**CSE 5311-001**

**PROJECT 1**

**PROJECT ON SORTING TECHNIQUES:**

Implement and compare the following sorting algorithm:

* Merge sort
* Heapsort
* Quicksort (Regular quick sort\* and quick sort using 3 medians)
* Insertion sort
* Selection sort
* Bubble sort

**OVERVIEW OF SORTING TECHNIQUES:**

Sorting technique is a process of arranging a collection of items or data in a specific order. Sorting techniques are used in computer science to sort data in a particular order. There are many sorting algorithms available where it often depends on the size of the data to be sorted and the constraints of the system on which it is running.

Below is the table shows the run time complexity of each sorting techniques.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type of Sorting** | **Best Case** | **Average Case** | **Worst Case** |
| Insertion Sort | O(n) | O(n^2) | O(n^2) |
| Selection Sort | O(n^2) | O(n^2) | O(n^2) |
| Heap Sort | O(n log n) | O(n log n) | O(n log n) |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) |
| Quick Sort | O(n log n) | O(n log n) | O(n^2) |
| Bubble Sort | O(n) | O(n^2) | O(n^2) |

**PROCEDURE:**

We need to sort the array of numbers from the set of random numbers. There are different sorting techniques that have been used for sorting. Also we have to observe that the how efficient the run time complexity of each sorting technique and for this there are three cases i’e Best case, Average case, Worst case.

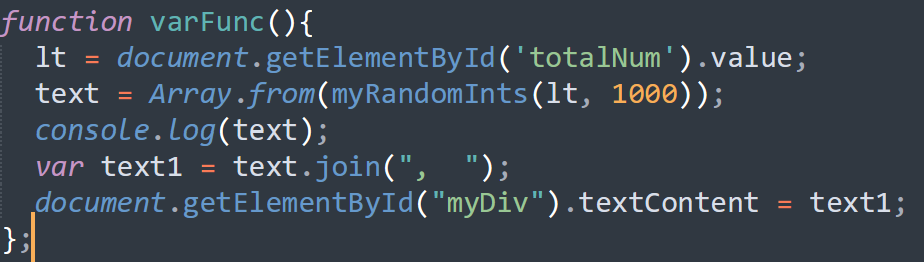
**IMPLEMENTATION:**

To get an array of numbers to be sorted, first we need set of numbers. Here I have implemented a function to generate random numbers. These generated random numbers are stored in a variable so If I pass them to the sorting function, it displays the numbers sorted according to each technique.

For the UI part I have used HTML and for the backend operations JavaScript have been used for sorting the numbers.

**Backend implementation part:**

The function for randomly generated numbers:

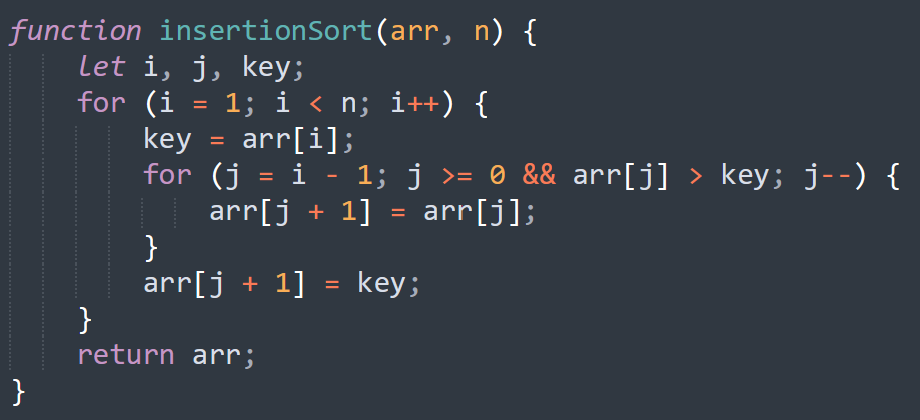


Here a user will give input of how many numbers to be generated. So it generates random numbers upto 1000.

Then, these numbers are stored in a an array and passed through different sorting techniques

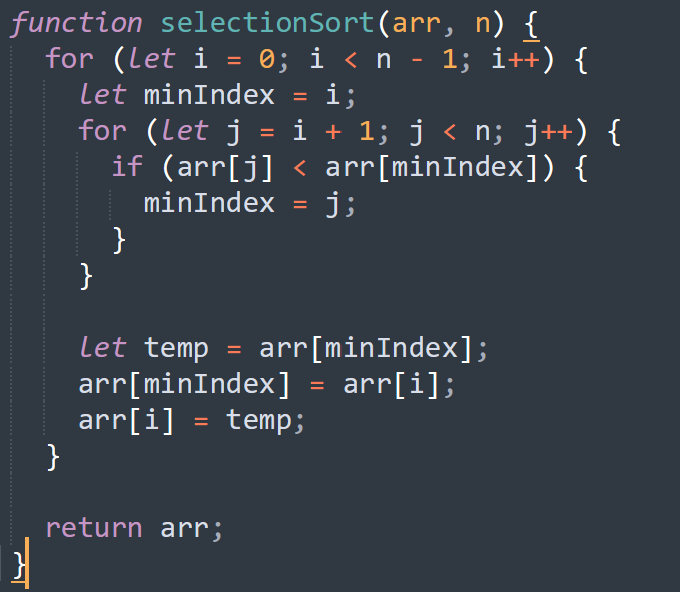
**Insertion Sort:**

In this code the insertion sort compares each element to adjacent element and swaps if it is in wrong order. Below is the code I have implemented.



