

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:	<pre> C Exp2.c > partion(int [], int, int) 1 #include<stdio.h> 2 #include<math.h> 3 #include<stdlib.h> 4 #include<time.h> 5 6 void swap(int*x, int *y){ 7 int temp = *x; 8 *x = *y; 9 *y = temp; 10 } 11 12 void printArray(int *arr, int n){ 13 for(int i = 0; i < n; i++){ 14 printf("%d\n", arr[i]); 15 } 16 } 17 18 //Quick Sort 19 int partion(int arr[], int lb, int ub){ 20 int piovt = arr[lb]; 21 int start = lb; 22 int end = ub; 23 </pre>
-----------------	---

```

C Exp2.c > partion(int [], int, int)
24     while(start < end){
25         while(arr[start] <= pivot){
26             start++;
27         }
28         while(arr[end] > pivot){
29             end--;
30         }
31         if(start < end){
32             swap(&arr[start], &arr[end]);
33         }
34     }
35     swap(&arr[lb], &arr[end]);
36     return end;
37 }
38
39 void quickSort(int arr[], int lb, int ub){
40     if(lb < ub){
41         int loc = partion(arr, lb, ub);
42         quickSort(arr, lb, loc-1);
43         quickSort(arr, loc+1, ub);
44     }
45 }
46
47 //Merge Sort
48 void merege(int arr[], int lb, int mid, int ub){
49     int i = lb;
50     int j = mid+1;
51     int k = lb;
52     int arrn[100000] = {};
53     while(i <= mid && j <= ub){
54         if(arr[i] <= arr[j]){
55             arrn[k] = arr[i];
56             i++;
57         }
58         else{
59             arrn[k] = arr[j];
60             j++;

