Name	Nishant Dinesh Satere
UID no.	202271009
Experiment No.	2

AIM:	Experiment based on divide and conquer approach.
Program 2	
PROBLEM STATEMET :	For this experiment, you need to implement two sorting algorithms namely Quicksort and Merge sort methods

PROGRAM:	Exp2.c > partion(int [], int, int)
	#include<stdio.h>
	#include<math.h>
	#include<stdlib.h>
	#include<time.h>
	
	void swap(int*x, int *y){
	int temp = *x;
	*x = *y;
	*y = temp;
	}
	
	void printArray(int *arr, int n){
	for(int i = 0; i < n; i++){
	printf("%d\n", arr[i]);
	}
	}
	
	//Quick Sort
	int partion(int arr[], int lb, int ub){
	int piovt = arr[lb];
	int start = lb;
	int end = ub;

Exp2.c > partion(int [], int, int)

```c
    while(start < end){
        while(arr[start] <= piovt){
            start++;
        }
        while(arr[end] > piovt){
            end--;
        }
        if(start < end){
            swap(&arr[start], &arr[end]);
        }
    }
    swap(&arr[lb],&arr[end]);
    return end;
}

void quickSort(int arr[], int lb, int ub){
    if(lb<ub){
        int loc = partion(arr, lb,ub);
        quickSort(arr,lb,loc-1);
        quickSort(arr,loc+1,ub);
    }
}

//Merge Sort
void merege(int arr[], int lb, int mid, int ub){
    int i = lb;
    int j = mid+1;
    int k = lb;
    int arrn[100000] = {};
    while(i <= mid && j <= ub){
        if(arr[i] <= arr[j]){
            arrn[k] = arr[i];
            i++;
        }
        else{
            arrn[k] = arr[j];
            j++;
```

Exp2.c > partion(int [], int, int)

```c
        }
        k++;
    }
    if(i > mid){
        while(j <= ub){
            arrn[k] = arr[j];
            j++;
            k++;
        }
    }else{
        while(i <= mid){
            arrn[k] = arr[i];
            i++;
            k++;
        }
    }
}

void meregeSort(int arr[], int lb, int ub){
    if(lb<ub){
        int mid = (lb+ub)/2;
        meregeSort(arr, lb, mid);
        meregeSort(arr, mid+1, ub);
        merege(arr, lb , mid, ub);
    }
}

int main(){
    printf("\n");
    char *randNumders = "rNumber.txt";
    char *quicksort = "quick.txt";
    char *meregesort = "merge.txt";

    FILE *randnumfptr = fopen(randNumders, "w");
    FILE *sortednumfptr;

    int n = 100000;
```

Exp2.c > partion(int [], int, int)

```c
    int a[n];

    int num, startTime, endTime, rangeTill;

    //Created random array
    printf("Generating random array");
    for(int i = 0; i < n; i++){
        fprintf(randnumfptr, "%d\n", rand());
    }
    fclose(randnumfptr);

    //Quick sort
    randnumfptr = fopen(randNumders, "r");
    printf("\nReading random array");
    for(int i = 0; i < n; i++){
        fscanf(randnumfptr, "%d" , &a[i]);
    }
    fclose(randnumfptr);
    printf("\nDone\n");
    rangeTill = 100;

    printf("QuickSort\n");
    sortednumfptr = fopen(quicksort, "w");
    printf("Sorted numbers stored in (%s)",quicksort);

    while(rangeTill <= n){
        startTime = clock();
        quickSort(a, 0,rangeTill);
        endTime = clock();
        fprintf(sortednumfptr, "%d\n",endTime-startTime);
        rangeTill += 100;
    }
    fclose(sortednumfptr);
    printf("\nQuick sort done");


    //Merge Sort
```

Exp2.c > partion(int [], int, int)

