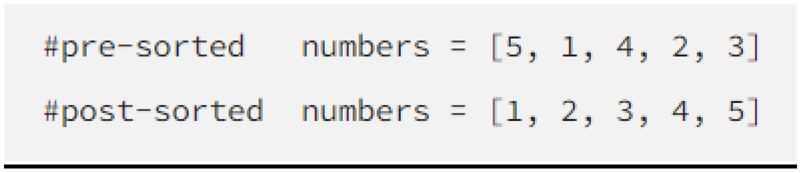
Lab Task 04 : Sorting and Searching with Unit Testing

What is the purpose of sorting?
What exactly is the purpose of a sorting algorithm? It is the job of a sorting algorithm to take an unordered list of numbers and, well, order them.



```
#pre-sorted   numbers = [5, 1, 4, 2, 3]

#post-sorted  numbers = [1, 2, 3, 4, 5]
```

Sorting an unordered list of numbers greatly increases the ***power*** you have over those numbers. Sorted lists allow you to better pick apart the information hiding inside the numbers

There are a number of sorting algorithms. However we will only cover three sorting algorithms.

1. Insertion Sort
2. Bubble Sort
3. Quick Sort (Iterative)

Insertion Sort

In this, the array is sorted into two, the first one being sorted part at the left side & other one being unsorted part at the right side. We iterate over the array and for every iteration we pick the first element from unsorted array and insert it at the correct position in the sorted part. This process is continuously repeated until the whole array is sorted.

Time Complexity : $O(n^2)$ in worst case & $O(n)$ in Best case

Space Complexity : $O(1)$

Bubble Sort

In this, we start from the begnining of the array, swap the first two elements, if the first one is bigger than the second one. This process is continuously repeted until we reach the end of the array. In this way, the array elements are put in a sorted order as smaller elements bubble upto the beginning of the array.

Time Complexity : O(n^2)
Space Complexity : O(1)

Quick Sort (Iterative Version)

In this, we pick a random element & partition the array, such that all numbers that are less than the partitioning element come before it & all elements greater than it come after the partioning element. Repeatedly partitioning the array will eventually result into the sorted array. The time complexity depends on the position of the partitioned element which makes the sorting slower.

Time Complexity : O(n log(n)) in best or average & (n^2) in worst case; **Space Complexity** : O(log(n))

Note: Quick sort can also be done through recursion. However we will do it iteratively in this lab.

LAB TASK

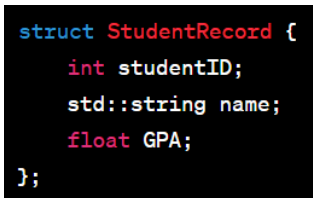
Question 1:

Imagine you are tasked with sorting a list of student records in C++. Each student record consists of the following information:

Student ID (an integer)
Name (a string)
GPA (a float)

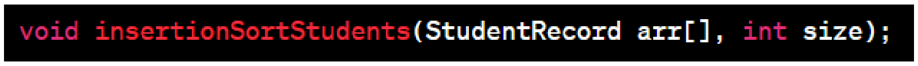
You are required to use the insertion sort algorithm to sort these student records based on their GPAs in non-decreasing order. Write a C++ function that takes an array of student records and its size as input and sorts the records using insertion sort.

Write a C++ function insertionSortStudents that sorts an array of student records based on their GPAs using the insertion sort algorithm. The student records should be sorted in non-decreasing order of GPA. Each student record is represented as a struct with the following definition:



```
struct StudentRecord {
    int studentID;
    std::string name;
    float GPA;
};
```

The function should have the following signature:



```
void insertionSortStudents(StudentRecord arr[], int size);
```

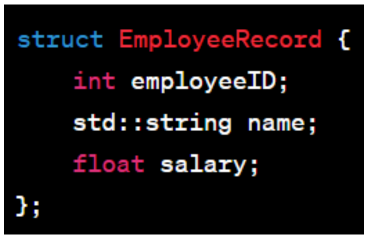
Provide the implementation of the insertionSortStudents function and any additional helper functions you may need.

<u>Question 2:</u>
You are tasked with sorting a list of employee records in C++ using the iterative quicksort algorithm. Each employee record consists of the following information:

Employee ID (an integer)
Name (a string)
Salary (a float)

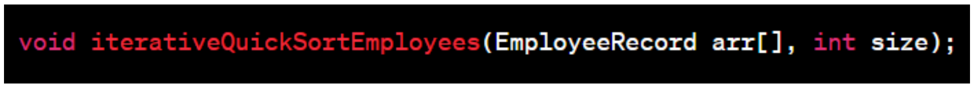
Your goal is to implement an iterative version of the quicksort algorithm to sort these employee records based on their salaries in non-decreasing order. The sorting should be done in-place without using recursion.

Write a C++ function iterativeQuickSortEmployees that sorts an array of employee records based on their salaries using the iterative quicksort algorithm. The employee records should be sorted in non-decreasing order of salary. Each employee record is represented as a struct with the following definition:



```cpp
struct EmployeeRecord {
    int employeeID;
    std::string name;
    float salary;
};
```

The function should have the following signature:



```cpp
void iterativeQuickSortEmployees(EmployeeRecord arr[], int size);
```

Provide the implementation of the iterativeQuickSortEmployees function and any additional helper functions you may need.

Question 3:

Imagine you are managing a database of student grades, and you need to perform two key operations: sorting the grades using the bubble sort algorithm and searching for a specific student's grade using binary search. Each student is uniquely identified by their student ID, and their corresponding grade is represented as an integer.

Write a C++ program that first performs the following tasks:

Implement a function bubbleSort to sort an array of student grades in non-decreasing order using the bubble sort algorithm. The function should have the following signature:

```cpp
void bubbleSort(int grades[], int numStudents);
```

Next, implement a function binarySearch to search for a specific student's grade based on their student ID. The function should perform a binary search on the sorted array of grades and return the grade if the student is found or -1 if the student ID is not in the database. The function should have the following signature:

```cpp
int binarySearch(int grades[], int numStudents, int studentID);
```