#include <string>

#include <iostream>

#include <ctime>

#include <cstdlib>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Merge Sort \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void display(int arr[], int size){ /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for(int i = 0; i<size;i++){ /\* A function to \*/

cout<< arr[i]<<" "; /\* display the array \*/

} /\* elements \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

cout<<endl;

}

int merge(int arr[], int low, int middle, int high){

int arrLeft[10],arrRight[10]; // Two temporary arrays to hold the two arrays to be merged

int sizeRight, sizeLeft, i, j, k; //i = arrLeftPtr (Array left pointer), j = arrRightPtr (Array right pointer)

sizeLeft = middle - low + 1; // sizeLeft = Size of left array

sizeRight = high - middle; // sizeRight = Size of Right array

for(i = 0 ; i < sizeLeft ; i++) //putting the values of the real array of numbers in this left splitted section in the arrLeft

arrLeft[i] = arr[low + i];

for(j = 0; j < sizeRight ; j++) //putting the values of the real array of numbers in this right splitted section in the arrRight

arrRight[j] = arr[ middle + j + 1];

arrLeft[i] = 9999; // To mark the end of each temporary array, pretend that "9999" is infinity that no number can surpass that number

arrRight[j] = 9999;

i = 0; // Returning both pointers to zero position

j = 0;

for(k = low ; k <= high ; k++) { //process of combining two sorted arrays

if(arrLeft[i] <= arrRight[j]) // if the number in arrLeft smaller than or equal put it in the real array

arr[k] = arrLeft[i++]; // then increment the arrLeftPtr to go through with the rest of the numbers in the array

else // else put the number in arrRight in the real array

arr[k] = arrRight[j++]; // then increment the arrRightPtr to go through with the rest of the numbers in the array

}

return 0;

}

int mergeSort(int arr[],int low,int high){

int middle;

if(low < high) {

middle = (low + high) / 2; // Find a middle value to divide the array

// Divide and Conquer

mergeSort(arr, low, middle);

mergeSort(arr, middle + 1, high);

// Sort and Combine

merge(arr, low, middle, high);

}

return 0;

}

int main(){

const int size = 10;

int nums[size];

/\*

for(int i = 0 ;i<size;i++){

cin>>nums[i];

}

mergeSort(nums, size , 0, size - 1);

for(int i = 0; i< size;i++){

cout<< nums[i]<<" ";

}

\*/

srand(unsigned (time(0)));

for(int i = 0 ; i< size;i++){

nums[i]= (rand() % 100) + 1;

}

cout<<"Unsorted array: "<<endl;

display(nums, size);

mergeSort(nums,0,size - 1);

cout<<"Sorted array: "<<endl;

display (nums, size);

return 0;

}

//int counter = 0;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* QuickSort \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// void quickSort(int arr[], int left, int right) {

// int i = left, j = right;

// int temp; //integer used for swapping

// int pivot = arr[(left + right) / 2]; //set pivot as the middle value in the array

//

// /\* partition \*/

//

// while (i <= j) {

// while (arr[i] < pivot){ //To keep values < pivot on the left

// i++; //Till you reach a value which is > pivot

// counter++;

// }

// while (arr[j] > pivot){ //To keep values > pivot on the right

// j--; //Till you reach a value which is < pivot

// counter++;

// }

// if (i <= j) {

// temp = arr[i]; /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// arr[i] = arr[j]; /\*Swap the values so you can have values < pivot\*/

// arr[j] = temp; /\* on the left and values > pivot on the right \*/

// /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// i++;

// j--;

// counter++;

// }

// };

//

// /\* recursion \*/

// if (left < j){ //recursion for left hand-side

// quickSort(arr, left, j);

// counter++;

// }

//

// if (i < right){ //recursion for right hand-side

// quickSort(arr, i, right);

// counter++;

// }

//

// }

//

//

//

// main(){

//

// int x[5] = {5,1,6,4,2}; //Array to be sorted

// quickSort(x,0,4); //QuickSort function call

// counter++;

//

// for(int i=0;i<5;i++){

// cout<<x[i]; //Print out the sorted array

// counter++;

// }

// cout<<endl;

// cout<<"The Counter for the whole process: "<<counter<<endl;

// }

// Thread tryouts

//void task1(string msg)

//{

// cout << "task1 says: " << msg;

//}

//

//int main()

//{

// // Constructs the new thread and runs it. Does not block execution.

// thread t1(task1, "Hello");

//

// // Makes the main thread wait for the new thread to finish execution, therefore blocks its own execution.

// t1.join();

//}