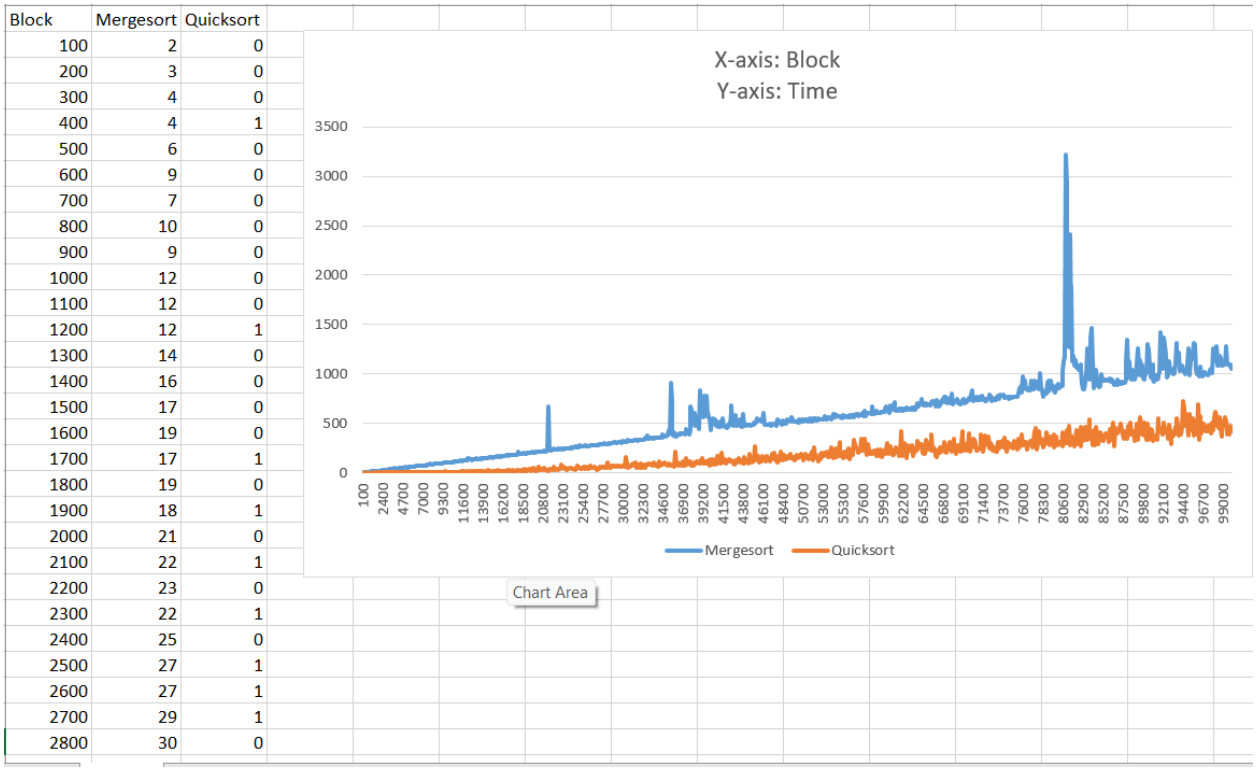
```c
    //Merge Sort
    randnumfptr = fopen(randNumders, "r");
    printf("\nReading random array");
    for(int i = 0; i < n; i++){
        fscanf(randnumfptr, "%d", &a[i]);
    }
    fclose(randnumfptr);

    printf("\nDone\n");
    rangeTill = 100;

    printf("mergesort\n");
    sortednumfptr = fopen(meregesort, "w");
    printf("Sorted numbers stored in (%s)",meregesort);

    while(rangeTill <= n){
        startTime = clock();
        mergeSort(a , 0 , rangeTill);
        endTime = clock();
        fprintf(sortednumfptr, "%d\n", endTime-startTime);
        rangeTill += 100;
    }
    fclose(sortednumfptr);
    printf("\nmerge sort done");
    return 0;
}

```

GRAPH:	
CONCLUSION:	In this experiment, we calculated & analyzed the time consumed by quick & merge sorting algorithms

Block	Mergesort	Quicksort
100	2	0
200	3	0
300	4	0
400	4	1
500	6	0
600	9	0
700	7	0
800	10	0
900	9	0
1000	12	0
1100	12	0
1200	12	1
1300	14	0
1400	16	0
1500	17	0
1600	19	0
1700	17	1
1800	19	0
1900	18	1
2000	21	0
2100	22	1
2200	23	0
2300	22	1
2400	25	0
2500	27	1
2600	27	1
2700	29	1
2800	30	0