**Modules and their descriptions**

1. Main module

Has all global variable declarations. Elements from the DOM are selected and event listeners are assigned to all elements the user can interact with. The webpage window upon loading calls a sequence of functions like *renderInitialBars(), renderBars()* etc. The job of these functions is to create a 10 element array with random values and render its corresponding bars on the screen. All bars rendered on the screen are then selected and stored in another div element array called ‘blocks’. Various sorting modules then make use of ‘blocks’ array to target individual bars and apply animations on them

1. Animations module

Contains functions like *turnyellow(),turnblue(),turngreen(),turnred().* These take the indexes of the bars to be animated and change their color. This module also contains specific animation functions like *AnimateMergeSort().* The exact role of these functions will be explained in further modules

1. Utility Functions module

Contain several utility functions that are being used throughout the code. For example there is *swapelements(a,b)* which exchanges elements in the element array. There is also *swapbars(a,b)* which exchanges elements in the blocks array. *setHeight(bar)* is responsible for reading the value of each bar in the array stored in JS and deciding its height when it will be rendered as a bar on the screen. Another very important utility function is the *sleep()* function. This function uses JS promises to put the execution of the code to rest for a specified number of milliseconds. This creates the much needed pauses in the animations.

1. BubbleSort module

the bubble sorting algorithm is coded in JS. as the algorithm proceeds to sort the array, the execution is stopped using *await sleep() function* . During these pauses, the relevant animation functions are called to show the animation effect

1. mergeSort module

Contains the merge sort algorithm. We have not used the traditional version of merge sort that uses O(n) space. We have rather used a modified version which does not require any auxiliary space and instead arranges the elements inside the parent array itself. Since the parent array keeps changing each time merging is happening, it becomes harder to animate this process. So instead of animating the blocks on the fly as the sorting is happening, a different approach is followed. In this we let the sorting process complete first. As this process is happening, all the animations to be performed are recorded in the animations object and not applied immediately. This object is then returned by the *mergeSort()* function. Another function called *animateMergeSort(instructions)* is then called and the animation object is passed to it as a parameter. This functions then reads over the animation sequence stored inside the animation object and applies those animations on the blocks that are seen on the screen.

1. quickSort module

Contains 2 main functions…*quickSort()* and *Partition().* *quickSort()* calls itself recursively till the algorithm completes. In each call of *quickSort()*, it calls *Partition()* function. *Partition()* is responsible for its core function as well as for animation part. In every call of *Partition(),* it firstly modifies ‘blocks’ array with freshly selected values of all bars from the DOM. This is done so that the latest properties of the bars like color and value are updated in the blocks. Then as the execution of the function continues the appropriate animations are applied by changing the color and value properties of bars inside the blocks array. *await sleep()* is called at the appropriate moments to give pauses in between the animation.

1. heapSort() module

Contains 2 main function…*heapSort(arr)* and *heapify(arr,N,i)*. First the whole array is heapified/ arranged in the form of a maxHeap. To show how the heap looks, the heap is displayed on the screen in the form of bars. Here the execution stops for 4 seconds to let the viewer grasp what has happened. After this, the heap sort algorithm begins execution. As each element is removed from the heap, it is the free space available at the end of the array and colored purple to signify that it is sorted. Then heapify() function is called again on the unsorted array. The first element in the Maxheap i.e. the maximum element is colored yellow.