DIGITAL ANALYSIS AND ALGORITHM

EXPERIMENT - 02

NAME :- MANTHAN AYALWAR

UID :- 2021700003

BATCH :- D1

Aim :- Experiment based on divide and conquers approach.

Code :-

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
// declaring function for insertion sort based on length
void merge(int arr[], int l, int m, int r)
{
	int n_arr[r - l + 2];
	int i = l;
	int j = m + 1;
	int k = 0;
	while (k < r - l + 1)
	{
		if (arr[i] < arr[j])
		{
			n_arr[k] = arr[i];
			i += 1;
			k += 1;
			if (i > m)
			{
```

```
                while (k < r - l + 1)
                {
                        n_arr[k] = arr[j];
                        j += 1;
                        k += 1;
                }
                break;
        }
    }
    else if (arr[i] >= arr[j])
    {
        n_arr[k] = arr[j];
        j += 1;
        k += 1;
        if (j > r)
        {
                while (k < r - l + 1)
                {
                        n_arr[k] = arr[i];
                        i += 1;
                        k += 1;
                }
                break;
        }
    }
}

int p = 0;

// copy all elements to og array
for (int a = l; a < r + 1; a += 1)
```

```
        {
                arr[a] = n_arr[p];
                p += 1;
        }
}

// declaring function for merge sort based on length
void merge_sort(int arr[], int l, int r)
{
        if (l < r)
        {
                int m = l + ((r - l) / 2);
                merge_sort(arr, l, m);
                merge_sort(arr, m + 1, r);
                merge(arr, l, m, r);
        }
}

void quick_sort(int arr[], int l, int r)
{
        if (l >= r)
        {
                return;
        }

        int pivot = arr[l];
        int j = l + 1;
        for (int i = l; i < r + 1; i += 1)
        {
                if (arr[i] < pivot)
                {
```

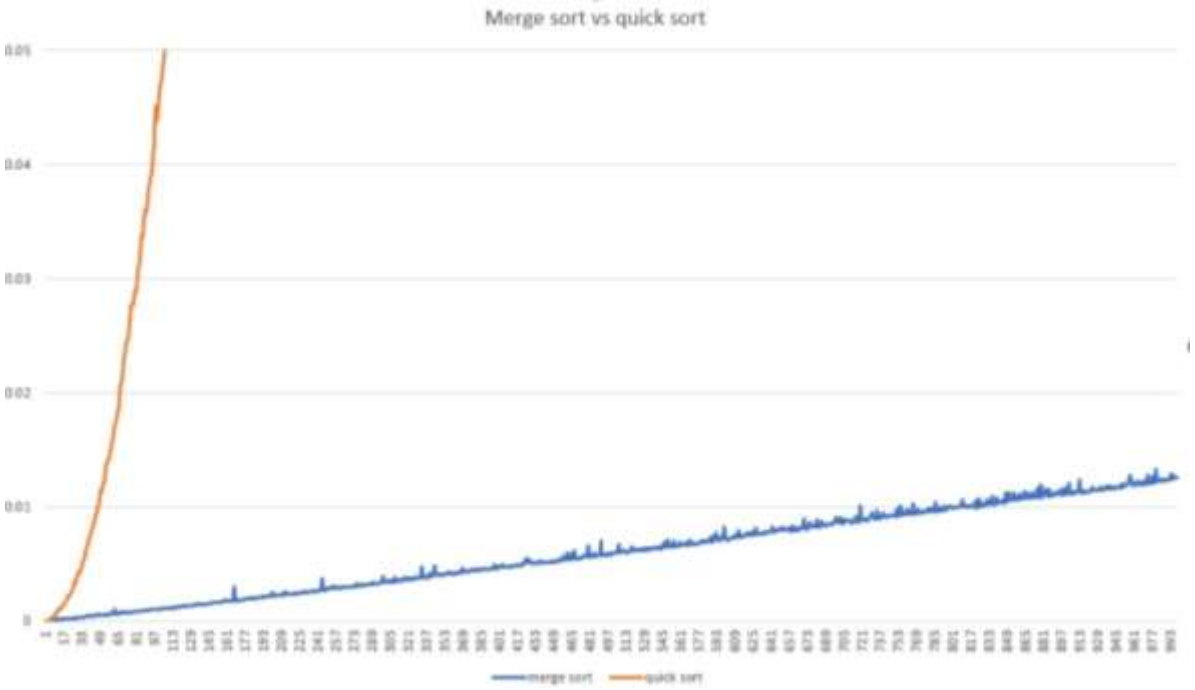
```
                int temp = arr[j];
                arr[j] = arr[i];
                arr[i] = temp;
                j += 1;
            }
        }
        for (int k = l + 1; k < j; k += 1)
        {
            arr[k - 1] = arr[k];
        }
        arr[j - 1] = pivot;
        quick_sort(arr, l, j - 2);
        quick_sort(arr, j, r);
}

int main()
{
        // opening file to store input
        FILE *fp = fopen("input.txt", "w");
        if (fp == NULL)
        {
            printf("Error opening the file");
            return -1;
        }

        // inputting 1 lakh random ints to input.txt
        for (int i = 0; i < 100000; i += 1)
        {
            fprintf(fp, "%d ", rand());
        }
        fclose(fp);
```

```
// opening file to store output
FILE *fop = fopen("output.txt", "w");
if (fop == NULL)
{
	printf("Error opening the file");
	return -1;
}
// outputting code starts
int b = 1;
for (int j = 100; j < 100000; j += 100)
{
	int arrs[j];
	FILE *fir = fopen("input.txt", "r");
	if (fir == NULL)
	{
		printf("Error opening the file");
		return -1;
	}

	for (int k = 0; k < j; k += 1)
	{
		fscanf(fir, "%d ", &arrs[k]);
	}
	double t_mergesort = 0.0;
	double t_quicksort = 0.0;
	clock_t begin = clock();

	merge_sort(arrs, 0, j);
	clock_t end = clock();
	t_mergesort += (double)(end - begin) / CLOCKS_PER_SEC;
```

```
            begin = clock();
            quick_sort(arrs, 0, j);
            end = clock();

            t_quicksort += (double)(end - begin) / CLOCKS_PER_SEC;
            fprintf(fop, "%d\t%f\t%f\n", b, t_mergesort, t_quicksort);
            printf("%d\t%f\t%f\n", b, t_mergesort, t_quicksort);
            b += 1;
            fclose(fir);
        }
        fclose(fop);
}
```

PLOTTING THE DATA OBTAINED AFTER EXECUTION IN EXCEL:

Graph of running time of quick sort (Lomuto partitioning scheme) and merge sort vs input size:



Observation :-

Quick sort beats merge sort for almost all input sizes in time complexity

Number of comparison operations required by merge sort is exactly linearly proportional to the input size. When quick sort is implemented using Lomuto partitioning scheme, number of comparisons required is higher than merge sort; but if Hoare's partitioning scheme is used, fewer comparisons are required than merge sort. Merge sort graph has less slope than quick sort graph .

Conclusion :- From this experiment I learned that Quick sort performs better than merge sort for randomised input of size smaller