PLEASE NOTE: All functions must have the signatures provided, using different functions will result in 0 marks for that implementation. Please include your name at the top of each .cpp and upload to Moodle. ***Note: For each part you should write only one source program, which includes all the functionality you have been asked for to test your implementation.***

---

10 marks for good coding practice, e.g., clear well-presented code, well-chosen variable and class names and appropriate comments – if you do not put your name as a comment on the file you may lose these.

Marks may be deducted for bad programming practice [-80 Marks]
For example, including .cpp, files, global variables, etc, etc.

Part A (19 Marks)

1. Develop two template functions, firstly **a template function for the recursive version of the QuickSort algorithm.** This quickSort function should use a second **template function called partition.** The partition function should use the **STL** function `swap()` to switch two elements in the array (see swap example below).

2. Your functions **must** have the following declarations.

```
template<typename T>
void quickSort(T[], int low, int high);
```
[7 marks]

```
template<typename T>
int partition(T[], int low, int high);
```
[7 marks]

3. Code a main program that tests your quicksort function by sorting an array of integers and then doubles. **[1 mark]**

4. Add as a comment two ways to improve the QuickSort algorithm. **[4 marks]**

Part B (16 Marks)

You are given two sorted integer arrays, A and B, and A has a large enough buffer at the end to hold B. Write a method in pseudocode to merge B into A in sorted order. For Example:

```
Input: a[] = {10, 12, 13, 14, 18, empty, empty, empty, empty,
empty }  ;
       b[] = {16, 17, 19, 20, 22};
Output: a[] = {10, 12, 13, 14, 16, 17, 18, 19, 20, 22};
```
[16 marks]

Part C (14 Marks)

Using a single source.cpp file create a "shape" hierarchy: a base class called `Shape` and derived classes called `Circle, Square`, and `Triangle`. In the base class, make a virtual function called `draw()`, which outputs "shape", and override this in each of the derived classes to output the name of the shape. In a `main()` make an array of pointers to Shape objects and call `draw()` through the base-class pointers on each object in this array. The correct name for each Shape object should be displayed. **[14 marks]**

Part D (41 Marks)

1. Develop a Binary Search **Tree Node class** that stores integers. [1 mark]. The Node only has the following node constructor method:

- `TreeNode(int data);` [2 marks]

[3 marks for part D1]

2. Implement a **Binary Search Tree Class** that stores Tree Nodes. The Tree will have the following:

- an overloaded **s**`tream insertion operator <<` as a stand-alone function that outputs all the nodes in the Tree in a preorder sequence. [4 marks]

public methods:

- `BinaryTree()` A tree constructor and creates an empty tree [1 mark]
- `~BinaryTree()` A tree destructor that frees up memory [4 marks]
- `TreeNode* deSerialise(ifstream& InFile)` reads a BST from a text file. [12 marks]
- `void add(int)` calls private method below [1 mark]
- `int Serialise(ofstream& OutFile)` calls private method below. [1 mark]

Private methods

i`nt serialise(ofstream& fp, TreeNode* root)` save the BST to a text file, returns -1 if it fails. [9 marks]

`void add(TreeNode* toAdd, TreeNode* attachHere)` Note: This should use recursion. Duplicates are not allowed. [4 marks]

[36 marks in total for part D2]

3. Write a **Main Program** that tests all the above methods of your binary search tree.

[2 marks for part D3]