Code:

Merge sort

prac2Merge.java > prac2Merge > merge(int[], int, int, int)

```java
import java.util.Arrays;

public class prac2Merge {
    Run | Debug
    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        System.out.println("Original array: " + Arrays.toString(arr));

        long startTime = System.nanoTime();
        mergeSort(arr, left:0, arr.length - 1);
        long endTime = System.nanoTime();
        long duration = (endTime - startTime);

        System.out.println("Sorted array: " + Arrays.toString(arr));
        System.out.println("Time taken for sorting (in nanoseconds): " + duration);
    }

    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int middle = (left + right) / 2;

            mergeSort(arr, left, middle);
            mergeSort(arr, middle + 1, right);

            merge(arr, left, middle, right);
        }
    }

    public static void merge(int[] arr, int left, int middle, int right) {
        int n1 = middle - left + 1;
        int n2 = right - middle;

        int[] leftArr = new int[n1];
        int[] rightArr = new int[n2];

        for (int i = 0; i < n1; i++) {
            leftArr[i] = arr[left + i];
        }
        for (int i = 0; i < n2; i++) {
            rightArr[i] = arr[middle + 1 + i];
        }

        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (leftArr[i] <= rightArr[j]) {
                arr[k] = leftArr[i];
                i++;
            } else {
                arr[k] = rightArr[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = leftArr[i];
            i++;
            k++;
        }
        while (j < n2) {
            arr[k] = rightArr[j];
            j++;
            k++;
        }
    }
}
```

Output:

```
Original array: [12, 11, 13, 5, 6, 7]
Sorted array: [5, 6, 7, 11, 12, 13]
Time taken for sorting (in nanoseconds): 7500
```

Quick Sort:

prac2Quick.java > prac2Quick > partition(int[], int, int)

```java
import java.util.Arrays;

public class prac2Quick {
    Run | Debug
    public static void main(String[] args) {
        int[] arr = {12, 11, 13, 5, 6, 7};
        System.out.println("Original array:-"+Arrays.toString(arr));

        long startTime = System.nanoTime();
        quickSort(arr, low:0, arr.length - 1);
        long endTime = System.nanoTime();

        System.out.println("Sorted array: " + Arrays.toString(arr));

        // Calculate the time taken in milliseconds
        long duration = (endTime - startTime);
        System.out.println("Time taken to sort: " + duration + " nanoseconds");
    }
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);

            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }
    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = (low - 1);

        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(arr, i, j);
            }
        }
        swap(arr, i + 1, high);
        return i + 1;
    }
    public static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

Output:

```
Original array:-[12, 11, 13, 5, 6, 7]
Sorted array: [5, 6, 7, 11, 12, 13]
Time taken to sort: 6100 nanoseconds
```