**Practical No : 2**

**Name :** Samrudhi Pawar

**Roll No :** 44

**Title :** Develop a program to design a class for Concurrent Quick Sort Using Divide and Conquer Strategies. Also Compute it's time complexity.

**Program :**

#include <iostream>

#include <cstdlib>

using namespace std;

class QuickSort {

public:

// Method to perform Quick Sort

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

if (low < high) {

// Partition the array

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

// Recursively sort the two halves

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Method to partition the array

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

int pivot = arr[high]; // Pivot element

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

for (int j = low; j <= high - 1; j++) {

// If the current element is smaller than or equal to the pivot

if (arr[j] <= pivot) {

i++; // Increment index of smaller element

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

}

}

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

return (i + 1);

}

// Utility function to swap two elements

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

int t = \*a;

\*a = \*b;

\*b = t;

}

// Method to take user input for array elements

void inputArray(int arr[], int size) {

cout << "Enter " << size << " elements for the array:" << endl;

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

cin >> arr[i];

}

}

// Method to print the array

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

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

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

}

cout << endl;

}

};

int main() {

QuickSort sorter;

int size;

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

cin >> size;

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

// Take input

sorter.inputArray(arr, size);

cout << "Array before sorting: ";

sorter.printArray(arr, size);

// Perform Quick Sort

sorter.quickSort(arr, 0, size - 1);

cout << "Array after sorting: ";

sorter.printArray(arr, size);

// Deallocate memory

delete[] arr;

return 0;

}

**Output :**

Enter the size of the array: 8

Enter 8 elements for the array:

119

17

15

12

16

18

4

11

Array before sorting: 119 17 15 12 16 18 4 11

Array after sorting: 4 11 12 15 16 17 18 119 //best:o(nlogn),average:o(nlogn),worst:o(n2)