**ITCS 6114 - Algorithms & Data Structures**

**Project 1: Comparison-based Sorting Algorithms**

**-**Ajinkya Gadgil (801200445), Saumitra Apte (801202857)

**Project Overview**

This project includes implementation, comparison and analysis of different sorting algorithms for different input sizes and comparing the time required for the execution of the sorting algorithms for 3 cases.

1. Input is already sorted (Best Case)
2. Input is sorted reversely (Worst Case)
3. Average case:
   1. In average case we shuffle the input array and execute the sorting algorithms for ‘number\_of\_repetitions’ by default we have used ‘number\_of\_repetitions’ is 3. So, for average case for each algorithm, for each input size runs for 3 times and we take the average of all 3 times and that is how we get the average time.

Different sorting algorithms used for comparison are as below:

1)Insertion sort

2) Heap Sort

3) Merge Sort

4) In place Quick sort

5) Modified Quick sort

All the algorithms have been implemented in Python.

Data Structures used in the project are as below:

1. **Dictionary:**
   1. We have used dictionary for storing the key value pairs as {key:value}. Where key is the input size and value are the array generated for that input size. We have used random module of python to generate random numbers. We have used **random.sample(sequence,k)** which creates an array in a given sequence and k number of items in the array.
2. **List**: We have used list mainly for:
   1. We store input sizes in a list on which all the algorithms would run.
   2. We have used list as a value field in dictionary which stores the randomly generated array for the given input sizes.
   3. To store execution time for all 3 cases.

**Execution Flow Understanding**

1. We have created separate files for each sorting algorithm and created a file named **“sorting\_algorithm\_main.py”** which imports all the algorithm classes and we create object of each class and call the respective main methods of each sorting algorithms.
2. We are randomly generating the inputs using python random module for each input sizes and store it in the dictionary such as {10: [*random list of 10 elements*], 1000: [*random list of 1000 elements*]…..60000:[*random list of 1000 elements*]}
3. For each algorithm we pass three parameters –

**input\_dictionary**-which has random input arrays as per size

**input\_sizes** -which has input sizes on which we are running algorithms **no\_of\_repetitions**- no of times we want to run the algorithm for each input size to calculate average case execution time

1. Inside the algorithm, there are three cases for average, already sorted and reverse sorted array. In average case, as per given no of repetitions, array is shuffled and sorted and average is calculated.
2. As the array is sorted in average case same array is used for best case i.e. when the array is sorted
3. Array is reverse sorted and passed to the sorting algorithm to analyse the worst case.

**Input**: User needs to choose sorting algorithm to run from choices corresponding to each sorting algorithm. Also, there is a choice to run all algorithms at the same time as well.

**Output**: There will be 3 lists for corresponding algorithm in the output which will show execution times for average, already sorted and reverse sorted case corresponding to the input sizes.

**Complexity Analysis, Comparison and Graphs**

**Average case for all algorithms:**

|  |  |
| --- | --- |
| Insertion Sort | **Θ**(n^2) |
| Merge Sort | **Θ**(nlogn) |
| Heap Sort | **Θ**(nlogn) |
| In place Quick Sort | **Θ**(nlogn) |
| Modified Quick sort | **Θ**(nlogn) |

**Time Comparisons (in ms):**

Table

Description automatically generated

**Graph:**

Chart, line chart

Description automatically generated

**Already Sorted Case (Best Case):**

|  |  |
| --- | --- |
| Insertion Sort | Ω (n) |
| Merge Sort | Ω (nlogn) |
| Heap Sort | Ω (nlogn) |
| In place Quick Sort | Ω (nlogn) |
| Modified Quick sort | Ω (nlogn) |

**Time Comparisons (in ms):**

Table

Description automatically generated

**Graph:**

Chart, line chart

Description automatically generated

**Reverse Sorted Case (Worst Case)**

|  |  |
| --- | --- |
| Insertion Sort | O (n) |
| Merge Sort | O (n logn) |
| Heap Sort | O (n logn) |
| In place Quick Sort | O (n logn) |
| Modified Quick sort | O (n logn) |

**Time Comparisons (in ms):**

Table

Description automatically generated

**Graph:**

Chart, line chart

Description automatically generated

**Individual Sorting Algorithms for different cases Graphs**

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated