Name	SMRUTI SONEKAR
UID no.	2021700064
Experiment No.	2

AIM:	**To implement the sorting functions e.g. Merge Sort and Quick Sort**
Program 1	
PROBLEM STATEMENT :	MERGE SORT
ALGORITHM/ THEORY:	Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithms. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the **merge()** function to perform the merging. The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list. • Best Case Complexity: Order of n*log(n)/O(n*log n). • Average Case Complexity: Order of n*log(n)/O(n*logn). • Worst Case Complexity: Order of n*log(n)/O(n*logn).

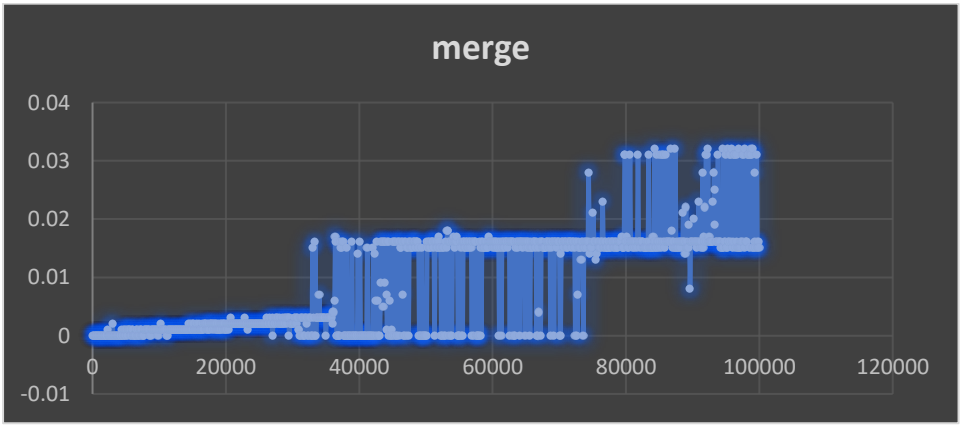
PROGRAM:	`#include<stdio.h>`
	`#include <conio.h>`
	`#include <stdlib.h>`
	`#include <time.h>`

```
void mergesort( int a[],int lb,int ub);
void mergeFunc(int a[], int lb,int mid,int ub);

void mergeFunc(int a[], int lb,int mid,int ub)
  {
      int i,j,k,n1,n2;

      n1=(mid-lb)+1;
      n2=ub-mid;
      int x[n1],y[n2];

      for(i=0;i<n1;i++)
      {
          x[i]=a[lb+i];
      }

      for(j=0;j<n2;j++)
      {
          y[j]=a[mid+1+j];
      }

      i=j=0;
      k=lb;

      while(i<n1 && j<n2)
      {
          if(x[i]<=y[j])
          {
              a[k]=x[i];
              i++;
          }
          else
          {
              a[k]=y[j];
              j++;
          }
          k++;
      }

      while(i<n1)
      {
          a[k]=x[i];
          i++;k++;
```

```
      }

        while(j<n2)
      {
          a[k]=y[j];
          j++;k++;
      }
  }



void mergesort( int a[],int lb,int ub)
  {
    if(lb < ub)
    {
      int mid= (ub+lb)/2 ;
      mergesort( a,lb,mid);
      mergesort(a,mid+1,ub);

      mergeFunc(a,lb,mid,ub);
    }
}



void display(int a[],int n)
{
  int i;
  printf("\nsorted array:\n");
  for(i=0;i<n;i++)
  {
     printf("%d ,",a[i]);
  }
}


int main()
 {
         int n,k=0;
    clock_t start,end;double time1;
   // printf("\nenter the no. of elements :");
    //scanf("%d",&n);

      FILE *fh;
      fh=fopen("C:\\Users\\Lenovo\\Desktop\\input.txt","r");
```

```
      if(fh==NULL)
      {
        printf("ERROR in file opening");
      }
          int block =1;
          n=100;
          while(block<=1000)
          {
               int a[n];
              for(k=0;k<n;k++)
              {
                fscanf(fh,"%d",&a[k]);
                //printf("%d ,",a[k]);
              }
                       start=clock();
                      mergesort(a,0,n-1);
                      //display(a,n);
                       end=clock();

                  time1= ((double)(end - start))/CLOCKS_PER_SEC;
                   printf("\n  %lf ",time1);
                   n=n+100;
                   block++;
          }

   fclose(fh);
}
```

