**Module 24**

**Problem**

**Nonuniform data**:

Write a client that generates test data that is not uniform, including the following:

**Case 1:** Half the data is 0s, half 1s.

**Case 2:** Half the data is 0s, half the remainder is 1s, half the remainder is 2s, and so forth.

**Case 3:** Half the data is 0s, half random int values. Develop and test hypotheses about the effect of such input on the performance of the algorithms in this section.

**Approach:**

InsertionSort.java ,Selection.java generates the output for the Cases.

**Analysis:**

**Selection Sort:**

|  |  |  |
| --- | --- | --- |
| **Cases** | **output** | **Output** |
| **Case 1** | **0.017** | **0.033** |
| **Case 2** | **0.019** | **0.035** |
| **Case 3** | **0.022** | **0.039** |

**Insertion Sort :**

|  |  |  |
| --- | --- | --- |
| **Cases** | **output** | **Output** |
| **Case 1** | **0.013** | **0.031** |
| **Case 2** | **0.016** | **0.033** |
| **Case 3** | **0.019** | **0.037** |

**Observation:**

Selection sort must scan the remaining parts of the array when placing an element, insertion sort only scans as many elements as necessary. That means that when the array is already sorted or almost sorted, insertion sort performs in O(n) time.

Therefore insertion sort took lower run time than the selection sort for the sorted arrays.

And to mention insertion sort is a stable sort whereas the selection sort is not a stable sort.