**Assignment No 2**

**Name: Abhishek Santosh Gaikwad** **Class:** TY A1

**Roll No.:** 371017 **Prn:** 22210975

**AIM:** **Implementation the following algorithm using Divide & Conquer method. (a)Merge sort (b) Quick Sort Also display execution time for different size of input and perform the analysis**

**Code:-**

**#include <bits/stdc++.h>  // Includes most of the standard library headers.**

**#include <chrono>        // Includes the chrono library for measuring time.**

**using namespace std;     // Allows using standard library names without the std:: prefix.**

**// Merge Sort Functions**

**// Merge function to combine two sorted halves of an array**

**void merge(int \*arr, int start, int mid, int end) {**

**int len1 = mid - start + 1;   // Length of the first half**

**int len2 = end - mid;         // Length of the second half**

**int \*left = new int[len1];    // Temporary array to hold the first half**

**int \*right = new int[len2];   // Temporary array to hold the second half**

**// Copying the first half into left[]**

**for (int i = 0; i < len1; ++i)**

**left[i] = arr[start + i];**

**// Copying the second half into right[]**

**for (int i = 0; i < len2; ++i)**

**right[i] = arr[mid + 1 + i];**

**int i = 0, j = 0, k = start;  // Initial indices for left[], right[] and merged array**

**// Merge the temp arrays back into arr[start..end]**

**while (i < len1 && j < len2) {**

**if (left[i] <= right[j])**

**arr[k++] = left[i++];**

**else**

**arr[k++] = right[j++];**

**}**

**// Copy the remaining elements of left[], if any**

**while (i < len1)**

**arr[k++] = left[i++];**

**// Copy the remaining elements of right[], if any**

**while (j < len2)**

**arr[k++] = right[j++];**

**delete[] left;    // Free the memory allocated for left[]**

**delete[] right;   // Free the memory allocated for right[]**

**}**

**// Recursive merge sort function**

**void mergeSort(int arr[], int start, int end) {**

**if (start >= end)  // Base case: If the array has one or no elements**

**return;**

**int mid = start + (end - start) / 2;  // Find the middle point**

**mergeSort(arr, start, mid);           // Sort the first half**

**mergeSort(arr, mid + 1, end);         // Sort the second half**

**merge(arr, start, mid, end);          // Merge the sorted halves**

**}**

**// Quick Sort Functions**

**// Partition function to place the pivot element at the right position**

**int partition(int arr[], int low, int high) {**

**int pivot = arr[high];  // Pivot element is the last element**

**int i = low - 1;        // Index of smaller element**

**// Rearrange elements based on pivot**

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

**if (arr[j] < pivot) {**

**++i;**

**swap(arr[i], arr[j]);**

**}**

**}**

**swap(arr[i + 1], arr[high]);  // Place the pivot in the correct position**

**return i + 1;                 // Return the partition index**

**}**

**// Recursive quick sort function**

**void quickSort(int arr[], int low, int high) {**

**if (low < high) {               // Base case: If there are one or no elements**

**int pi = partition(arr, low, high);  // Partition the array**

**quickSort(arr, low, pi - 1);         // Sort the elements before partition**

**quickSort(arr, pi + 1, high);        // Sort the elements after partition**

**}**

**}**

**// Function to print the array**

**void printArray(int arr[], int size) {**

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

**cout << arr[i] << " ";**

**cout << endl;**

**}**

**int main() {**

**ios\_base::sync\_with\_stdio(false);  // Disable synchronization between C and C++ standard streams**

**int size;**

**cout << "Enter the size of the array: ";**

**cin >> size;  // Input the size of the array**

**int \*arr = new int[size];  // Dynamically allocate memory for the array**

**cout << "Enter the elements of the array: ";**

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

**cin >> arr[i];  // Input the elements of the array**

**cout << "Given array: ";**

**printArray(arr, size);  // Print the original array**

**// Merge Sort**

**int \*arrCopy = new int[size];  // Create a copy of the array for merge sort**

**copy(arr, arr + size, arrCopy);  // Copy the original array to arrCopy**

**auto start\_time = chrono::high\_resolution\_clock::now();  // Record start time**

**mergeSort(arrCopy, 0, size - 1);  // Perform merge sort**

**auto end\_time = chrono::high\_resolution\_clock::now();  // Record end time**

**chrono::duration<double> merge\_sort\_time = end\_time - start\_time;  // Calculate duration**

**cout << "Sorted array using Merge Sort: ";**

**printArray(arrCopy, size);  // Print the sorted array**

**cout << "Merge Sort Time: " << merge\_sort\_time.count() << " s\n";  // Print the time taken**

**// Quick Sort**

**copy(arr, arr + size, arrCopy);  // Reset arrCopy to the original array**

**start\_time = chrono::high\_resolution\_clock::now();  // Record start time**

**quickSort(arrCopy, 0, size - 1);  // Perform quick sort**

**end\_time = chrono::high\_resolution\_clock::now();  // Record end time**

**chrono::duration<double> quick\_sort\_time = end\_time - start\_time;  // Calculate duration**

**cout << "Sorted array using Quick Sort: ";**

**printArray(arrCopy, size);  // Print the sorted array**

**cout << "Quick Sort Time: " << quick\_sort\_time.count() << " s\n";  // Print the time taken**

**delete[] arr;      // Free the memory allocated for the original array**

**delete[] arrCopy;  // Free the memory allocated for the copy array**

**return 0;  // End of the program**

**}**

**Output:-**



