Assignment 1 , 2 :-

#include<bits/stdc++.h>

using namespace std;

using namespace std::chrono;

int partition(vector<int>& arr, int low, int high) {

int pivot = arr[low];

int i = low;

int j = high;

while (i < j) {

while (arr[i] <= pivot && i <= high - 1) {

i++;

}

while (arr[j] > pivot && j >= low + 1) {

j--;

}

if (i < j) swap(arr[i], arr[j]);

}

swap(arr[low], arr[j]);

return j;

}

void quicksort(vector<int> &arr, int low, int high) {

if (low < high) {

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

quicksort(arr, low, pivotIndex - 1);

quicksort(arr, pivotIndex + 1, high);

}

}

// Mergesort implementation

void merge(vector<int> &arr, int low, int mid, int high) {

vector<int> temp;

int left = low;

int right = mid + 1;

while (left <= mid && right <= high) {

if (arr[left] <= arr[right]) {

temp.push\_back(arr[left]);

left++;

}

else {

temp.push\_back(arr[right]);

right++;

}

}

while (left <= mid) {

temp.push\_back(arr[left]);

left++;

}

while (right <= high) {

temp.push\_back(arr[right]);

right++;

}

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

arr[i] = temp[i - low];

}

}

void mergeSort(vector<int> &arr, int low, int high) {

if (low >= high) return;

int mid = (low + high) / 2 ;

mergeSort(arr, low, mid);

mergeSort(arr, mid + 1, high);

merge(arr, low, mid, high);

}

int main() {

vector<int> values\_to\_sort;

srand(static\_cast<unsigned>(time(0)));

for (int i = 0; i < 500; ++i) {

values\_to\_sort.push\_back(rand() % 1000);

}

vector<int> values\_for\_mergesort = values\_to\_sort;

auto start\_time\_quicksort = high\_resolution\_clock::now();

quicksort(values\_to\_sort, 0, values\_to\_sort.size() - 1);

auto end\_time\_quicksort = high\_resolution\_clock::now();

auto duration\_quicksort = duration\_cast<nanoseconds>(end\_time\_quicksort - start\_time\_quicksort);

auto start\_time\_mergesort = high\_resolution\_clock::now();

mergeSort(values\_for\_mergesort, 0, values\_for\_mergesort.size() - 1);

auto end\_time\_mergesort = high\_resolution\_clock::now();

auto duration\_mergesort = duration\_cast<nanoseconds>(end\_time\_mergesort - start\_time\_mergesort);

cout << "Quicksort - Sorted values: ";

for (int val : values\_to\_sort) {

cout << val << " ";

}

cout << "\nTime taken for Quicksort: " << duration\_quicksort.count() << " nanoseconds" << endl;

cout << "\nMergesort - Sorted values: ";

for (int val : values\_for\_mergesort) {

cout << val << " ";

}

cout << "\nTime taken for Mergesort: " << duration\_mergesort.count() << " nanoseconds" << endl;

return 0;

}