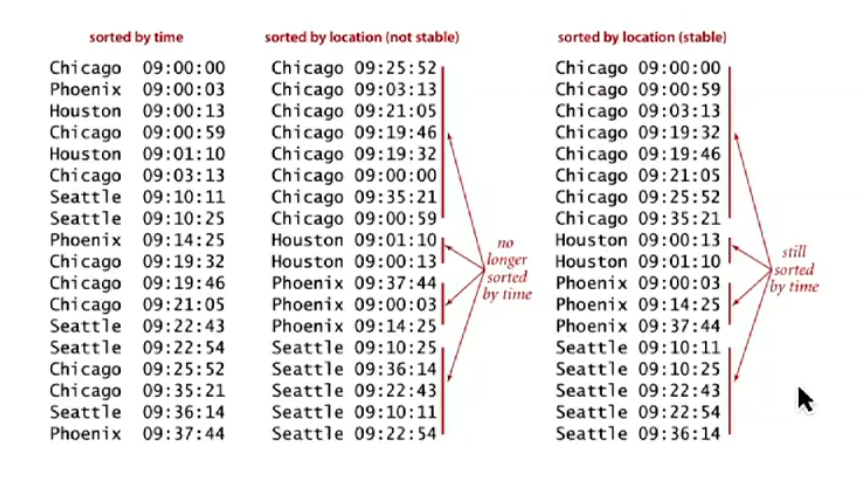


**Stability:** It is a property of the sorting algorithms. A stable sort preserves the relative order of items with equal keys. Insertion and Merge sort are stable. Selection and Shell sort is not stable.

But to be sure need to check the implementations to make sure the sort is stable or not (less than vs less than equal to)



# Merge Sort:

Here we use divide and conquer method. We divide the array into two halves, recursively sort each half and finally merge the two halves.

**Implementation:**

1. Copy everything to an auxiliary array.
2. Next step is copying the items from auxiliary array to original array. But this time on sorted order.
3. For this we need to keep 3 index pointers

**i** = current entry of left-half of aux array,

**j** = current entry of the right-half of aux array

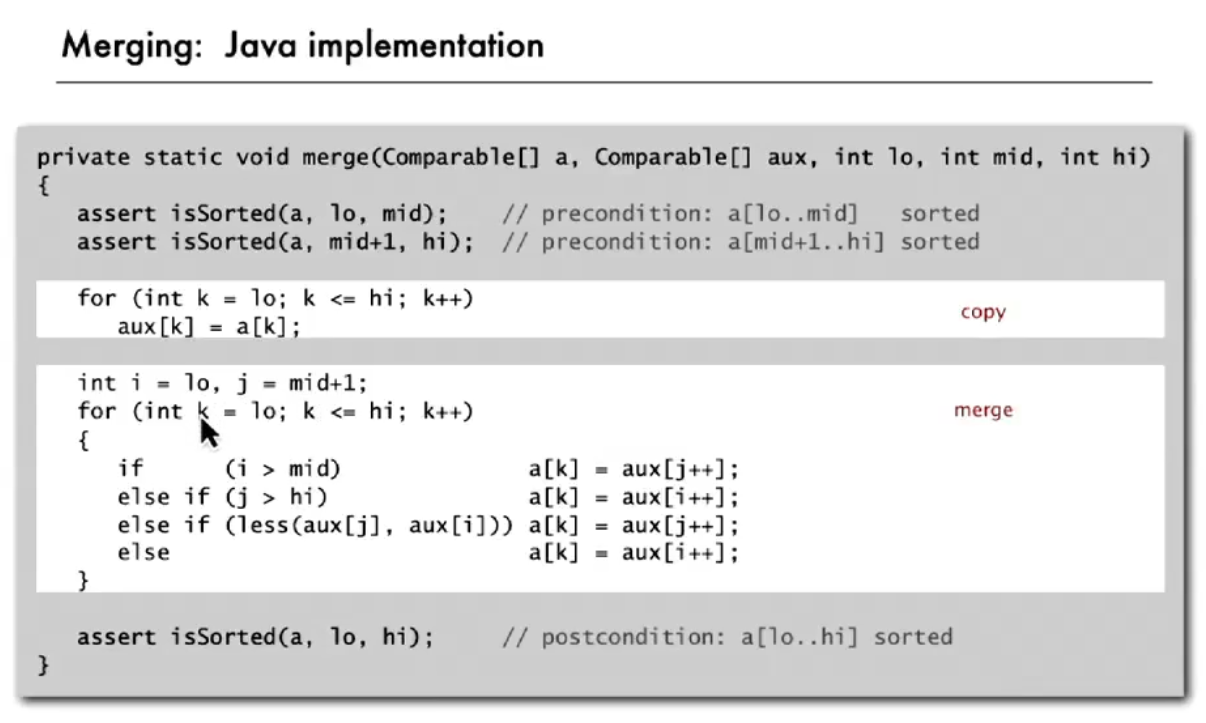
**k** = current entry of the sorted array

