Report

# Approach

I modified the existing algorithms of radix, insertion, selection, quick, merge and heap sort by adding code to measure time, comparisons and movements. In each algorithm’s start, the code starts measuring time and it stops measuring time when the array is completely sorted and when the algorithm completes.

To measure comparisons and movements, there are two integer variables that increment whenever a comparison is done and whenever an element in moved in the array respectively.

To generate different types of array, I used:

* Java.util.Random to generate the random array of X elements.
* A simple for loop to store index number at the respective indices for “In Order” array.
* An 80% and 20% split in which 80% is sorted and 20% is random in “Almost Ordered” array.
* A for loop which stores “length – index” at respective index for “Reverse Order” array.

# Challenges and Problems

I faced multiple challenges while measuring the performance of these algorithms using java’s time module. The times were not consistent each time, because IDE’s do caching sometime and the execution times increase after the first try.

Another problem I faced was that the statements which increment the number of comparisons and movements have to be written in the loops (in mid of algorithm), although it has a negligible effect on the execution time, but it may have a little effect.

# Assessment

Following results were obtained from the experiment.

|  |  |  |  |
| --- | --- | --- | --- |
| **Experimental Results** | **ArraySize** | **15286** |  |
| **List Property: InOrder** | **Comparisons** | **Movements** | **Total Time** |
| **Insertion Sort** | 0 | 0 | 3ms |
| **Selection Sort** | 116,838,540 | 0 | 57ms |
| **Quick Sort** | 228, 191 | 0 | 1ms |
| **Merge Sort** | 104,727 | 212,906 | 4ms |
| **Heap Sort** | 698,825 | 349,413 | 9ms |
| **Radix Sort** | 15,285 | 152,865 | 4ms |
|  |  |  |  |
| **Experimental Results** | **ArraySize** |  |  |
| **List Property: ReverseOrder** | **Comparisons** | **Movements** | **Total Time** |
| **Insertion Sort** | 116,823,255 | 116, 823,255 | 168ms |
| **Selection Sort** | 116,838,540 | 58,423,092 | 77ms |
| **Quick Sort** | 228,192 | 7,643 | 1ms |
| **Merge Sort** | 108,179 | 212,906 | 2ms |
| **Heap Sort** | 540,905 | 172,710 | 6ms |
| **Radix Sort** | 15,285 | 152,865 | 3ms |
|  |  |  |  |
| **Experimental Results** | **ArraySize** |  |  |
| **List Property: AlmostOrder** | **Comparisons** | **Movements** | **Total Time** |
| **Insertion Sort** | 17,315,712 | 17,315,712 | 13ms |
| **Selection Sort** | 116,838,540 | 124,257 | 77ms |
| **Quick Sort** | 395,011 | 41,940 | 4ms |
| **Merge Sort** | 130,366 | 212,906 | 4ms |
| **Heap Sort** | 563,945 | 206,160 | 6ms |
| **Radix Sort** | 15,285 | 152,865 | 1ms |
|  |  |  |  |
| **Experimental Results** | **ArraySize** |  |  |
| **List Property: RandomOrder** | **Comparisons** | **Movements** | **Total Time** |
| **Insertion Sort** | 58,237,365 | 58,237,365 | 37ms |
| **Selection Sort** | 116,838,540 | 138,649 | 77ms |
| **Quick Sort** | 309,787 | 51,796 | 5ms |
| **Merge Sort** | 193,345 | 212,906 | 5ms |
| **Heap Sort** | 558,734 | 191,665 | 8ms |
| **Radix Sort** | 15,285 | 152,865 | 3ms |

# Observation

From the above results, we can see that Insertion Sort works the best when the array is already sorted but it performs the worse when the array is sorted in reverse direction. It has a high execution time, high number of comparisons and high number of movements.

Selection Sort generally has the most amount of comparisons since it has a nested for loop which compares each element with other elements irrespective of array type.

Merge Sort and Quick Sort almost have same execution time as both use divide and conquer approach but the comparisons and movements of both are not same, Merge sort has less number of comparisons whereas Quick sort has less number of movements in all cases.

From all of the above sorting techniques, Radix Sort is the best performing algorithm, it has 15,285 comparisons each time and 152,865 movements irrespective of the type of array and it takes the lowest execution time in most cases.