**Source Code:**

#include<iostream>

#include<omp.h> #include<bits/stdc++.h> using namespace std; void sequential\_bubble\_sort(int arr[],int size){ int array[size]; for(int i = 0 ; i < size; i++){ array[i] = arr[i]; } double start = omp\_get\_wtime(); for(int i = 0; i < size - 1; i ++){ for(int j = 0; j < size - i - 1; j++){ if(array[j] > array[j+1]){ swap(array[j],array[j+1]);

}

}

} double end = omp\_get\_wtime(); cout << "Sequential Bubble Sort:\n";

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

// cout << array[i] << " ";

// }

cout << endl; cout << "Time Required: " << end - start << endl;

} void parallel\_bubble\_sort(int arr[],int size){ int array[size]; for(int i = 0 ; i < size; i++){ array[i] = arr[i];

}

double start = omp\_get\_wtime(); for(int k = 0; k < size;k ++){ if(k % 2 == 0){ #pragma omp parallel for for(int i = 1; i < size - 1; i += 2){ if(array[i] > array[i+1]){ swap(array[i],array[i+1]);

}

} } else{

#pragma omp parallel for for(int i = 0; i < size - 1; i += 2){ if(array[i] > array[i+1]){ swap(array[i],array[i+1]);

}

}

}

} double end = omp\_get\_wtime(); cout << "Parallel Bubble Sort:\n";

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

// cout << array[i] << " ";

// }

cout << endl; cout << "Time Required: " << end - start << endl;

} void merge(int array[],int low, int mid, int high,int size){ int temp[size]; int i = low; int j = mid + 1;

int k = 0; while((i <= mid) && (j <= high)){ if(array[i] >= array[j]){ temp[k] = array[j]; k++; j++; } else{ temp[k] = array[i]; k++; i++; } } while(i <= mid){ temp[k] = array[i]; k++; i++; } while(j <= high){ temp[k] = array[j]; k++; j++; } k = 0; for(int i = low;i <= high;i++){ array[i] = temp[k]; k++;

} } void mergesort(int array[],int low,int high,int size){ if(low < high){

int mid = (low + high) / 2; mergesort(array,low,mid,size); mergesort(array,mid+1,high,size); merge(array,low,mid,high,size);

} } void perform\_merge\_sort(int arr[],int size){ int array[size]; for(int i = 0 ; i < size; i++){ array[i] = arr[i];

} double start = omp\_get\_wtime(); mergesort(array,0,size-1,size); double end = omp\_get\_wtime(); cout << "Merge Sort:\n"; // for(int i = 0 ; i < size; i++){

// cout << array[i] << " ";

// }

cout << endl; cout << "Time Required: " << end - start << endl;

} void p\_mergesort(int array[],int low,int high,int size){ if(low < high){ int mid = (low + high) / 2; #pragma omp parallel sections

{

#pragma omp section p\_mergesort(array,low,mid,size); #pragma omp section p\_mergesort(array,mid+1,high,size);

}

merge(array,low,mid,high,size);

} } void perform\_p\_merge\_sort(int arr[],int size){ int array[size]; for(int i = 0 ; i < size; i++){ array[i] = arr[i]; } double start = omp\_get\_wtime(); p\_mergesort(array,0,size-1,size); double end = omp\_get\_wtime(); cout << "Parallel Merge Sort:\n";

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

// cout << array[i] << " ";

// }

cout << endl; cout << "Time Required: " << end - start << endl;

} int main(int argc, char const \*argv[])

{ int SIZE; int MAX = 1000; cout << "Enter size of array: "; cin >> SIZE; int array[SIZE]; for(int i = 0 ; i < SIZE; i ++){ array[i] = rand() % MAX;

}

// cout << "Initial Array:\n";

// for(int i = 0 ; i < SIZE; i++){

// cout << array[i] << " ";

// }

cout << endl; sequential\_bubble\_sort(array,SIZE); parallel\_bubble\_sort(array,SIZE); perform\_merge\_sort(array,SIZE); perform\_p\_merge\_sort(array,SIZE); return 0;

}