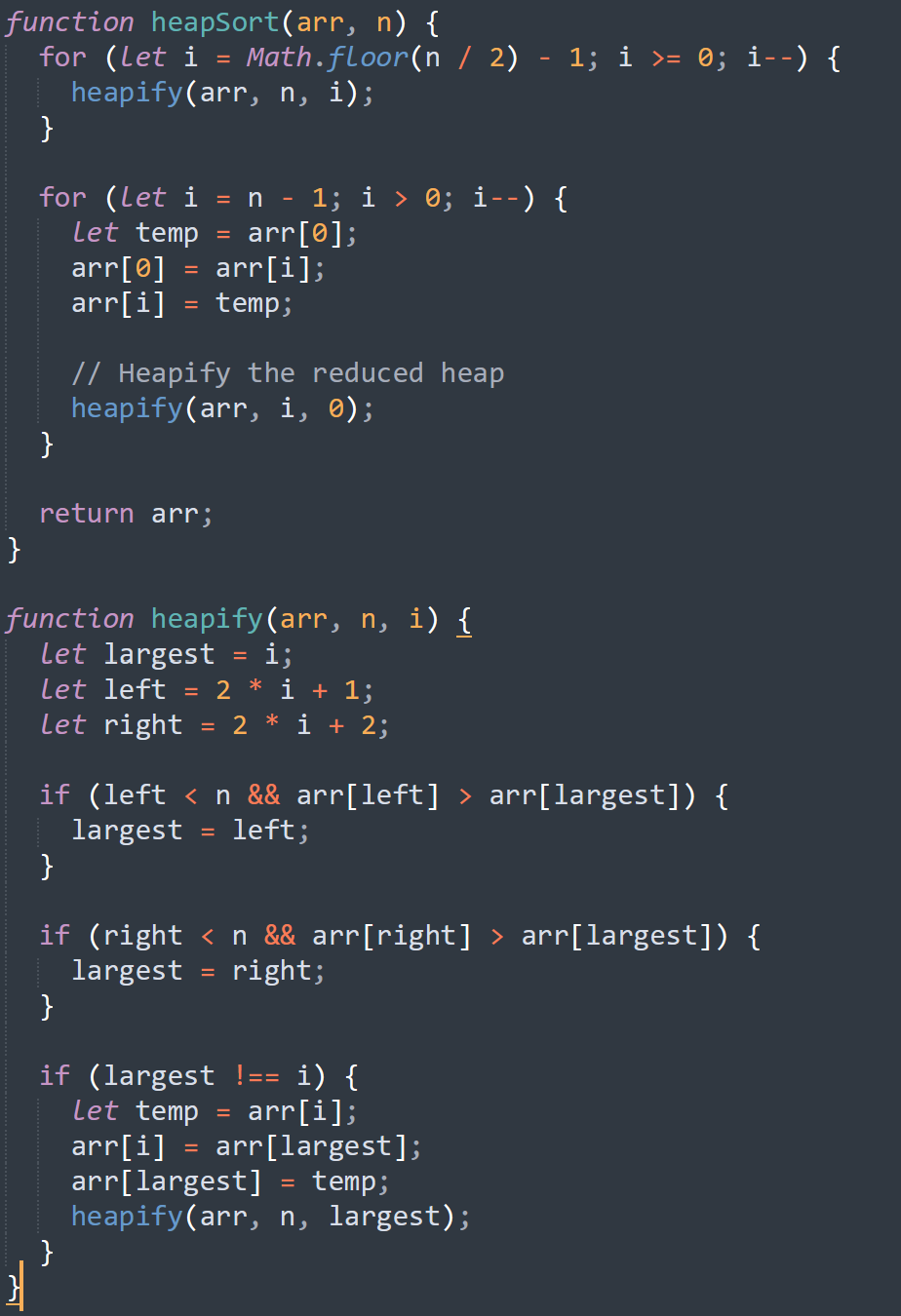
**Selection Sort:**

In the below code, it check the minimum element in the array and swaps with the first portion of the array if it is unsorted.



