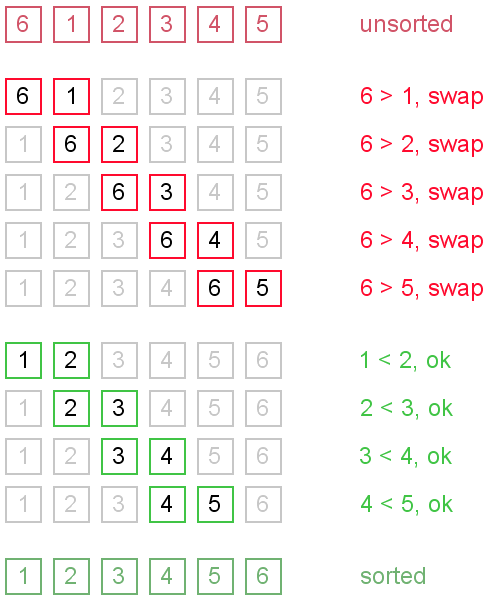
**Assignment 3**

**Bubble sort:**

Bubble sort is a simple sorting algorithm in which each element is compared to the adjacent element and swaps the elements if left element is greater than right element. This step are continued till the whole list is sorted.



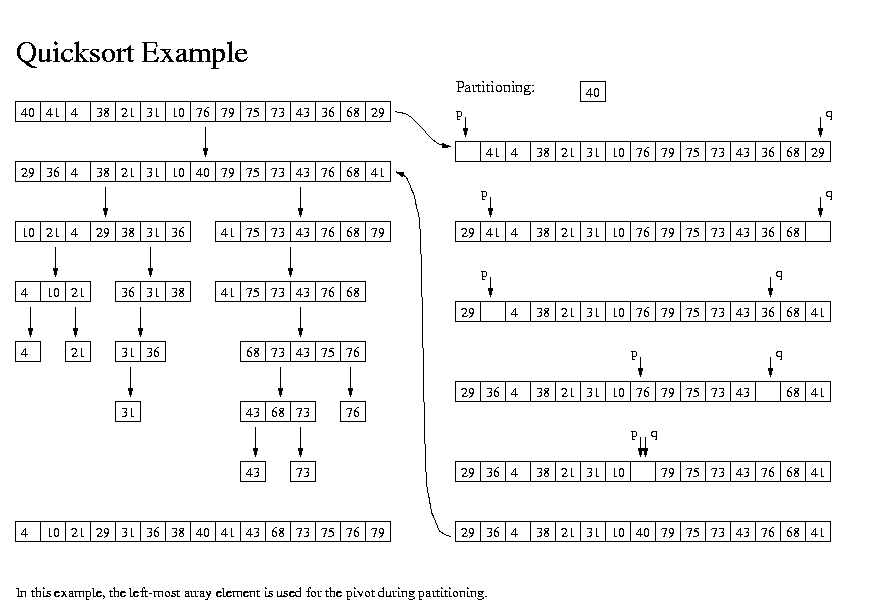
**Time complexity**: In worst case, the list must be iterated n times for each elements of the list, its time complexity is O(n2)

**Quick sort:**

Quick sort is based on partitioning the list into smaller list. A large list is partitioned into two list one of which holds values smaller than the specified value, say pivot, based on which the partition is made, and another list holds values greater than the pivot value.

First step is to choose the pivot, generally the element at the highest index. Take 2 variable to point to the left for lower index and right for higher index of the list. If value at left is less than pivot than increment the left variable and if value at right is greater than pivot than decrement right. If both the previous condition fails than swap the element at left and right. And if left > = right than the point where they meet is the position of pivot.

Quick sort partitions an array and then calls itself recursively twice to sort the two resulting subarrays



**Time complexity:** The time complexity of quick sort is O(n2)

**Selection sort:**

Selection sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty, and the unsorted part is the entire list. The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right



**Time complexity**: The time complexity of quick sort is O(n2)

**Merge Sort:**

Merge sort is a sorting technique based on divide and conquer technique. Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. The elements are combined in the exact manner as they were broken

For example:

Merge Sort Division

Merge Sort Division

Merge Sort Division

Element are compared in pairs and swapped if not in correct order.

Merge Sort Combine

In the next iteration of the combining phase, compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

Merge Sort Combine

And repeat will we get a final merge sorted list.

Merge Sort

**Time complexity:** The time complexity of quick sort is O(n log n)