Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department.
AOA / Even Sem 2021-22 / Experiment 2.

YASH SARANG.
47 / D6AD.
EXPERIMENT - 2.

Aim: Merge Sort and Quick Sort.

Theory:

MERGE SORT

Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The merge() function is used for merging two halves. The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

Time Complexity: Sorting arrays on different machines.
Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation:
$T(n) = 2T(n/2) + \theta(n)$

The above recurrence can be solved either using the Recurrence Tree method or the Master method. It falls in case II of the Master Method and

the solution of the recurrence is θ(nLogn). The time complexity of Merge Sort is θ(nLogn) in all 3 cases (worst, average and best) as merge sort always divides the array into two halves and takes linear time to merge two halves.

Auxiliary Space: O(n)

Drawbacks of Merge Sort:
1. Slower comparative to the other sort algorithms for smaller tasks.
2. The merge sort algorithm requires an additional memory space of 0(n) for the temporary array.
3. It goes through the whole process even if the array is sorted.

QUICKSORT

Like Merge Sort, QuickSort is a Divide and Conquer algorithm.
It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of QuickSort that pick pivot in different ways.
1. Always pick the first element as pivot.
2. Always pick the last element as the pivot.
3. Pick a random element as a pivot.
4. Pick median as the pivot.

The key process in quickSort is partition(). The target of partitions is, given an array and an element x of the array as a pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

Time Complexity: Average : O(nLogn)
from T(n) = T(n/9) + T(9n/10) + Theta(n)

Worst case : O(n^2)

Auxiliary Space: O(1)

CODE:

Code is in the Final.c file attached along with this doc.

OUTPUT:

	Selection Sort	Insertion Sort	Selection Sort(2 Way)	Merge Sort	Quick Sort
numbers_1.dat	11.182000	10.889000	7.477000	0.013000	0.063000
numbers_2.dat	11.291000	10.723000	7.441000	0.013000	0.054000
numbers_3.dat	11.204000	10.733000	7.418000	0.013000	0.057000
numbers_4.dat	11.272000	10.723000	7.449000	0.014000	0.061000
numbers_5.dat	11.311000	10.806000	7.410000	0.013000	0.060000

CONCLUSION:

By performing this experiment, I can conclude that Although the worst-case time complexity of QuickSort is O(n^2) which is more than all other sorting algorithms, QuickSort is faster in practice, because its inner loop can be efficiently implemented on most architectures and in most real-world data. QuickSort can be implemented in different ways by changing the choice of the pivot so that the worst-case rarely occurs for a given type of data.

However, merge sort is generally considered better when data is huge and stored in external storage. Due to the fact that they have a large number of

resources to prioritize the time efficiency over storage and computation amount.
