# Experiment 6

WAP to perform comparative analysis of iterative soring algorithm (Quick , Heap Quick).



Fig: 1 (Quick Sort)

****

Fig: 2 (Heap Quick Sort)

**Result Analysis and Discussion**

## This experiment is conducted using following specifications. The algorithm is implemented using C language (clang-1400.0.29.202). During this test all the apps were closed to improve the results of the experiment.

## 

Fig: 4 (PC used in experiment)

In this experiment the comparative analysis of iterative sort algorithms has been implemented and executed for different value of n. During this experiment for different value of n the time taken by the algorithm has been measured and tabulated as shown in table below.

|  |  |  |
| --- | --- | --- |
| Arr size (in lakhs) | Time sec  (Quick sort) | Time sec  (Heap sort) |
| 5 | 0.089221 | 0.173695 |
| 10 | 0.185897 | 0.384221 |
| 15 | 0.283526 | 0.629683 |
| 20 | 0.389866 | 0.865206 |

**Comparative Analysis:**

In this comparative analysis of Quicksort and Heapsort algorithms for sorting large datasets, I observed the following key findings:

**Performance Advantage of Quicksort:**

Quicksort consistently outperformed Heapsort in terms of execution time for all tested array sizes (5 lakhs to 20 lakhs). Quicksort demonstrated a clear speed advantage over Heapsort.

**Efficiency with Increasing Array Size:**

Both Quicksort and Heapsort exhibited increased execution times as the array size increased. This behavior aligns with their expected time complexity. Quicksort's execution times scaled efficiently, roughly doubling as the array size doubled. Heapsort's execution times were notably slower than Quicksort, and the doubling pattern was observed.

**Practical Implications:**

Based on the provided data and observations, if speed is a crucial consideration, Quicksort is the recommended choice for sorting large datasets. Quicksort consistently provided faster execution times compared to Heapsort for the given dataset and experimental conditions.

**Algorithm Selection:**

The choice of sorting algorithm should take into account various factors, including the specific characteristics of the dataset, worst-case performance considerations, and implementation details. While Quicksort demonstrated superior performance in this experiment, other scenarios may favor Heapsort due to its guaranteed worst-case time complexity of O(n log n).

In conclusion, this comparative analysis indicates that for sorting large datasets in the tested conditions, Quicksort is the preferred algorithm when speed is a primary concern, as it consistently delivered faster execution times than Heapsort across varying array sizes. However, selecting the most suitable sorting algorithm should always consider the unique requirements and constraints of the given problem or application.