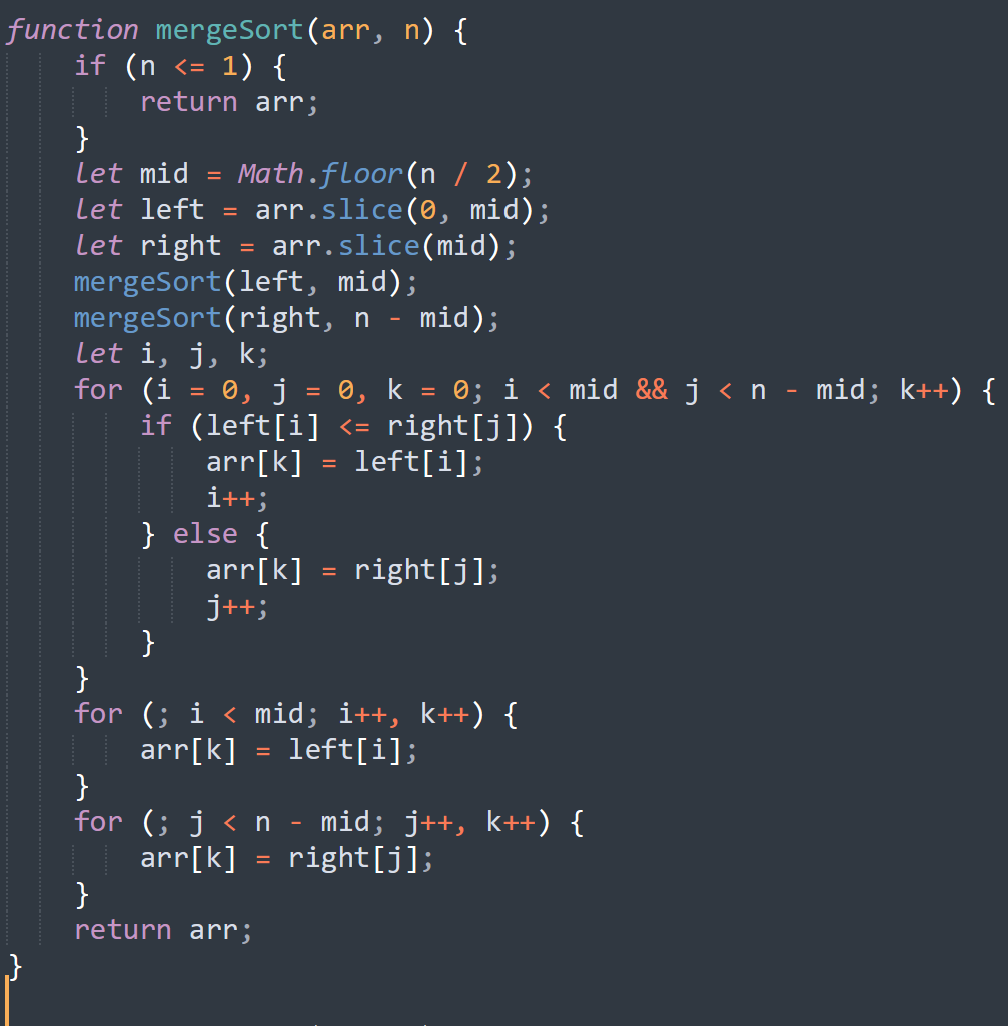
**Heap Sort:**

Heap sort is comparison-based algorithm. It builds binary heap, gets the maximum element from heap and places at the end of the array.



