Aim:- Implementation of Menu driven Merge Sort and Quick Sort

Source Code:-

```
#include <stdio.h>
#include <stdlib.h>
void bubbleSort(int[], int);
void insertionSort(int[], int);
void quickSort(int[], int, int);
void mergeSort(int[], int, int);
int main() {
printf("D10A_Atharva Chavan_9\n\n");
printf("****Implementation of Menu driven Merge Sort and Quick Sort****\n");
int n, arr[100];
int i;
int op, cont;
do {
printf("\nEnter size of array: ");
scanf("%d", &n);
printf("Enter values of array:\n");
for(i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
printf("Enter sorting algorithm to use:\n1: Bubble\n2: Insertion\n3: Quick
sort\n4:Merge sort\n");
scanf("%d", &op);
switch(op) {
```

```
case 1:
bubbleSort(arr, n);
break;
case 2:
insertionSort(arr, n);
break;
case 3:
quickSort(arr, 0, n - 1);
break;
case 4:
mergeSort(arr, 0, n - 1);
break;
default:
printf("Invalid option!");
} for(i
=
0; i
<
n; i++) {
printf("%d ", arr[i]);
}
printf("\nContinue? 1/0:\t");
scanf("%d", &cont);
} while(cont == 1);
return 0;
}
void bubbleSort(int arr[], int size) {
int i, j, temp;
```

```
for(i = 0; i < size - 1; i++) {
for(j = 0; j < size - i - 1; j++) {
if (arr[j] > arr[j + 1]) {
//swap
temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
}
}
}
}
void insertionSort(int arr[], int size) {
int i, j, key;
for(i = 1; i < size; i++) {
key = arr[i];
for(j = i; j > 0 && arr[j - 1] > key; j--) {
arr[j] = arr[j - 1];
}
arr[j] = key;
}
}
int partition(int arr[], int offset, int size) {
int x = arr[size];
int i = offset - 1;
int j;
int temp;
for(j = offset; j < size; j++) {
if (arr[j] <= x) {
```

```
i++;
temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
}
} temp =
arr[i
+
1];
arr[i + 1] = arr[size];
arr[size] = temp;
return i + 1;
}
void quickSort(int arr[], int offset, int size) {
int pivot;
if (offset < size) {
pivot = partition(arr, offset, size);
quickSort(arr, offset, pivot - 1);
quickSort(arr, pivot + 1, size);
}
}
void merge(int arr[], int offset, int mid, int size) {
int n1 = mid - offset + 1;
int n2 = size - mid;
int* l = (int*)calloc(n1 + 1, sizeof(int));
int* r = (int*)calloc(n2 + 1, sizeof(int));
int i, j, k;
for(i = 0; i < n1; i++) {
```

```
*(l + i) = arr[offset + i];
} for(j
=
0; j
<
n2; j++) { *(r + j) = arr[mid + j + 1];
}
*(l + n1) = 32767;
*(r + n2) = 32767;
i = j = 0;
for(k = offset; k <= size; k++) {
if(*(l + i) <= *(r + j)) {
arr[k] = *(l + i);
i++;
}
else {
arr[k] = *(r + j);
j++;
}
} free(l);
free(r);
}
void mergeSort(int arr[], int offset, int size) {
if (offset < size) {
int mid = (offset + size) / 2;
mergeSort(arr, offset, mid);
mergeSort(arr, mid + 1, size);
merge(arr, offset, mid, size);
```

```
}

}
```

Output:-

```
D10A_Atharva Chavan_9

****Implementation of Menu driven Merge Sort and Quick Sort****

Enter size of array: 5
Enter values of array:
12
37
64
62
20
Enter sorting algorithm to use:
1: Bubble
2: Insertion
3: Quick sort
4:Merge sort
1
12 20 37 62 64
Continue? 1/0:  1

Enter size of array: 4
Enter values of array:
32
79
5
59
Enter sorting algorithm to use:
1: Bubble
2: Insertion
```

```
3: Quick sort
4:Merge sort
2
5 32 59 79
Continue? 1/0:  1

Enter size of array: 4
Enter values of array:
30
2
89
1000
Enter sorting algorithm to use:
1: Bubble
2: Insertion
3: Quick sort
4:Merge sort
3
2 30 89 1000
Continue? 1/0:  1

Enter size of array: 4
Enter values of array:
795

312
456
```

```
974
Enter sorting algorithm to use:
1: Bubble
2: Insertion
3: Quick sort
4:Merge sort
4
312 456 795 974
Continue? 1/0:
```