programming.

```
BST
-root: BSTNode

+initialisation()
+get_root(): BSTNode
+set_root(BSTNode)
+insert(OBJECT)
+exists(OBJECT): BOOLEAN
+delete(OBJECT) : BOOLEAN
+inorder(): STRING
+string_convert(): STRING
```

```
Hybrid
-unsorted: LL
-sorted: BST
-rate_sorted: FLOAT

+initialisation(FLOAT)
+insert(OBJECT)
+exists(OBJECT): BOOLEAN
+delete(OBJECT) : BOOLEAN
+string_convert(): STRING
```

```
LL
-first: LLNode

+initialisation()
+get_first(): LLNode
+set_first(LLNode)
+insert(OBJECT)
+exists(OBJECT): BOOLEAN
+delete(OBJECT) : BOOLEAN
+string_convert(): STRING
```

```
BSTNode
-data: OBJECT
-left: BSTNode
-right: BSTNode
-tombstone: BOOLEAN

+initialisation()
+get_data(): OBJECT
+get_left(): BSTNode
+get_right(): BSTNode
+set_data(OBJECT)
+set_left(BSTNode)
+set_right(BSTNode)
+inorder(): STRING
+string_convert(): STRING
```

```
LLNode
-data: OBJECT
-prev: LLNode
-next: LLNode

+initialisation()
+get_data(): OBJECT
+get_prev(): LLNode
+get_next(): LLNode
+set_data(OBJECT)
+set_prev(LLNode)
+set_next(LLNode)
+inorder(): STRING
+string_convert(): STRING
```

The Hybrid Class:

- Takes in 1 parameter on initialisation, `rate_sorted`, which is used on insertion to determine the chance that the BST is used instead of the LL. This should be a FLOAT between 0 and 1. This will default to 1. You must perform exception handling on input for this parameter.
- Initialisation will result in the creation of both the BST and LL, regardless of the value of `rate_sorted`.
- The insert function requires the specification of the object to be stored, and utilised `rate_sorted` to determine if the object is to be stored in the BST or LL – i.e., you are to use random and if the resultant random value is greater than `rate_sorted`, insert the object into the LL, or else, insert it into the BST.
- The exists and delete functions both require a target object as a parameter, and will return True if the object is found or deleted respectively, or else will return False. Note that when called, these functions should first search the BST before searching the LL.
- The `string_convert` method refers to the inbuilt string conversion function of all objects. Change it such that it will print:

```
Hybrid Structure Contents

BST: [2, 3, 4, 5, 6, 7, 8]

LL: [10, 15, 1, 9, 18, 24]
```

  Note that the contents of the BST are printed using inorder traversal, while the LL contents are printed based on the order in which they are stored (i.e., first to last).

The BST Class:

- Typical BST implementations, except that tombstoning is utilised for deletion. You are to ensure that tombstoned values do not appear in the inorder and string_convert output.
- Note that string_convert is to utilise the inorder method; implement the latter using recursion.

LL Class:

- Typical LL implementation with insertion occurring in the front of the list.