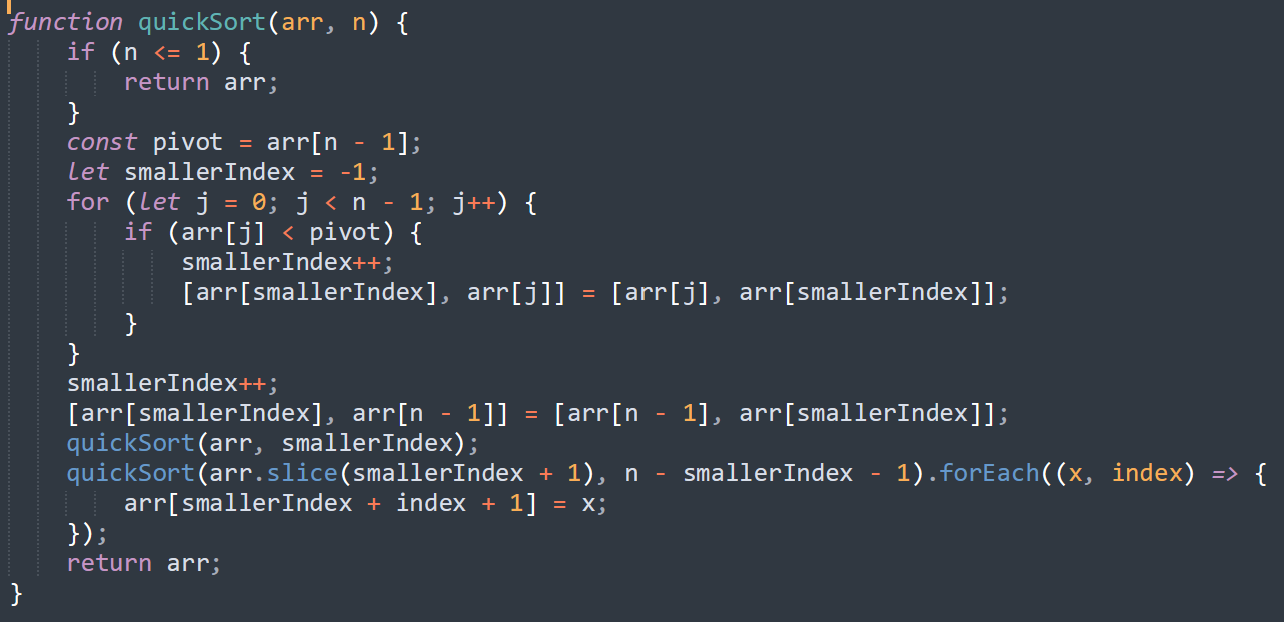
**Merge Sort:**

Merge sort follows divide and conquer approach. It basically divides the array into two and sorts each halves and merges back again.



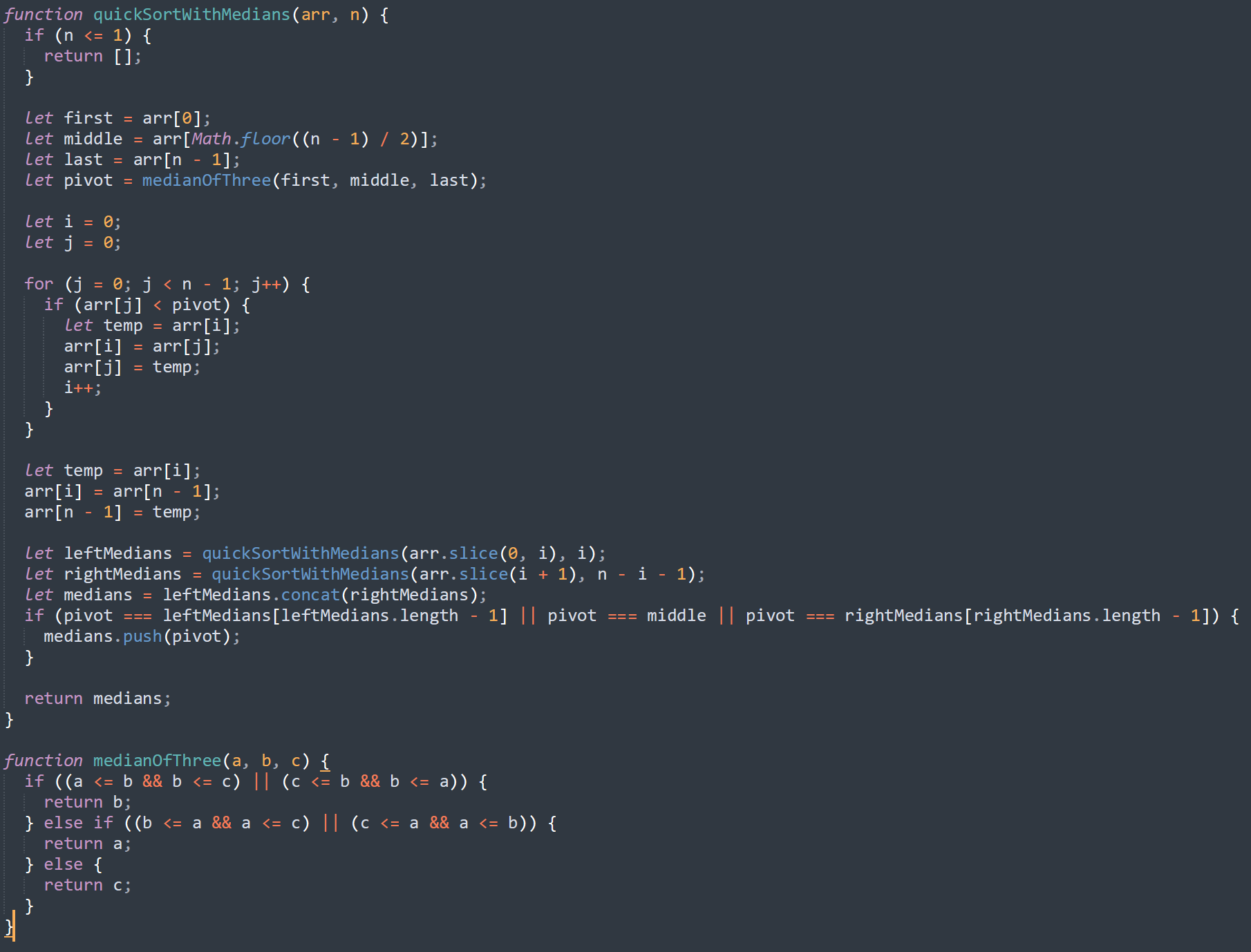
**Quick Sort:**

It is a sorting technique that selects the pivot element where the array should contain the elements which is less that pivot element and elements that must contain greater that pivot element.



