```
package com.Jaynish;

public class Main {

   public static void main(String[] args) {

  // write your code here
   }
   public static void mergeSort(int [] a){
       int middle = a.length/2;
       // sort front half
      for (int j =0; j< middle -1; j++) {       // O(N/2)
          for (int i = 0; i < middle - 1; i++) {    // O(N^2/4)
              if (a[i + 1] < a[i]) {
                  int temp = a[i];
                  a[i] = a[i + 1];
                  a[i + 1] = temp;
              }
          }
      }

       // sort second half
      for (int j =middle; j< a.length-1; j++) {     // O(N/2)
          for (int i = middle; i < a.length - 1; i++) { // O(N^2/4)
              if (a[i + 1] < a[i]) {
                  int temp = a[i];
                  a[i] = a[i + 1];
                  a[i + 1] = temp;
              }
          }
      }
       System.out.println("\n");
     for (int i = 0; i < middle-1; i++){   // O(N/2)
          for(int j = middle; j<a.length; j++){  // O(N^2/4)
              if (a[j]<a[i]){
                  int temp = a[j];
                  for(int k = j; k > i-1; k--){
                      a[k] = a[k-1];


                  }
                  a[i] = temp;
                  i++;
              }
          }
      }
     for (int j =0; j < a.length; j++){
          System.out.print(a[j]+" ");
     }
   }
```

}

B) The worst case complexity of the in place merge sort is O(N^2) because of the various double for loops that were used to sort the left half and the right half and then merge the two sorted halves.

The best case complexity of the in place merge sort is also O(N^2) as the for loops will be executed even if the given input is already sorted.

The highlighted (yellow) nested for loops cause bottlenecking and have a time complexity of O(N^2).

The highlighted(blue) operations keep the in place merge sort stable, as they are just very basic swap operations that are stable.