NAME:	**Sanskar Kamble**
UID:	**2021300054**
SUBJECT	**Design and Analysis of Algorithm**
EXPERIMENT NO :	**02**
DATE OF PERFORMANCE	**13/02/2023**
DATE OF SUBMISSION	**19/02/2023**
AIM:	**To Sort 1,00,000 randomly generated numbers using Merge sort and Quick sort.**
THEORY:	**MergeSort -** 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. **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. ● Always pick the first element as a pivot. ● Always pick the last element as a pivot (implemented below).Pick a random element as a pivot. ● Pick the median as the pivot.The key process in quicksort 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.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

const int limit = 100000;
const int block = 100;

void merge (long arr[],long l,long m,long r) {

    long i = 0, j = 0, k = l;
    long n1 = m - l + 1;
    long n2 = r - m;
    long 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];
    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 (long arr[],long l,long r) {
    if (l<r) {
        long m = l+(r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void merge_sort (FILE *f) {
        FILE *fp;
        fp = fopen("daa_2_merge_sort.txt", "w");
        fprintf(fp,"Block Size\tTime Taken\n");
        int size = 0;
        for (int times = 0; times<limit/block; times++) {
                size+=block;
                long arr [size];
                for (int i = 0; i<size; ++i)
                        fscanf(f,"%d",&arr[i]);
                // now our array is ready, we perform merge sort
                clock_t t;
                t = clock();
                mergeSort(arr, 0, size-1);
                t = clock()-t;
                double                    time_taken                    =
((double)t)/CLOCKS_PER_SEC;
                // storing the result in a file
                fprintf(fp,"%d\t%lf\n",size,time_taken);
        }
        fclose(fp);
}

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

```
   arr[i+1] = temp;
   return i+1;
}

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

void quick_sort (FILE *f) {
        FILE *fp;
        fp = fopen("daa_2_quick_sort.txt", "w");
        fprintf(fp,"Block Size\tTime Taken\n");
        int size = 0;
        for (int times = 0; times<limit/block; times++) {
                size+=block;
                long arr [size];
                for (int i = 0; i<size; ++i)
                        fscanf(f,"%d",&arr[i]);
                // now our array is ready, we perform quick sort
                clock_t t;
                t = clock();
                quickSort(arr, 0, size-1);
                t = clock()-t;
                double                    time_taken                    =
((double)t)/CLOCKS_PER_SEC;
                // storing the result in a file
                fprintf(fp,"%d\t%lf\n",size,time_taken);
        }
        fclose(fp);
}

int main () {


        // generating 1,00,000 integers and storing them in a file
        FILE *f;
        f = fopen("daa_2_random_integers.txt", "w");
        for (int i = 0; i<limit; ++i)
                fprintf(f,"%d\n",rand());
```

```
    // merge sort
    merge_sort(f);

    // quick sort
    quick_sort(f);

    fclose(f);

    return 0;
}
```

CONCLUSION:	By performing the above experiment I have successfully understood about the Time complexity of Merge sort and Quick sort.