|  |  |
| --- | --- |
| **EX.No. 7** | **MERGE SORT** |
| **Date : 24.01.2025** |

**AIM:**

To sort the given set of elements using Merge Sort

**ALGORITHM:**

STEP 1: Start the program

STEP 2: Call the sort( ) function with low and high values.

STEP 3: Find the mid value and Call sort( ) recursively if low<high.

STEP 4: Call merge( ) function with low, mid and high values for merging sub arrays

STEP 5: Print the values of the given array after merge( )

STEP 6: Display the values of the given array in a sorted ascending order

STEP 7: Stop the execution

**PROGRAM CODING:**

#include <stdio.h>

#include <stdlib.h>

#define max 10

int a[10] = {10, 44, 19, 96, 27, 31, 33, 35, 42, 49};

int b[10];

void merge(int low, int mid, int high) {

int l1, l2, i;

for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <=high; i++) {

if(a[l1] < a[l2])

b[i] = a[l1++];

else

b[i] = a[l2++];

}

while(l1 <=mid)

b[i++] = a[l1++];

while(l2 <= high)

b[i++] = a[l2++];

for(i = low; i <=high; i++)

a[i] = b[i];

}

void sort(int low, int high)

{

int mid;

if(low < high)

{

mid = (low + high) / 2;

sort(low, mid);

sort(mid+1, high);

merge(low, mid, high);

}

else

{

return;

}

}

int main( )

{

int i;

printf("List before sorting\n");

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

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

sort(0, max-1);

printf("\nList after sorting\n");

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

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

}

**RESULT:**

Thus the above program implemented and the given set of elements are sorted using Merge Sort successfully.

|  |  |
| --- | --- |
| **EX.No. 8** | **QUICK SORT** |
| **Date : 24.01.2025** |

**AIM:**

To sort the given set of elements using Quick Sort

**ALGORITHM:**

STEP 1: Start the program

STEP 2: Call the quicksort( ) function with low and high values.

STEP 3: Call partition( ) to find the pivot value and swap with pivot if low<high.

STEP 4: Call quicksort() function recursively for left and right subarrays

STEP 5: Print the values of the given array after quicksort( )

STEP 6: Display the values of the given array in a sorted ascending order

STEP 7: Stop the execution

**PROGRAM CODING:**

#include <stdio.h>

// function to swap elements

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

{

int t;

t = \*a;

\*a = \*b;

\*b = t;

}

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

{

int pivot, i,j;

// select the rightmost element as pivot

pivot = array[high];

i = (low - 1);

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

{

if (array[j] <= pivot)

{

i++;

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)

{

int pi;

if (low < high)

{

pi = partition(array, low, high);

quicksort(array, low, pi - 1);

quicksort(array, pi + 1, high);

}

}

// function to print array elements

void printarray(int array[], int size)

{

int I;

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

{

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

}

printf("\n");

}

int main()

{

int data[] = {8, 7, 2, 1, 70, 9, 6},n;

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

printf("Unsorted Array\n");

printarray(data, n);

quicksort(data, 0, n - 1);

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

printarray(data, n);

}

**RESULT:**

Thus the above program implemented and the given set of elements are sorted using Quick Sort successfully.