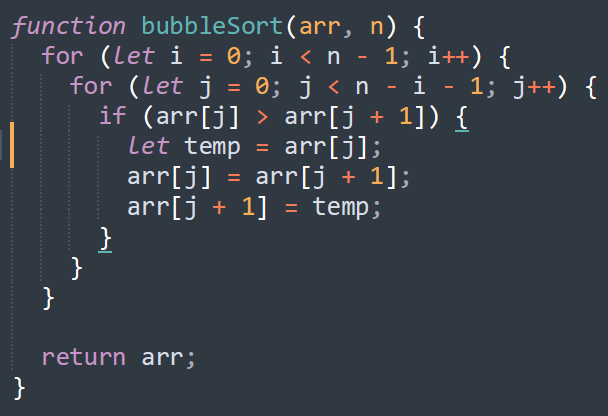
**Quick Sort with 3 medians:**

In this kind of quick sort, it selects three pivots from the array i’e first element, middle element and last element and gives the three medians



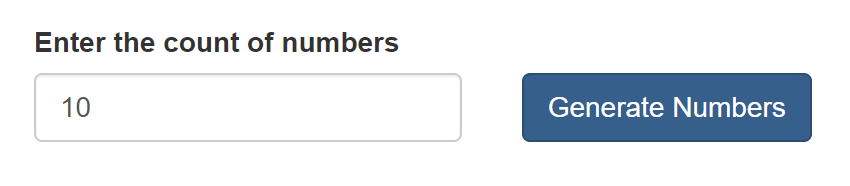
**Bubble** **Sort**:

It compares the adjacent elements and swaps the elements if the element is greater than the adjacent element



**Frontend Implementation Part:**

I have used HTML, bootstrap as frontend. A textbox have been created for giving the count of numbers to be generated.



A checkbox of options have been created for each sorting techniques. If we selected any of those or all of those then the elements will be sorted according to the techniques that has been created.