GRAPH:



Program 2	
PROBLEM STATEMENT :	QUICK SORT.
ALGORITHM/ THEORY:	Quicksort is the widely used sorting algorithm that makes n log n comparisons in average case for sorting an array of n elements. It is a faster and highly efficient sorting algorithm. Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element. Conquer: Recursively, sort two subarrays with Quicksort. • Best Case Complexity: O(n*log(n)). • Average Case Complexity: O(n*log(n)). • Worst Case Complexity: O(N^2).

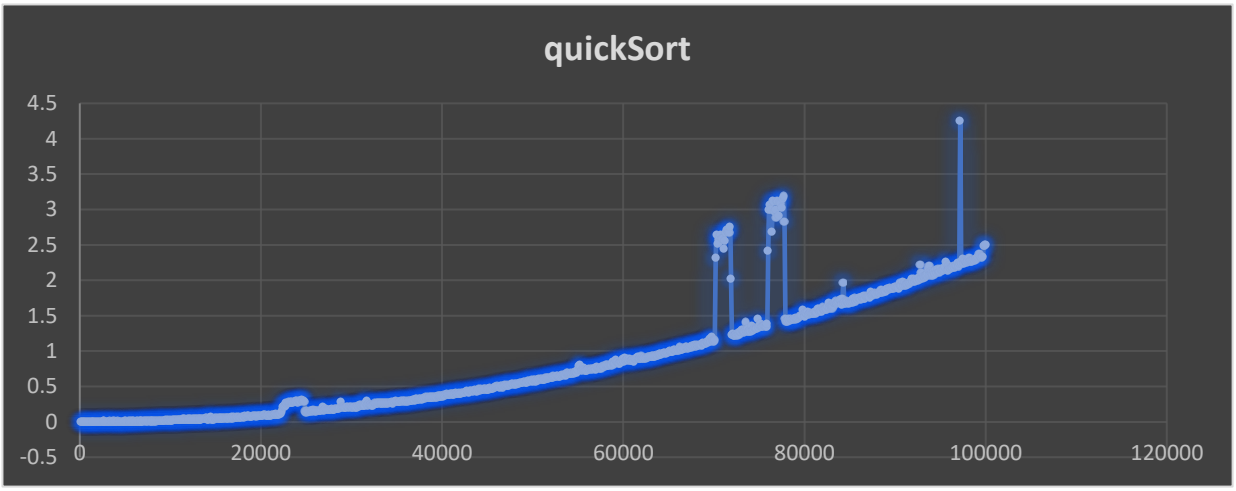
PROGRAM:	`#include <stdio.h>`
	`#include<stdlib.h>`
	`#include<time.h>`
	
