3) Quick Sort

User to input a number of elements (up to 25) that requires sorting. Before the quicksort is called, the user inputs the number of elements to be sorted and then the elements themselves.
We store the 25 numbers (the array's elements) in the array number, and represent the first and last element with the variables first and last. We then call the quicksort function, which moves the algorithm onto the next step.

If the first element is smaller than the last element, it sets the pivot to the first element.
It calls a while loop to increment i and decrement j depending on their relation to the pivot. In simpler terms, this checks whether elements lower/higher than the pivot and divides the whole array into two sub-arrays; this is the partition step.
The quicksort then recursively calls itself to sort the two sub-arrays accordingly until the whole array is sorted. And finally, the sorted array is printed.

----

4) Find kth maximum by partitioning the array

To find the K largest(or smallest) elements in an array using Quick Sort partitioning algorithm, we will find the pivot in the array until pivot element index is equal to K, because in the quick sort partioning algorithm all the elements less than pivot are on the left side of the pivot and greater than or equal to that are on the right side. So we can print the array from (N-pivot_Index) to N for K-largest elements)

First, we choose a pivot number
if K is lesser than the pivot_Index then repeat the step
if K is equal to pivot_Index: Print the array ((n-pivot_Index) to n for K-largest elements)
if K is greater than pivot_Index: Repeat the steps for the right part

----

5) Find the first and second minimum and maximum

To find the first maximum and minimum we declared two variables fmax and fmin, and initialize both of them with the first element of an array. Now, using for loop we find the first maximum and first minimum element. If an array element is greater than fmax than assign it to fmax, similarly if an array element is lesser than fmin than assign it to fmin. When for loop execution completed we got first maximum and first minimum element of the array.

Now, we use two variables smax and smin for second maximum and minimum. We initialize smax=fmin and smin=fmax. The logic is that the second maximum element will be never lesser than fmin in the array and second minimum element will be never greater than fmax in the array. If the array element is greater than smax and lesser than fmax then update smax with this element of the array. Similarly, if the array element is greater than fmin and lesser than smin then update smin with this array element. Finally, the required output is printed

7) Write an efficient program to delete all duplicate elements from a sorted array

Duplicates are always adjacent in a sorted array. Suppose we want to remove duplicate elements from an array arr. For that, we will run a for loop in which i will point to the index of each element of arr. We will make another pointer j that will point to the index of the elements of a new array temp where non-duplicate elements will be stored. We will initialize j by 0 as temp is initially empty. Then we will check whether the ith element of arr is equal to the (i+1)th element. If it is, then we will increment i by 1, else we will store the ith element in temp. Then we will increment i by 1 and j by 1.

We will run the for loop from the 1st element till the second last element of arr. We will not include the last element in the for loop as there is no (i+1)th element for the last element. After the loop is completed, store the last element in temp and increment j. Then copy the j elements of temp to arr and print the same.