**EXPERIMENT:**

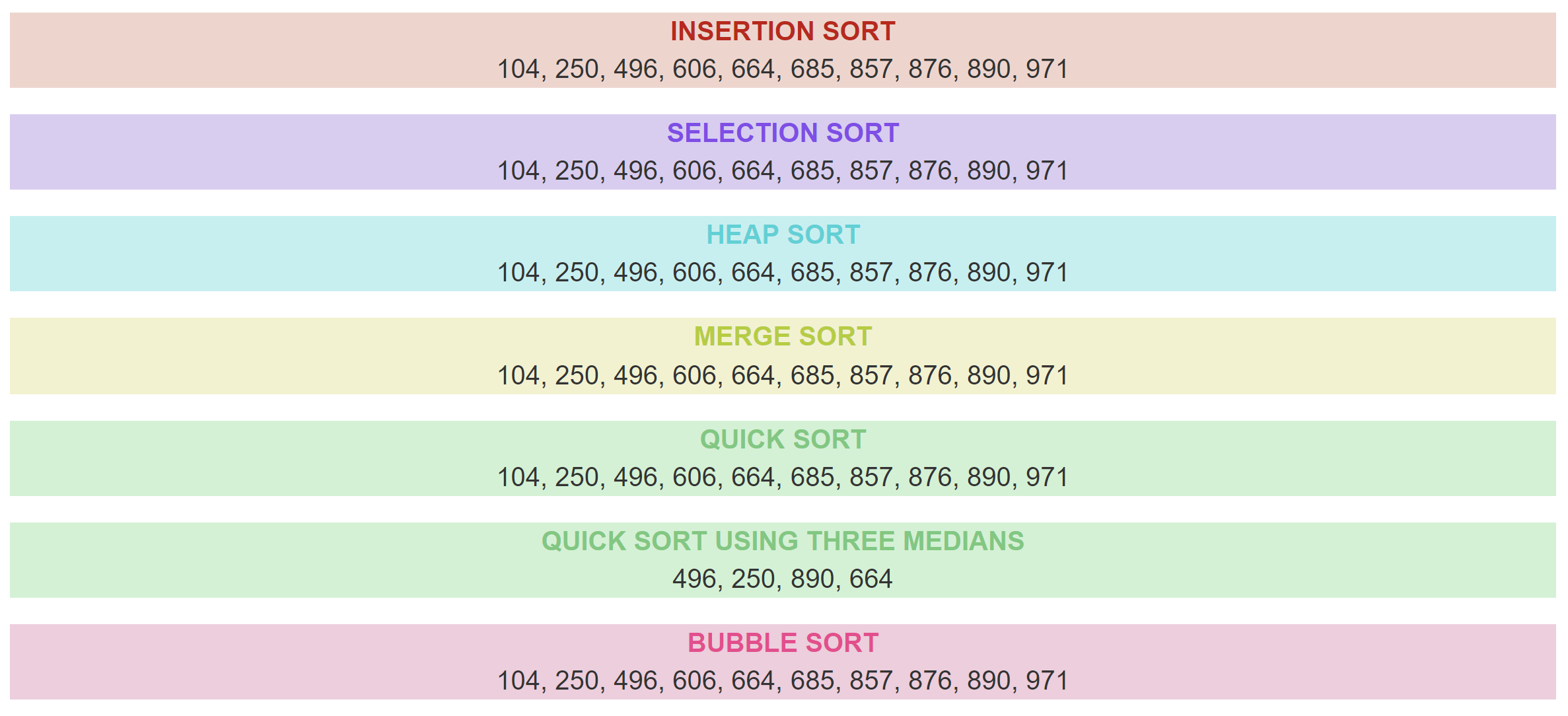
**Experiment 1**

Let us generate random numbers from the function we have implemented

890, 664, 496, 250, 876, 685, 104, 606, 971, 857

These are the numbers randomly generated from the function.

I have selected all the techniques and clicked on ‘Answer’ button.



|  |  |  |  |
| --- | --- | --- | --- |
| **Type of Sorting** | **Best Case** | **Average Case** | **Worst Case** |
| Insertion Sort | 0.039794921875 ms | 0.018798828125 ms | 0.002197265625 ms |
| Selection Sort | 0.0390625 ms | 0.00390625 ms | 0.0029296875 ms |
| Heap Sort | 0.057861328125 ms | 0.01416015625 ms | 0.0068359375 ms |
| Merge Sort | 0.0810546875 ms | 0.010009765625 ms | 0.005126953125 ms |
| Quick Sort | 0.22412109375 ms | 0.01513671875 ms | 0.017822265625 ms |
| Quick Sort 3 Medians | 0.098876953125 ms | 0.02392578125 ms | 0.00927734375 ms |
| Bubble Sort | 0.02197265625 ms | 0.003173828125 ms | 0.0029296875 ms |

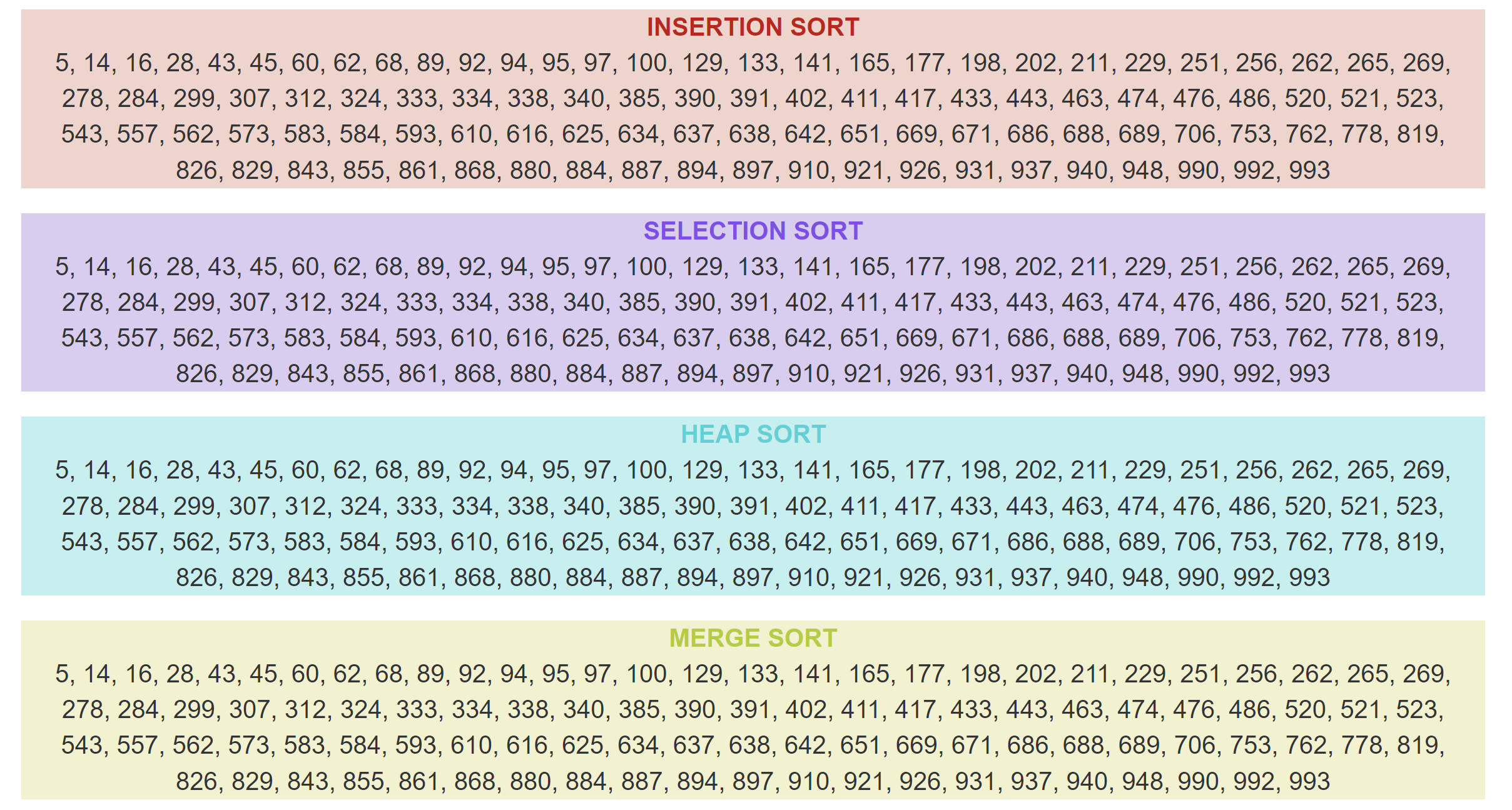
This above table shows that the run time complexities of best, average and worst case of the sorting techniques for the 10 random numbers. As we can observe that the quick sort is having the highest run time complexity with 0.2 milliseconds whereas the bubble sort is having lowest run time complexity with 0.02 milliseconds.

