**LOOK UP TABLE TECHINQUE:**

The LOOKUP TABLE TECHNIQUE is used to find out the index of the least valued element in the given part of the array.

If the given input is an array of the size ‘n’. The computed results are stored in a 2D array of size n\*n. It uses only the upper diagonal of the matrix to store the results. And the Lower is not used to store the result.

**Process:**

* First create the n\*n matrix.
* Let one side of the array index be ‘i’ and other be ‘j’. And ‘indx’ and ‘val’ be two variables.
* For j=0, check the value of the first element in the array, and assign value of index to variable indx and the value element to variable val and fill the indx’ in the matrix’s [j][i]th place as you check the further element in the array increment I with 1.
* if the value of the next element in the array is greater than the val the value of indx is not changed and fill the same value of indx in the matrix. move along horizontal until find the end in the matrix.
* Else if the value of the next element in the array is lass than the variable val, then assign the value of that element to the variable val and assign the value of index of that element to the indx variable. and fill the value of the indx variable in the matrix until you get the further smaller valued element.
* as you move to next row increase ‘j’ by 1 and start filling that row from there.
* at last we get the results stored in the 2D upper diagonal matrix.

**Conclusion:**

* The process time complexity = n2.
* Spacing technique complexity = n2.
* fetching result complexity =1.

**SQUARE ROOT DECOMPOSITION TECHNIQUE:**

The SQUARE ROOT DECOMPOSITION TECHNIQUE is used to reduce the space complexity of the LOOKUP TABLE TECHNIQUE.

Let us consider given input array is of size N. Then this technique uses an array of size sqrt(size\_of(input array)) to store the results.

**Process:**

* First calculate the value of sqrt(size\_of(input array)) and let that be m.
* Divide the given array into ‘m’ parts.
* Consider each part of the array and search for the smallest value in that part and copy the value of the index of the smallest value in the corresponding index of the resulting array.
* Repeat the above step until all the parts are completed.

**Conclusion:**

* But when the query is passed to get the index of the smallest element, first we have to check whether all the segments in the block are covered are not, if not check separately for each case. Else pick the value In the corresponding index of the resulting array.

**SEGMENT TREE:**

The input array is divided into two parts n those two arrays are further divided until we get a single element in all the arrays. It forms a binary tree structure.

**Process:**

* When at last we get the all single elemental arrays then then both the child values are compared and the parent is returned with the index of the least value.
* further the process is continued until the root node is reached.
* the value returned at the root node is the minimum element in the array.