Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

SE – COMP (SE-A) **Sub- DAA Lab**

Name	**Akshat Biniwale**
UID No.	**2021300014**
Subject	**Design and Analysis of Algorithms**
Class	**Comps A**
Experiment No.	**3**
AIM	Experiment on finding the running time of merge sort and insertion sort algorithm.

THEORY:

1. Merge Sort: Merge sort is a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array. In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

2. Quick sort: Like Merge Sort, Quick Sort 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 Quick Sort that pick pivot in different ways.

    a. Always pick the first element as a pivot.
    b. Always pick the last element as a pivot (implemented below)
    c. Pick a random element as a pivot.
    d. Pick median as the pivot.

The key process in Quick Sort is a partition(). The target of partitions is, given an array and an element x of an array as the 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.

ALGORITHM:

1. Merge Sort:

```
step 1: start
step 2: declare array and left, right, mid variable
step 3: perform merge function.
    if left > right
        return
    mid= (left+right)/2
    mergesort(array, left, mid)
    mergesort(array, mid+1, right)
    merge(array, left, mid, right)

step 4: Stop
```

2. Quick Sort:

   Partition Algorithm:

```
PARTITION (array A, start, end){
        pivot ? A[end]
        i ? start-1
        for j ? start to end -1 {
                do if (A[j] < pivot) {
                        then i ? i + 1
                        swap A[i] with A[j]
                }
        }
        swap A[i+1] with A[end]
        return i+1
}
```

Quick Sort Algorithm:

```
QUICKSORT (array A, start, end){
    if (start < end){
        p = partition(A, start, end)
        QUICKSORT (A, start, p - 1)
        QUICKSORT (A, p + 1, end)
    }
}
```

PROGRAM:

1. Quick Sort:

```
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

```
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = partition(arr, low, high);
        quickSort(arr, low, pivot - 1);
        quickSort(arr, pivot + 1, high);
    }
}
```

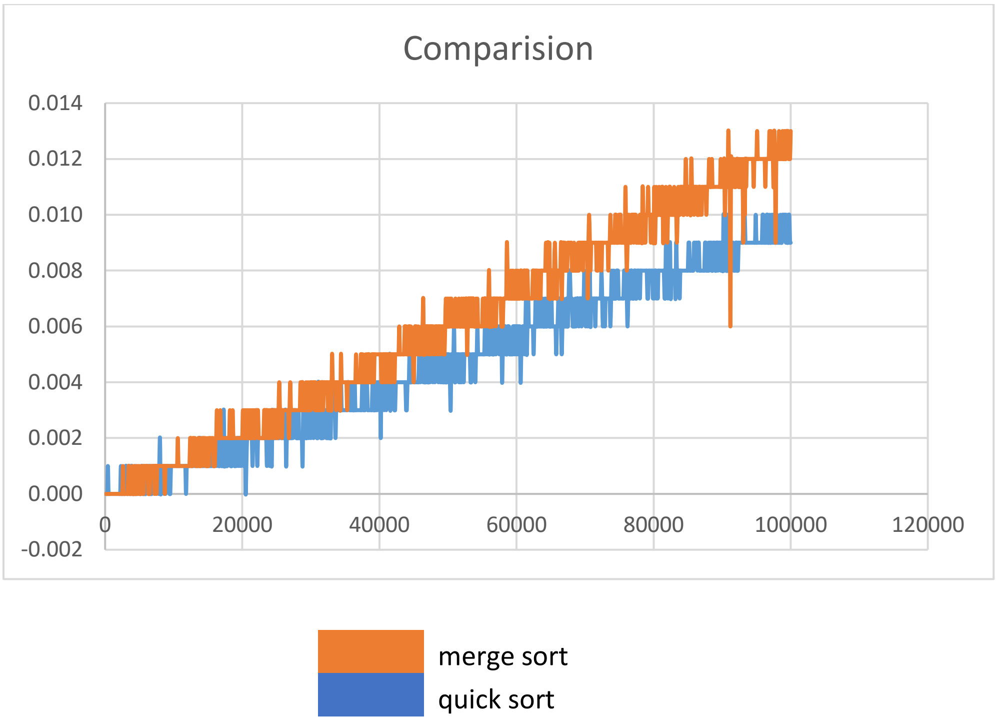
2. Merge Sort:

```
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
```

ANALYTICAL RESULT:

Quick sort has an average time complexity of O(n log n) and a worst case time complexity of O(n^2). On the other hand, merge sort has a time complexity of O(n log n) for both average and worst cases. Both quick sort and merge sort are efficient sorting algorithms, but quick sort is faster in average cases while merge sort is more consistent in its performance.


Comparision
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
-0.002
0
20000
40000
60000
80000
100000
120000
merge sort
quick sort