**C program Merge Sort**

#include <stdio.h>

#include <stdlib.h>

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there

are any \*/

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

/\* l is for left index and r is right index of the

sub-array of arr to be sorted \*/

void mergeSort(int arr[], int l, int r)

{

if (l < r) {

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l + (r - l) / 2;

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

/\* UTILITY FUNCTIONS \*/

/\* Function to print an array \*/

void printArray(int A[], int size)

{

int i;

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

printf("%d ", A[i]);

printf("\n");

}

/\* Driver code \*/

int main()

{

int arr[] = { 12, 11, 13, 5, 6, 7 };

int arr\_size = sizeof(arr) / sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}

**Quick sort in C**

#include <stdio.h>

// function to swap elements

void swap(int \*a, int \*b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

// function to find the partition position

int partition(int array[], int low, int high) {

// select the rightmost element as pivot

int pivot = array[high];

// pointer for greater element

int i = (low - 1);

// traverse each element of the array

// compare them with the pivot

for (int j = low; j < high; j++) {

if (array[j] <= pivot) {

// if element smaller than pivot is found

// swap it with the greater element pointed by i

i++;

// swap element at i with element at j

swap(&array[i], &array[j]);

}

}

// swap the pivot element with the greater element at i

swap(&array[i + 1], &array[high]);

// return the partition point

return (i + 1);

}

void quickSort(int array[], int low, int high) {

if (low < high) {

// find the pivot element such that

// elements smaller than pivot are on left of pivot

// elements greater than pivot are on right of pivot

int pi = partition(array, low, high);

// recursive call on the left of pivot

quickSort(array, low, pi - 1);

// recursive call on the right of pivot

quickSort(array, pi + 1, high);

}

}

// function to print array elements

void printArray(int array[], int size) {

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

printf("%d ", array[i]);

}

printf("\n");

}

// main function

int main() {

int data[] = {8, 7, 2, 1, 0, 9, 6};

int n = sizeof(data) / sizeof(data[0]);

printf("Unsorted Array\n");

printArray(data, n);

// perform quicksort on data

quickSort(data, 0, n - 1);

printf("Sorted array in ascending order: \n");

printArray(data, n);

}