# Assessed Exercise 1 – Part A

**This is the first part of a two-part assessed exercise. This part is worth 40 marks out of a total 100 marks. Both parts of the assessed exercise together account for 15% of the course total.**

**Please note that:**

1. **While you are free to *refer* to sample solutions provided in the relevant problem sets, you must not use them. That is, you must write your own code, *from scratch*.**
2. **In general, you should limit your code to using the barebones Python, without using any libraries (other than the obvious ones like time, random, sys). If you have any doubts about use of a particular library, then please ask the instructor.**

**Problem 1 [10 marks]**
Implement QUICKSORT in Python, based on the pseudocode for QUICKSORT introduced in the class, including procedure PARTITION implementing right-most element pivot selection.

**Problem 2 [10 marks]**
Using a function called `TimeSortingAlgorithms` that compares the execution times of different sorting algorithms, compare QUICKSORT from Part 1 with the running times of `BubbleSort`, `InsertionSort` and `MergeSort` (implemented in earlier Problem sets).Use datasets provided.

*See APPENDIX A for an example of what a successful run of Problem 1+2 should look like. The output you see in that Appendix you should be printed by your python program.*

**Problem 3 [10 marks]**

The *Dynamic Set* is an abstract data type (ADT) that can store distinct elements, without any particular order. As opposed to *static* or *frozen* sets, dynamic sets allow insertion and deletion of elements. That is, they are *mutable*. There are five main operations in the ADT:

- add(S,x): add element x to S, if it is not present already
- remove(S,x): remove element x from S, if it is present
- is_element(S,x): check whether element x is in set S
- set_empty(S): check whether set S has no elements
- set_size(S): return the number of elements of set S

Implement in Python[1] the Dynamic Set ADT defined above using

a) A doubly linked list. [5]
b) A static array implementation (size of array may be picked arbitrarily) [5]

**Problem 4 [10 marks]**

a) Compare the two implementations of the Dynamic Set ADT by carrying out the following empirical study. First, populate (an initially empty) set S with all the elements from dataset `Int20k.txt` provided on Moodle. Then, generate 100 random numbers in the interval [0, 49999]. For each random number x record the time taken to execute is_element(S,x), and store the results in a file called "results_problem4.txt", with one result per line (time only, in milliseconds, don't mention units), and a line each to indicate the start of linked-list results and then array results. So your text file should look something like this:

```
Results for Linked-list based implementation
<200 lines of results>
Results for Array based implementation
<200 lines of results>
```

Finally, calculate and print the average running time of is_element over 100 calls for the two implementations of the ADT. All of the above should be done as part of your Python code.

[6]

b) Comment on and explain your findings in a block comment at the top of your python code.

[4]

## Submission Instructions

**Submit your coursework on Moodle (check Moodle for deadline) in the form of the following files:**

1. `solution_problem_1_2.py`
2. `solution_problem_3_4.py`
3. `results_problem4.txt`

**The Python files should be written to meet API specification as required by the tester script provided for this part:** `tester.py`.

[1] You will need to engage with Object-Oriented design concepts to complete this part. Following is a good reference:
http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_I.html
http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_II.html

# APPENDIX A

```
================================================================
This is a test for USER sort Function :: QUICKSORT
================================================================
Creating random list of 10 integers in the range 1 - 1000

Unsorted list is :
 [515, 722, 454, 519, 527, 58, 514, 672, 73, 625]

Testing USER sort function...
TEST PASSED
Sorted list is :
 [58, 73, 454, 514, 515, 519, 527, 625, 672, 722]

================================================================
This is a test for USER sort Function :: BUBBLESORT
================================================================
Creating random list of 10 integers in the range 1 - 1000

Unsorted list is :
 [916, 187, 941, 165, 898, 959, 692, 539, 794, 541]

Testing USER sort function...
TEST PASSED
Sorted list is :
 [165, 187, 539, 541, 692, 794, 898, 916, 941, 959]


================================================================
This is a test for USER sort Function :: INSERTIONSORT
================================================================
Creating random list of 10 integers in the range 1 - 1000

Unsorted list is :
 [916, 187, 941, 165, 898, 959, 692, 539, 794, 541]

Testing USER sort function...
TEST PASSED
Sorted list is :
 [165, 187, 539, 541, 692, 794, 898, 916, 941, 959]

================================================================
This is a test for USER sort Function :: MERGESORT
================================================================
Creating random list of 10 integers in the range 1 - 1000

Unsorted list is :
 [925, 24, 503, 693, 505, 278, 841, 272, 189, 986]

Testing USER sort function...
TEST PASSED
Sorted list is :
 [24, 189, 272, 278, 503, 505, 693, 841, 925, 986]

================================================================
Timing 5 Sorting Algorithms on variety of input sizes
================================================================

Time taken to sort int10.txt:
QUICKSORT      : 0.014499993994832039 milliseconds
BUBBLESORT     : XXXX milliseconds
INSERTIONSORT  : XXXX milliseconds
MERGESORT      : XXXX milliseconds


Time taken to sort int50.txt:
QUICKSORT      : XXXX milliseconds
BUBBLESORT     : XXXX milliseconds
INSERTIONSORT  : XXXX milliseconds
MERGESORT      : XXXX milliseconds


Time taken to sort int100.txt:
QUICKSORT      : XXXX milliseconds
BUBBLESORT     : XXXX milliseconds
INSERTIONSORT  : XXXX milliseconds
MERGESORT      : XXXX milliseconds


Time taken to sort int1000.txt:
QUICKSORT      : XXXX milliseconds
BUBBLESORT     : XXXX milliseconds
INSERTIONSORT  : XXXX milliseconds
MERGESORT      : XXXX milliseconds


Time taken to sort int1000_presorted_ascending.txt:
QUICKSORT      : XXXX milliseconds
BUBBLESORT     : XXXX milliseconds
INSERTIONSORT  : XXXX milliseconds
MERGESORT      : XXXX milliseconds


Time taken to sort int1000_presorted_descending.txt:
QUICKSORT      : XXXX milliseconds
QUICKSORT_MED3 : XXXX milliseconds
BUBBLESORT     : XXXX milliseconds
INSERTIONSORT  : XXXX milliseconds
MERGESORT      : XXXX milliseconds
```