```

C Exp2.c >  partition(int [], int, int)

```
61     }
62     k++;
63 }
64 if(i > mid){
65     while(j <= ub){
66         arrn[k] = arr[j];
67         j++;
68         k++;
69     }
70 }else{
71     while(i <= mid){
72         arrn[k] = arr[i];
73         i++;
74         k++;
75     }
76 }
77 }
78
79 void meregeSort(int arr[], int lb, int ub){
80     if(lb<ub){
81         int mid = (lb+ub)/2;
82         meregeSort(arr, lb, mid);
83         meregeSort(arr, mid+1, ub);
84         merege(arr, lb , mid, ub);
85     }
86 }
87
88 int main(){
89     printf("\n");
90     char *randNumders = "rNumber.txt";
91     char *quicksort = "quick.txt";
92     char *meregesort = "merge.txt";
93
94     FILE *randnumfptr = fopen(randNumders, "w");
95     FILE *sortednumfptr;
96
97     int n = 100000;
```

```

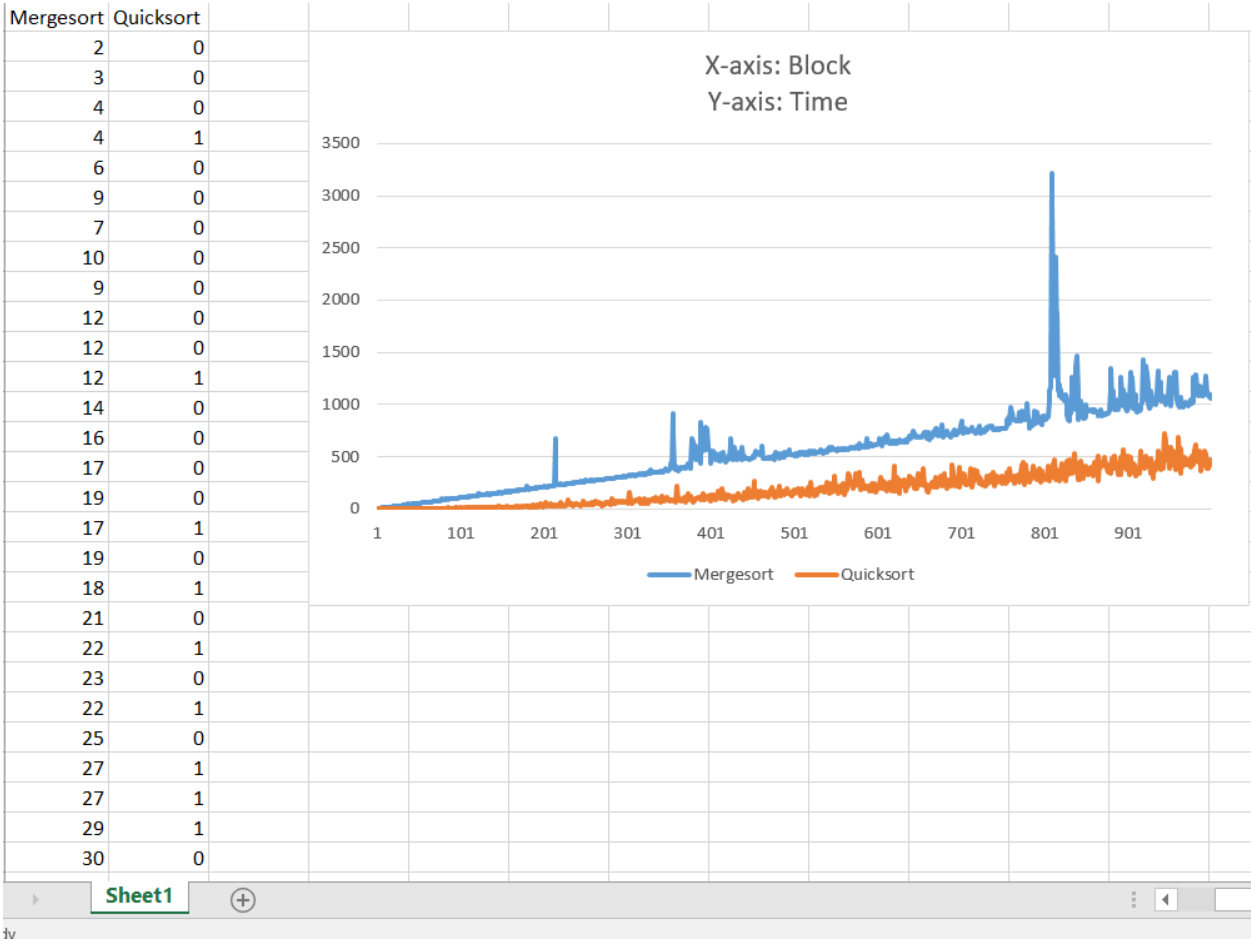
C Exp2.c > partion(int [], int, int)
98     int a[n];
99
100     int num, startTime, endTime, rangeTill;
101
102     //Created random array
103     printf("Generating random array");
104     for(int i = 0; i < n; i++){
105         fprintf(randnumfptr, "%d\n", rand());
106     }
107     fclose(randnumfptr);
108
109     //Quick sort
110     randnumfptr = fopen(randNumders, "r");
111     printf("\nReading random array");
112     for(int i = 0; i < n; i++){
113         fscanf(randnumfptr, "%d" , &a[i]);
114     }
115     fclose(randnumfptr);
116     printf("\nDone\n");
117     rangeTill = 100;
118
119     printf("QuickSort\n");
120     sortednumfptr = fopen(quicksort, "w");
121     printf("Sorted numbers stored in (%s)",quicksort);
122
123     while(rangeTill <= n){
124         startTime = clock();
125         quickSort(a, 0,rangeTill);
126         endTime = clock();
127         fprintf(sortednumfptr, "%d\n",endTime-startTime);
128         rangeTill += 100;
129     }
130     fclose(sortednumfptr);
131     printf("\nQuick sort done");
132
133
134     //Merge Sort

```

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

```
134     //Merge Sort
135     randnumfptr = fopen(randNumbers, "r");
136     printf("\nReading random array");
137     for(int i = 0; i < n; i++){
138         fscanf(randnumfptr, "%d", &a[i]);
139     }
140     fclose(randnumfptr);
141
142     printf("\nDone\n");
143     rangeTill = 100;
144
145     printf("mergesort\n");
146     sortednumfptr = fopen(meregesort, "w");
147     printf("Sorted numbers stored in (%s)", meregesort);
148
149     while(rangeTill <= n){
150         startTime = clock();
151         mergeSort(a , 0 , rangeTill);
152         endTime = clock();
153         fprintf(sortednumfptr, "%d\n", endTime-startTime);
154         rangeTill += 100;
155     }
156     fclose(sortednumfptr);
157     printf("\nmerge sort done");
158     return 0;
159 }
160
```

GRAPH:



CONCLUSION:

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