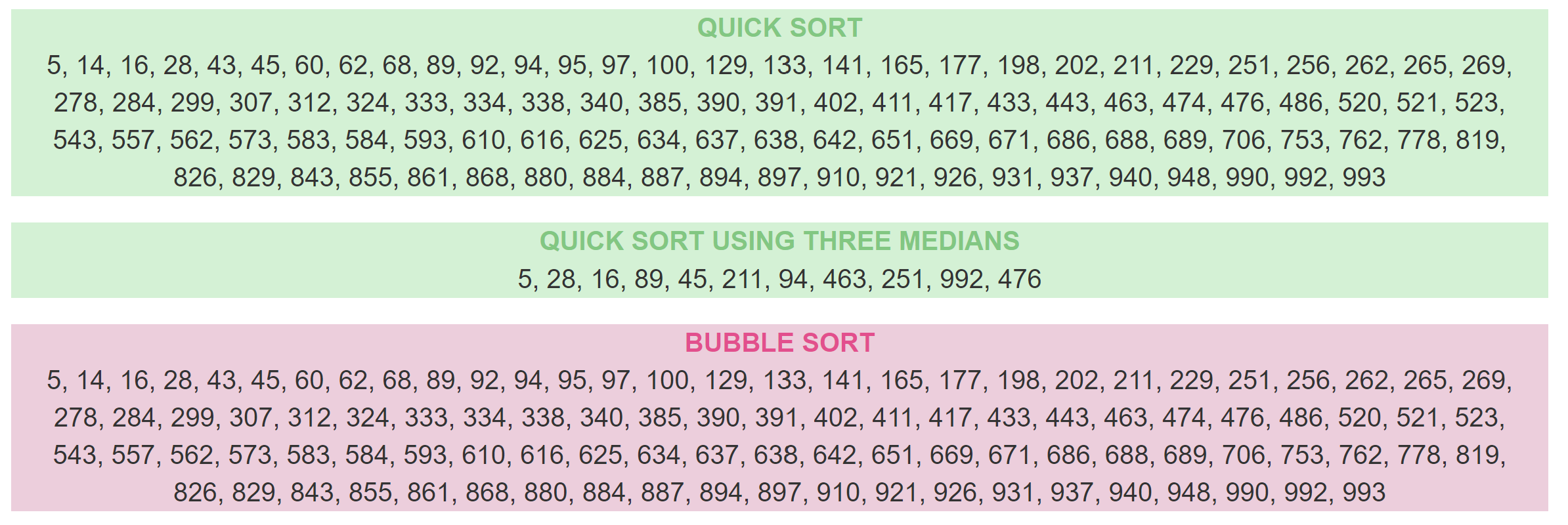
**Experiment 2**

Let us generate random numbers of 100 numbers

931, 129, 826, 937, 391, 762, 45, 333, 251, 177, 819, 887, 14, 610, 686, 402, 463, 62, 669, 43, 68, 324, 843, 573, 133, 584, 278, 940, 486, 165, 202, 638, 557, 855, 307, 141, 868, 543, 92, 443, 688, 476, 583, 706, 990, 894, 474, 926, 390, 269, 625, 198, 5, 385, 284, 262, 89, 910, 95, 671, 100, 948, 433, 521, 97, 94, 229, 523, 634, 16, 334, 520, 880, 897, 211, 651, 616, 753, 265, 829, 992, 921, 593, 28, 417, 411, 562, 778, 340, 861, 312, 60, 256, 299, 884, 338, 637, 642, 993, 689

