**TASK 1:** For this task we used count inversion for counting the possible pairs. It is also a divide and conquer method, we divide the array with recursion and first we check the left portion of the array till mid(array[:mid]) and count the pairs then we check the right portion from mid to the last(array[mid:), then we cross check both of the divided array and return the sum of the left,right and cross checked pairs count. The time complexity of this task is O(nlogn).

**TASK 2:** For this task we gone for the divide and conquer method, we divide the array with recursion and check the left portion and right portion recursively and then we check the cross as adding the max of the left portion and the max of square of the left portion, then finally return the maximum value from the left, right and cross checked value. The complexity for this task is O(nlogn)

**TASK 3:** For this task we used quick sort algorithm, which also follows divide and conquer method but this time we divided the array using a pivot element, for this code we have chose the pivot element as the first element of the array, after every partition we keep the pivot element in it’s sorted position and return the pivot index, using the pivot index we keep conquering the sub arrays while sorting with recursion and in the end we get the array sorted.

**TASK 4:** For this task we used the partition function from the quick sort algorithm. As we know, everytime partition happens the pivot element gets placed into its sorted position. So if the pivot index+1 (as the task is 1 index based) is equal to the k, then we actually find the kth smallest element, if the index is smaller than k then we know the element is in the right sub array so we only work with that part and again if the index is larger than k then the element is in the left sub array, that’s how we find the kth smallest element.