	`void display(int a[],int n)`
	`{`
	`  int i;`

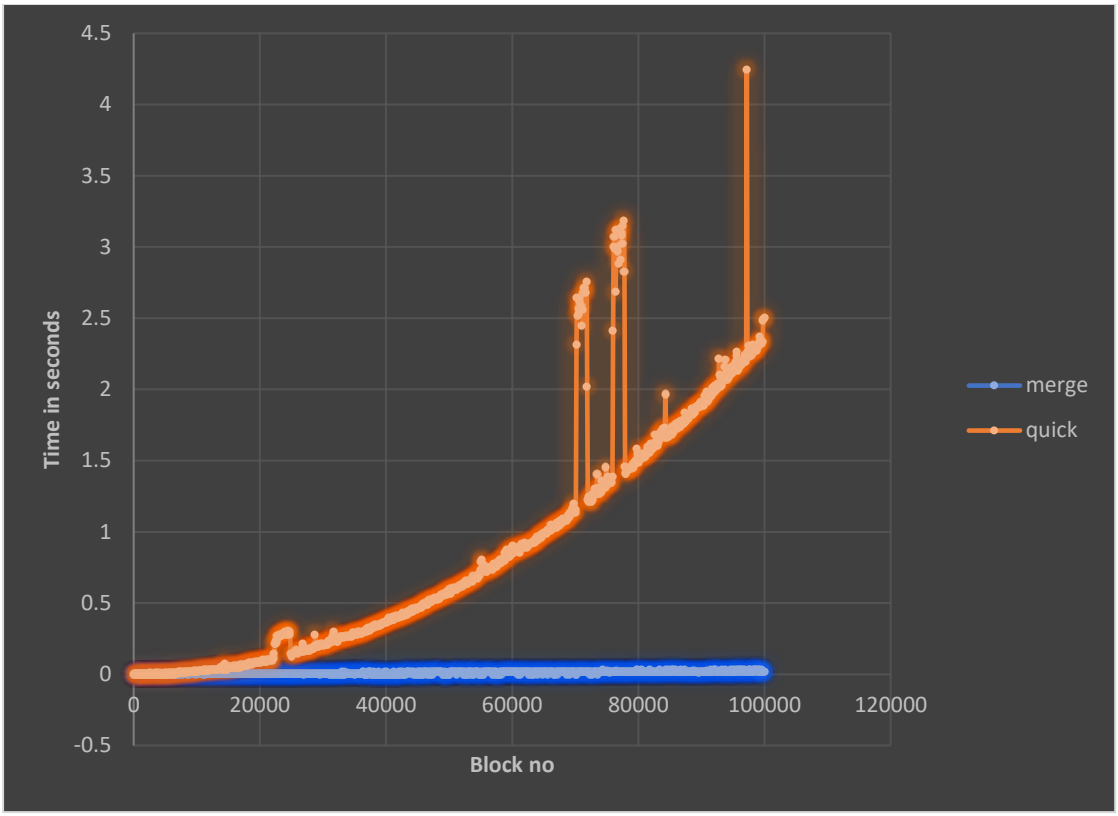
```
    printf("\nANS:\n");
    for(i=0;i<n;i++)
    {
        printf("%d ,",a[i]);
    }
}
void swap (int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

int part(int a[],int lb,int ub)
{
    int pivot=a[lb];
    int i,j;
    i=lb;j=ub;

    while(i<j)
    {
        while(a[i]<=pivot)
        {
            i++;
        }
        while(a[j]> pivot)
        {
            j--;
        }

        if(i<j)
        {
            swap(&a[i],&a[j]);
        }
    }

    swap(&a[lb],&a[j]);

    return j;

}
```

```c
void Quicksort(int a[],int lb, int ub)
{
    if(lb<ub)
    {
        int loc;
        loc=part(a,lb,ub);
        Quicksort(a,lb,loc-1);
        Quicksort(a,loc+1,ub);
    }
}

int main()
{
    printf("\nHello World\n");

   /*int a[] = { 11, 30, 24, 7, 31, 16, 39, 41 };
    int n = sizeof(a) / sizeof(a[0]); */

     int n,k=0;
    clock_t start,end;double time1;
   // printf("\nenter the no. of elements :");
    //scanf("%d",&n);

      FILE *fh;
      fh=fopen("C:\\Users\\Lenovo\\Desktop\\input.txt","r");
      if(fh==NULL)
      {
        printf("ERROR in file opening");
      }
          int block =1;
          n=100;
          while(block<=1000)
          {
               int a[n];
              for(k=0;k<n;k++)
              {
                fscanf(fh,"%d",&a[k]);
                //printf("%d ,",a[k]);
              }
                       start=clock();
                      Quicksort(a,0,n-1);
```

```
                    //display(a,n);
                     end=clock();

                time1= ((double)(end - start))/CLOCKS_PER_SEC;
                 printf("\n  %lf ",time1);
                 n=n+100;
                 block++;
          }

   fclose(fh);
}
```

RESULT/GRAPH:



RESULT GRAPH:	
CONCLUSION:	Merge Sort is better as worst case of merge sort is still O(n*log(n)) but of quick sort is O(N^2) which is worse in large number of datasets . AS in resultant graph we can see That curve of merge sort is less than quick sort in case of time taken to sort.