Objective(s) : To practice and gain skills in implementing sorting algorithms.

task 1: MyMergeSort.java

Implement MyLinkedList.Java, inside package \Lab08\pack

```
public class MyMergeSort {
  public void mSort(int[] arr) {
    mSort(arr,0,arr.length-1);
  }
  private void mSort(int[] arr, int low, int high) {
    if (low >= high)    return;
    int mid = low + ((high - low) >> 1);
    mSort(arr,low,mid);
    mSort(arr,mid+1,high);
    merge(arr,low,mid,high);
  }
  private void merge(int[] arr, int low,
                     int mid, int high) {
    int[] tmp = new int[high - low + 1];
    /* your code */

  }
}
static void task_1() {
  int[] arr = {42,17,-5,88,23,91,-12,65,7,30,55,-9,2,48,76,1,-22,99,14,61,
                                    37,83,-18,50,29,72,6,40,11,68}; // 30 elements
  MyMergeSort sol = new MyMergeSort();
  sol.mSort(arr);
  Syste.out.println(Arrays.toString(arr));
}
```

Note that You can save space by creating a temporary array int[] tmp = new int[arr.length] for reuse during merging (with a little adjustment to the merging indices).

task 2: Implement MyQuickSort.Java, inside package \Lab08\pack using lomuto's partition algorithm.

```
public class MyQuickSort_Sol {
  public void qSort(int[] arr) {
    qSort_lumoto(arr, 0, arr.length-1);
  }
  private void qSort_lumoto(int[] arr, int low, int high) {
    if (low < high) {
        int pivot_index = partition_lomuto(arr, low, high);
        qSort_lumoto(arr, low, pivot_index - 1);
        qSort_lumoto(arr, pivot_index + 1, high);
    }
  }
  private int partition_lomuto(int[] arr, int low, int high) {
        int pivot_v = arr[high];
        int i = low, tmp;
        /* your code */

        return i;
    }
}
static void task_2() {
    int[] arr = {42,17,-5,88,23,91,-12,65,7,30,55,-9,2,48,76,1,-22,99,14,61,
                                    37,83,-18,50,29,72,6,40,11,68}; // 30 elements
    MyQuickSort_Sol sol = new MyQuickSort_Sol();
    sol.qSort(arr);
    System.out.println(Arrays.toString(arr));
}
```

We use Lomuto's scheme because it is easier to modify for related problems.

task 3: Dutch National Flag Algorithm

If the array contains only three distinct values (for example: min, mid, and max), you can use the Dutch National Flag algorithm to place each element in the correct region:

- If the current element equals min, swap it with the element at the left index.
- If it equals max, swap it with the element at the right index.
- Once the index j passes right, all min values will be on the left, all max values will be on the right, and all mid values will be in the middle.

Complete the static void dutch_national_flag(int[] arr)

```
static void dutch_national_flag(int[] arr) {
    int left_value = arr[0];    // min
    int right_value = arr[0];   // max
    for (int i = 1; i < arr.length; i++) {
        left_value = Math.min(left_value, arr[i]);
        right_value = Math.max(right_value, arr[i]);
    }
    /* your code */
}
static void task_3() {
               // R  B  W  W  B  B  R  W  W  R  R  W  R  B W
    int[] arr = {-1, 0,-2,-2, 0, 0,-1,-2,-2,-1,-1,-2,-1, 0,-2};
    dutch_national_flag(arr);
    System.out.println(Arrays.toString(arr));
}
```

QuickSort can degrade to $O(n^2)$ when the data contains many duplicate pivot values. Integrating three-way partitioning (where mid is treated as the pivot and everything smaller goes to the left, everything larger goes to the right) can help avoid this pitfall.

task 4: When the data range is not too large, Counting Sort can be applied. Its runtime is O(n). We can use this method to solve the k-th smallest element problem.

Complete the int k_th_min_element(int[] arr, int k)

```
    static int k_th_min_element(int[] arr, int k) {
        int min = arr[0];
        int max = arr[0];
        for (int i = 1; i < arr.length; i++) {
            min = Math.min(min, arr[i]);
            max = Math.max(max, arr[i]);
        }
        int[] count = new int[max - min + 1];
        /* your code */
        return -1; // exception
    }

}
static void task_4() {
    int[] arr = {-1,0,-2,-2,0,0,-1,-2,-2,-1,-1,-2,-1,0,-2};
    System.out.println(k_th_min_element(arr,2));
}
```

submission: MyMergeSort_XXYYYY.Java, MyQuickSort_XXYYYY.java and Lab08_XXYYYY.java.

Due Date: TBA