CSCI 311 Data structures and Algorithms

Lab 2 Assignment

Project Design : The project extends the functionality of previous assignment lab1, by adding functionalities to already existing class WordCount.

The class includes the following additional functions for sorting the map output:-

- **insertionSortByValue** : Takes vector pair as its input and applies insertion sort on vector pair and sorts it by values.

- **InsertionSortByKey** : Takes vector pair as its input and applies insertion sort on vector pair and sorts it by keys.

TIMING :- Insertion sort performs best when the input is already sorted and is worst when the input is reverse sorted. Comparing the given inputs, insertion sort takes longest time for inp4.txt as compared to other sorting methods.

- **maxHeapify** : Builds max heap.
- **HeapSort** : Takes vector pair as its input and applies heap sort on vector pair and sorts it by keys.

- **maxHeapifyValue** : Maintains max heap property.
- **BuildMaxHeapValue** : Bulids max heap.
- **HeapSortValue** : Takes vector pair as its input and applies heap sort on vector pair and sorts it by values.

  TIMING :Heap sort functions on an average run time of O(nlogn).

- **quickSortByValue** : Takes vector pair as its input and applies quick sort on vector pair and sorts it by values.

- **Partition** : Returns the pivot element.

- **quickSortByKey** : Takes vector pair as its input and applies quick sort on vector pair and sorts it by keys.
- **PartitionKey** : Returns partition element.

  TIMING: Among all the sorting algorithms , quick sort is the fastest. Best case run time is theta(nlogn). Quick sort is the slowest when all the elements are equal. But as per the input files given, the inputs are evenly spaced out and quick sort gives best performance. In my assignment I am selecting the pivot as the last element, but if I decide to select random pivot, the run time decreases considerably. Thus randomized quick sort has better run time as compared to fixed pivot.

- **MergeSort** : Takes vector pair as its input and applies merge sort on vector pair and sorts it by values.
- **MergeKey** :- Merges the partitioned subarrays.
- **MergeSortByKey** : Takes vector pair as its input and applies quick sort on vector pair and sorts it by values.
- **Merge** :Merges the partitioned subarrays.

  TIMING : Merge sort performs fairly well as compared to insertion sort but not as good as quick sort.

- **print** :- Function to print the result vector.

  PERFORMANCE BASED ON SIZE OF INPUT : In general the run time of each sorting algorithm increases with the increase in size of inputs. More number of inputs take more time than lesser number of inputs. Overall, insertion sort performs the worst among all the sorts. Other sorts like quick, merge and heap perform fairly well for the inputs.