Lab 5
Quick Sort

1. [5 points] **]**

2. [5 points] You're exploring a cave filled with chests of various sizes, each containing valuable treasures represented by array elements. To efficiently organize your findings, you decide to sort the treasures using a unique method. Instead of always choosing the biggest or smallest chest as a reference, you opt for **randomness** to keep your quest unpredictable. Develop a function for quicksort, where you **randomly select** a chest (pivot) to partition the array. Display the array partition and the corresponding pivot selected in each iteration. Finally display the sorted array in descending order.

   Example:
   Input: [9, 8, 7, 6, 5, 4, 3, 2, 1]
   Expected Output: [9,8,7,6,5,4,3,2,1]

Practice questions

Question 1: Quick Sort with Median as Pivot
Implement the Quick Sort algorithm in C++ using the median of three elements (first, middle, and last) as the pivot. Write a function `void quickSortMedianPivot(int arr[], int low, int high)` that sorts
the array `arr` in ascending order. The function should take the array, the lower index `low`, and the higher index `high` as parameters. Use the median of three elements as the pivot for partitioning.

Merge Sort
Question 1: Implement the Merge Sort algorithm in C++. Write a function `void mergeSort(int arr[], int low, int high)` that sorts the array `arr` in ascending order. The function should take the array, the lower index `low`, and the higher index `high` as parameters.

Question 2: Merge Sort with Iterative Approach
Implement the Merge Sort algorithm in C++ using an iterative approach. Write a function `void mergeSortIterative(int arr[], int size)` that sorts the array `arr` in ascending order. The function should take the array and its size as parameters. Use an iterative approach instead of the traditional recursive implementation.

Question 3: Merge Sort with Adaptive Optimization
Implement the Merge Sort algorithm in C++ with an adaptive optimization that improves

performance for partially sorted arrays. Write a function `void mergeSortAdaptive(int arr[], int low,
int high)` that sorts the array `arr` in ascending order. The function should take the array, the lower
index `low`, and the higher index `high` as parameters. Implement a mechanism to detect partially
sorted arrays and optimize the merge process accordingly.