Μηχανικών Πληροφορικής &
Ηλεκτρονικών Συστημάτων

ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

Αντικειμενοστρεφής Προγραμματισμός

MergeSort

QuickSort

Ασδρέ Κατερίνα
asdre@ihu.gr

MergeSort

1ος τρόπος:

```
public class MergeSort {
    public static void sort (int[] A) {
        mSort (A, 0, A.length-1);
    }

    public static void mSort (int[] A, int f, int l) {
       if (f==l) return;
       int mid=(f+l)/2;      // Μεσαία θέση
       mSort (A, f, mid);    // Αναδρομική κλήση, 1ο μισό
       mSort (A, mid+1, l); // Αναδρομική κλήση, 2ο μισό
       merge(A,f,l,mid);     // Συγχώνευση
    }
```

1ος τρόπος:

```
public static void merge(int []A, int f, int l, int mid) {
    // Create L ← A[f..mid] and M ← A[mid+1..l]
    int n1 = mid - f + 1;
    int n2 = l - mid;
    int[] L = new int[n1];
    int[] M = new int[n2];
    for (int i = 0; i < n1; i++)    L[i] = A[f + i];
    for (int j = 0; j < n2; j++)    M[j] = A[mid + 1 + j];
    int i=0, j=0, k=f;
    while (i < n1 && j < n2) {
        if (L[i] <= M[j]) A[k++] = L[i++];
        else  A[k++] = M[j++];
    }
    while (i < n1)  A[k++] = L[i++];
    while (j < n2)  A[k++] = M[j++];
}}
```

2ος τρόπος:

```
import java.util.Arrays;
public class MergeSort {
  public static void mergeSort(int[] A) {
   if (A.length > 1) {
     int[] firstHalf = new int[A.length/2];
     System.arraycopy(A, 0, firstHalf, 0, A.length/2);
     mergeSort(firstHalf);
     int secondHalfLength = A.length -A.length/2;
     int[] secondHalf = new int[secondHalfLength];
     System.arraycopy(A,A.length/2,secondHalf,0,secondHalfLength);
     mergeSort(secondHalf);
     merge(firstHalf, secondHalf, A);
  }
 }
```

2ος τρόπος:

```
public static void merge(int[] list1, int[] list2, int[] temp) {
    int current1 = 0;
    int current2 = 0;
    int current3 = 0;
    while (current1 < list1.length && current2 < list2.length) {
        if (list1[current1] < list2[current2])
            temp[current3++] = list1[current1++];
        else
            temp[current3++] = list2[current2++];
    }
    while (current1 < list1.length)
        temp[current3++] = list1[current1++];
    while (current2 < list2.length)
        temp[current3++] = list2[current2++];
  }
}
```

QuickSort

```
public class QuickSort {
  public static void quickSort(int[] list) {
       QSort(list, 0, list.length -1);
   }
  public static void QSort(int[] list, int first, int last) {
    if (last > first) {
        int pivotIndex = partition(list, first, last);
        QSort(list, first, pivotIndex -1);
        QSort(list, pivotIndex + 1, last);
    }
  }
```

```
public static int partition(int[] list, int first, int last) {
    int pivot = list[first];
    int low = first + 1;
    int high = last;
    while (high > low) {
      while (low <= high && list[low] <= pivot) low++;
      while (low <= high && list[high] > pivot) high--;
      if (high > low) {
          int temp = list[high];
          list[high] = list[low];
          list[low] = temp;
      }
    }
```

```
    if (pivot > list[high]) {
        list[first] = list[high];
        list[high] = pivot;
        return high;
    }
    else {
        return first;
    }
}
}
```