Let’s select all the techniques

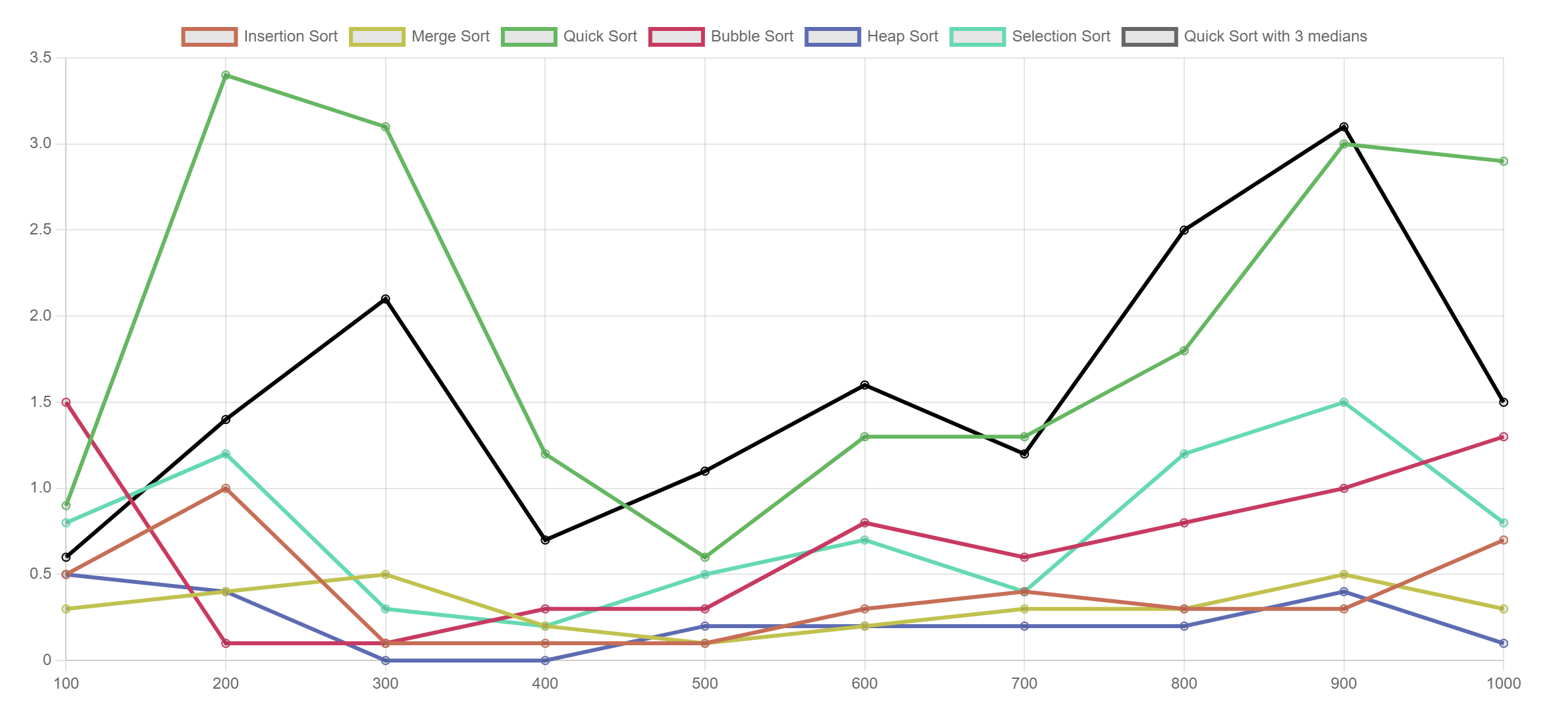




|  |  |  |  |
| --- | --- | --- | --- |
| **Type of Sorting** | **Best Case** | **Average Case** | **Worst Case** |
| Insertion Sort | 0.04736328125 ms | 0.108154296875 ms | 0.005859375 ms |
| Selection Sort | 0.18701171875 ms | 0.11181640625 ms | 0.169189453125 ms |
| Heap Sort | 0.057861328125 ms | 0.01416015625 ms | 0.0068359375 ms |
| Merge Sort | 0.14306640625 ms | 0.051025390625 ms | 0.051025390625 ms |
| Quick Sort | 1.460205078125 ms | 1.124755859375 ms | 1.033935546875 ms |
| Quick Sort 3 Medians | 0.215087890625 ms | 0.072021484375 ms | 0.10595703125 ms |
| Bubble Sort | 0.14599609375 ms | 0.091064453125 ms | 0.10791015625 ms |

The above table shows the run time complexities for 100 random generated arrays. Quick sort takes more run time with 1.46 milliseconds whereas insertion sort takes less time with 0.04 seconds.

**GRAPH FOR RUN TIME FOR DIFFERENT SORTING TECHNIQUES:**



In the above graph, X-axis represents the input size and Y- axis represents performance time. It represents that merge sort starts first with 0.4 milliseconds and ends with 2.8 milliseconds for the input size 1000 whereas bubble sort starts at 1.5 milliseconds but Heap sort ends with least time i’e 0.2 milliseconds.