5 MOST COMMON SORTING ALGORITHMS

BUBBLE

SELECTION

INSERTION

MERGE

QUICK

BUBBLE SORT

TO SORT AN ARRAY *A*[0..*N*-1]:

```
FOR (INT LAST = N -1; LAST >= 1; LAST --)
{
  // MOVE THE LARGEST ENTRY IN A[0…LAST] TO A[LAST]
  FOR (INT INDEX = 0; INDEX <= LAST-1; INDEX++)
  {
      //SWAP ADJACENT ELEMENTS IF NECESSARY
       IF (A[INDEX] > A[INDEX+1])
       {
          INT TEMP = A[INDEX];
          A[INDEX] = A[INDEX+1];
          A[INDEX + 1] = TEMP;
       }
  }
}
```

SELECTION SORT

```
FOR (LAST = N -1; LAST >= 1; LAST --)
{
    //MOVE THE LARGEST ENTRY IN A[0…LAST] TO A[LAST]
    //DETERMINE POSITION OF LARGEST IN A[0..LAST] AND STORE IN MAXINDEX
    INT MAXINDEX = 0;
    FOR (INT INDEX = 1; INDEX <= LAST; INDEX++)
    {
        IF (A[INDEX] > A[MAXINDEX])
            MAXINDEX = INDEX;
    }
    // MAXINDEX IS POSITION OF LARGEST IN A[0..LAST]
    // SWAP A[MAXINDEX] WITH A[LAST]
    INT TEMP = A[MAXINDEX];
    A[MAXINDEX] = A[LAST];
    A[LAST] = TEMP;
}
```

INSERTION SORT

```
//A[0..0] IS SORTED
FOR (INDEX = 1; INDEX <= N -1; INDEX ++)
{
    // A[0..INDEX-1] IS SORTED
    // INSERT A[INDEX] AT THE RIGHT PLACE IN A[0..INDEX]
    INT UNSORTEDVALUE = A[INDEX];
    SCAN = INDEX;
    WHILE (SCAN > 0 && A[SCAN-1] > UNSORTEDVALUE)
    {
       A[SCAN] = A[SCAN-1];
       SCAN --;
    }
    // DROP IN THE UNSORTED VALUE
    A[SCAN] = UNSORTEDVALUE;
    // NOW A[0..INDEX] IS SORTED
}
// NOW A[0..N -1] IS SORTED, SO ENTIRE ARRAY IS SORTED
```

QUICKSORT

- THIS RECURSIVE PROCEDURE WILL REPEATEDLY PARTITION THE SUBLISTS UNTIL A[START..END] IS SORTED:

```
VOID DOQUICKSORT(INT A[ ], INT START, INT END)
{
    IF (START < END)
    {
        // PARTITION A[START..END] AND GET THE PIVOT POINT
        INT PIVOTPOINT = PARTITION(A, START, END);
        // RECURSIVELY DO THE FIRST SUBLIST
        DOQUICKSORT(A, START, PIVOTPOINT-1);
        // RECURSIVELY DO THE SECOND SUBLIST
        DOQUICKSORT(A, PIVOTPOINT+1, END);
    }
}
```

HOW TO PARTITION A[START..END]

```
INT PARTITION(INT A[ ], INT START, INT END)
{
    INT PIVOTVALUE = A[START];
    ENDOFLEFTLIST = START;
    // AT THIS POINT A[ENDOFLEFTLIST] == PIVOTVALUE
    FOR (INT SCAN = START + 1; SCAN <= END; SCAN ++)
    {
        IF (A[SCAN] < PIVOTVALUE)
        {
            ENDOFLEFTLIST ++;
            SWAP(A, ENDOFLEFTLIST, SCAN);
            // AT THIS POINT A[ENDOFLEFTLIST] < PIVOTVALUE
        }
    }
    // MOVE THE PIVOTVALUE BETWEEN THE LEFT AND RIGHT SUBLISTS
    SWAP(A, START, ENDOFLEFTLIST);
    RETURN ENDOFLEFTLIST;
}
```

```
class MergeSort
{
    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]

// Driver method
    public static void main(String args[])
     {
        int arr[] = {12, 11, 13, 5, 6, 7};

        System.out.println("Given Array");
        printArray(arr);

        MergeSort ob = new MergeSort();

        ob.sort(arr, 0, arr.length-1);

        System.out.println("\nSorted array");
        printArray(arr);
     }
}
```

```
// Main function that sorts arr[l..r] using
// merge()

void sort(int arr[], int l, int r)
{
    if (l < r)
    {
        // Find the middle point
        int m = (l+r)/2;

        // Sort first and second halves
        sort(arr, l, m);
        sort(arr , m+1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}
```

/* **Merge Sort Class Continued** */

```
void merge(int arr[], int l, int m, int r)
{
    // Find sizes of two subarrays to be merged
    int n1 = m - l + 1;
    int n2 = r - m;

    /* Create temp arrays */
    int L[] = new int [n1];
    int R[] = new int [n2];

    /*Copy data to temp arrays*/
    for (int i=0; i<n1; ++i)
        L[i] = arr[l + i];
    for (int j=0; j<n2; ++j)
        R[j] = arr[m + 1+ j];
```

```
    /* Merge the temp arrays */

    // Initial indexes of first and second
subarrays
    int i = 0, j = 0;

    // Initial index of merged subarry array
    int k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
```

```
    /* Copy remaining elements of L[] if any */

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy remaining elements of R[] if any */

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```