1. Next step is compared minimum in each subarray.

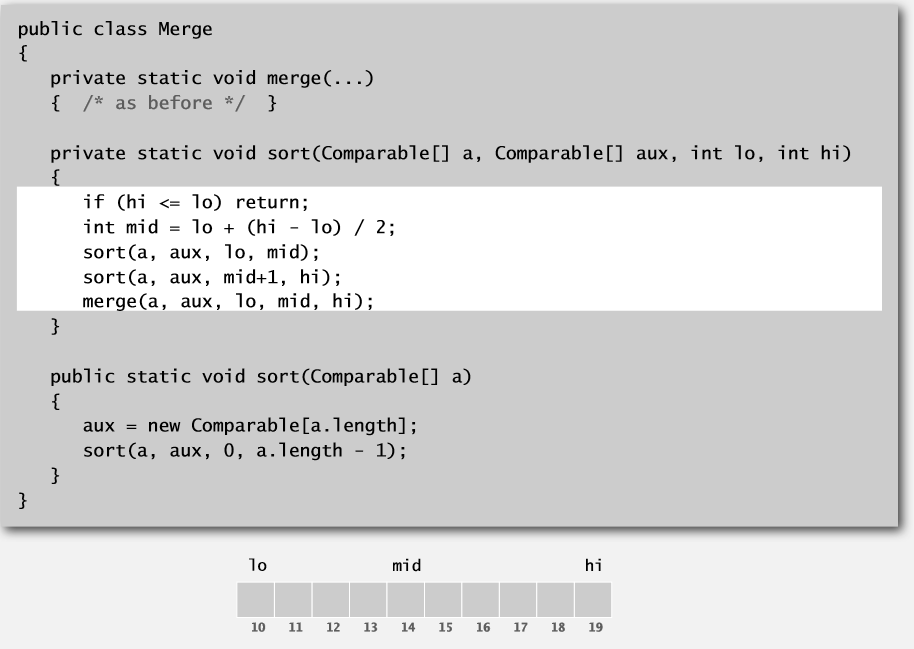
**Case 1:** If (aux[i] <= aux[j]) then arr[k] = aux[i]; k++; i++;

**Case 2:** if (aux[i] > aux[j]) then arr[k] = aux[j]; k++; j++;

1. If one of the sub-arrays is empty, we just take the rest of the element from other array.

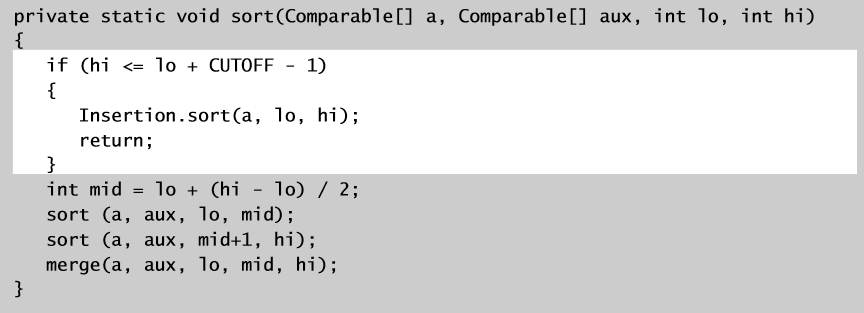


it’s important not to create the auxiliary array in the recursive function. It will create extra overhead in terms of space. This bug sometimes makes merge sort perform poorly. Thus, it is created in this method which in turn calls the sorting method.

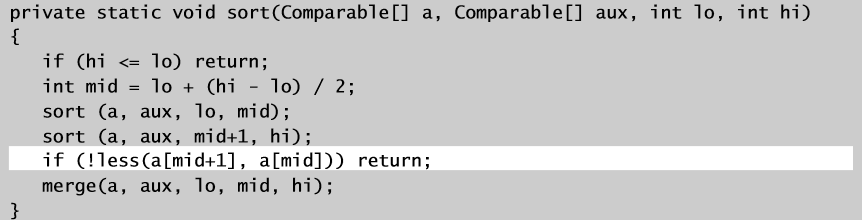


**Improvements:**

1. For smaller size array merge sort is too much overhead for tiny subarray. So, we cutoff for insertion sort for around 7 items.



1. Another improvement can be to stop if the array is already sorted. We can check if the biggest item in the first half <= smallest item in the second half to make sure

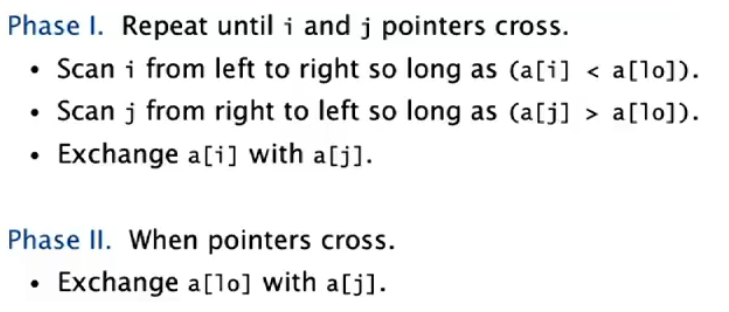


# Quick Sort:

Quick Sort is in place algorithm. So, doesn’t need the extra space unlike merge sort. It is not stable sort.

1. Shuffle the Array
2. Partition the array in a way, that for some j
   1. Arr[j] is in place
   2. No smaller entry to the right of j
   3. No larger entry to the left of j.
3. Then Sort each piece recursively.

**Partitioning:**

****

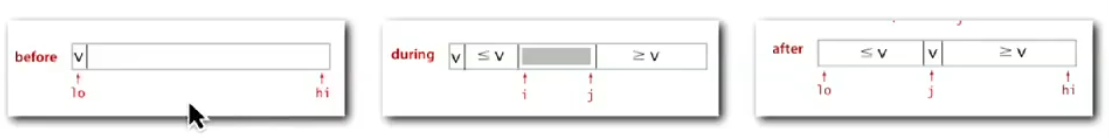






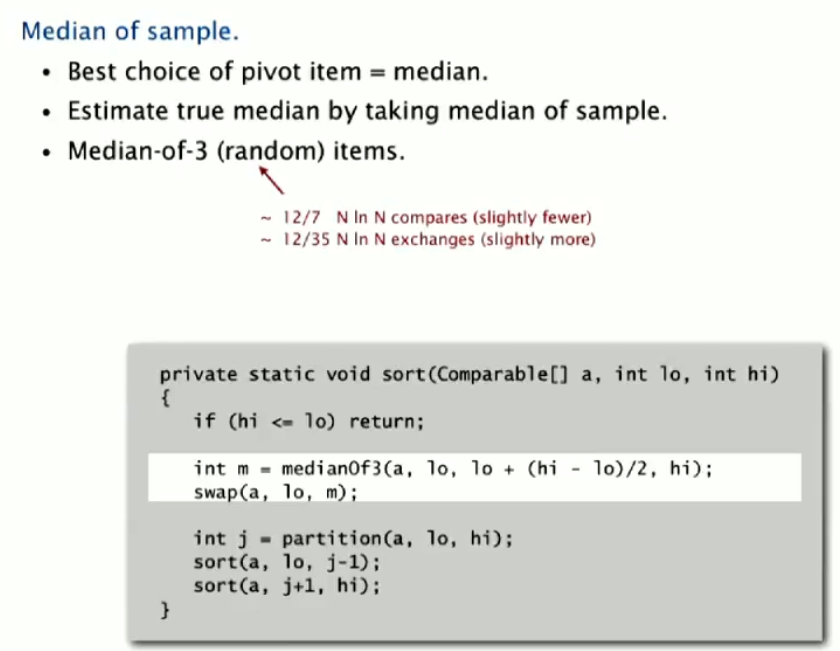






**Improvement:**

1. Similar to merge sort, for smaller subarray too much overhead. So we can cutoff to insertion sort for around 10 items.
2. Estimating the partitioning element near the middle.

****

**Performance:**

